# 浅谈动态笛卡尔树问题

南京市第一中学 陈翰文

## 1 前言

本文介绍了一种新的动态笛卡尔树算法，可以在 $O(n \log n)$ 的时间内显式的维护笛卡尔树加点过程中边的变化。过去学界使用的技术较为繁琐 [1]，不适合引入 OI。本文提出的新算法实现了相同的功能，简化了过去的工作。

在应用方面，本文介绍了相关技术和 OI 经典问题的联系，并给出 Splay 复杂度的组合化证明。

## 2 问题介绍

**定义 2.1** (笛卡尔树)**.** [1] 定义 **笛卡尔树** 为一种二叉树，每一个节点由一个键值二元组 $(k, w)$ 构成．要求 $k$ 满足二叉搜索树（*BST*）的性质，而 $w$ 满足堆的性质。

特别约定，本文讨论的笛卡尔树均为 OI 术语中的“大根笛卡尔树”，定义中提到的 $k, w$ 各自互不相同，那么这棵笛卡尔树的结构是唯一的。

**定义 2.2** (动态笛卡尔树问题)**.** 有 $n$ 次操作，每次在平面中加入一个点 $(k_i, w_i)$，要求动态维护平面中点集构成的笛卡尔树，指出每轮边集的变化情况。

约定 $k_i$ 是一个 $1 \sim n$ 的排列，$w_i$ 互不相同。

单次修改笛卡尔树，边的修改量可达 $O(n)$ 级别。因而想要在低于 $O(n^2)$ 内显式维护边的变化，必然要借助摊还。故我们讨论聚焦于笛卡尔树的只加点修改。

笛卡尔树的最大断边次数的上下界都是 $O(n \log n)$，因此在**显式维护**的前提下，$O(n \log n)$ 将是我们可以做到的最好结果。

这个问题和 Splay 有诸多相似之处，本文的部分灵感来自 Splay，并最终给出一个组合化证明 Splay 复杂度的新方法。

[1] 摘自 [2] https://oi-wiki.org/ds/cartesian-tree/

## 3 动态操作中断边数的界

在笛卡尔树中断边和连边的数量始终是同阶的，不妨只分析断边数。

**定理 3.1** (最大断边数下界)**.** 动态笛卡尔树问题中最大断边数至少是 $O(n \log n)$ 量级的。

证明. 只需证在一个随机的动态笛卡尔树问题中，断边数期望是 $O(n \log n)$ 量级的即可。

考虑这样随机：考虑生成一个 $1 \sim n$ 的随机排列，从小到大依此在对应位置加入排列中的每一个元素。根据期望的线性性，总断边的期望等于每一次加点断边数的和。设 $sz$ 是当前加入的节点数，单次加边的断边数量容易证明是期望 $O(\log sz)$ 的。 □

这表明如果想要**显式维护**笛卡尔树的连边断边，$O(n \log n)$ 的复杂度是不可避免的。

值得一提的是，Splay 的随机数据下表现同样“差劲”，两者在这点上“臭味相投”。

**定理 3.2** (最大断边数上界)**.** 动态笛卡尔树问题中最大断边数至多是 $O(n \log n)$ 量级的。

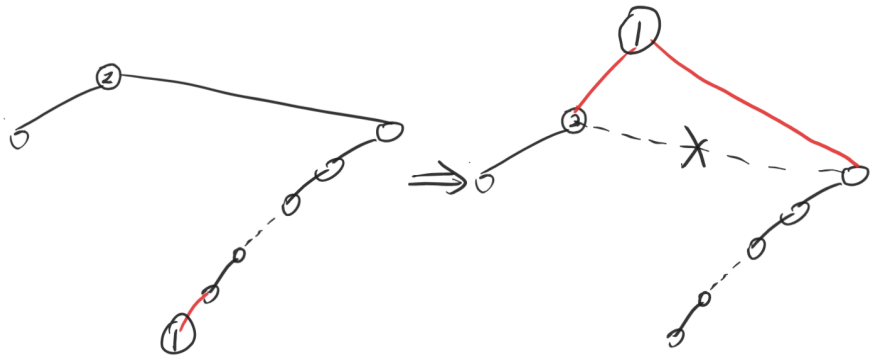
该定理非常关键，本文将使用两种不同的方法分别证明这一定理。这表明，在感性上，特殊构造的笛卡尔树和随机笛卡尔树在断边数上，没有特别大的区别。

我们沿用 Splay 的势能分析函数。

**定义 3.1** (树熵)**.** 定义**树熵** $H(T)$ 为树 $T$ 的熵，$H(T) = \sum_{i=1}^{n} \log sz_i$。

但笛卡尔树修改过程并不能直接理解为 Splay 旋转。具体来说这种错误的方法是这样的：把节点插入在笛卡尔树的最低端，然后用 Splay 的旋转浮动到合适的位置，因为为了保持笛卡尔树的“刚性”结构，应当“zig-zig”的子结构可能被迫需要“zig-zag”，破坏了原本的势能分析，伪装 Splay 旋转的算法可以被 Hack 到 $O(n^2)$，见图1。

图 1: 将 1 号点不断旋转到最上方




我们可以源源不断在最底下插入类似的一号点，并控制其权值，使其必须浮动到最上面，并在过程中仅仅断掉一条边。但此时复杂度来到了 $O(n^2)$。

所以本文引入了一种新的分析方法。

## 3.1 证明 1

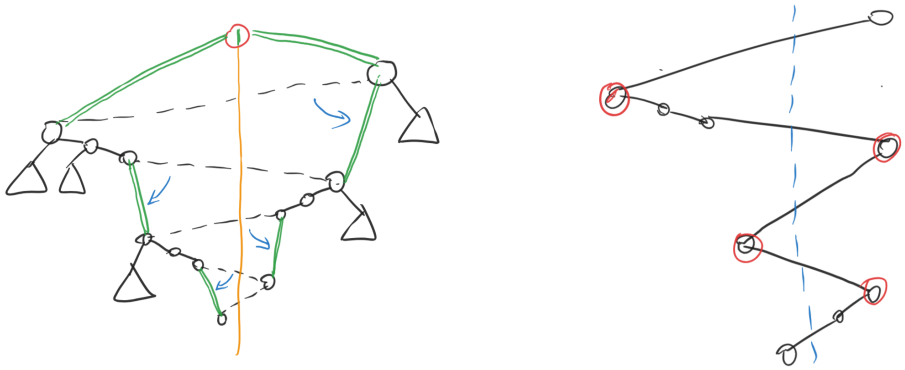
证明. 考虑使用重链剖分辅助分析势能。（这一灵感来源于 LCT 中关于虚实链的分析）

设势能函数为树熵 $H(T)$。对当前的笛卡尔树进行轻重链剖分。注意这里轻重链剖分只是一个证明中构建出来的工具，并不是真的在算法流程中维护。可以理解为每次证明都在脑海中重新重剖了整颗笛卡尔树。

分析树熵的变化。

新增了一个点，先把这个点定位到其最终落在的子树中加入，对于这个子树来说这个点比子树中的所有点都大。不失一般性，可以假装这个点比全树的任何一个点都要大。这个点本身至多增加 $\log n$ 的势能，我们首先把这 $\log n$ 的势能加入树熵。

树形态的变化必然如同图2(a)，在一条交替路径上断开一些边然后向下重新连接。图中绿色双边代表新增的边，黑色虚边代表删除的边，三角代表子树，蓝色的箭头表示每条被修改的边的旋转方向。



(a) 加入 1 号点树上的变化　　(b) 拐点示意图

图 2: (a)(b)

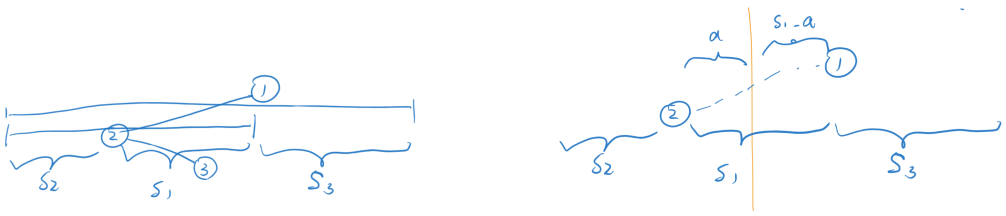
对于除了一号之外的点来说，单点的 $\log sz$ 都必然减小，只需分析减小量。由于断开的边在一条链上，最多只有 $\log n$ 条轻边。于是只需证如果断开了 $k$ 条重边，势能必定减小 $O(k)$ 即可。

不妨用一个线段的在图中横坐标的跨度代表中间跨过了多少节点。我们来考察每一个交错路径拐弯的地方，不妨称之为拐点（图2(b)红色加粗即为拐点）

不妨设某处拐点情况为图3(a)，其中 2 号点为拐点，只需证明 1 号点和 2 号点减少的势能和至少为 $O(1)$ 即可。可以钦定 (1,2),(2,3) 都是重边，即 $s_1 \geq s_2, s_1 + s_2 \geq s_3$，其中 $s_i$ 为图3(a)中笛卡尔树划分的区间长度。但这个 1 可能是另一个拐点，会导致算重（算两次），不过这在渐进意义下无关痛痒。

劈开的地方必然在 1 和 2 中间, 设 $s_1$ 被劈成了大小分别为 $a, (s_1 - a)$ 的两部分（见图3(b)），要证明的就是 $(\log(s_2 + s_1) - \log(s_2 + a)) + (\log(s_3 + s_1 + s_2) - \log(s_3 + s_1 - a)) > O(1)$。容易发

现在重链性质下这一点是对所有 $a$ 是恒成立的。




(a) 拐点特例 (b) 劈开之后

图 3: (a)(b)

简要说明一下，如果劈开位置离 2 更近，即 $2a \le s_1$，那么

$$\begin{aligned}
\log(s_2+s_1)-\log(s_2+a) &= \log(\frac{s_2+s_1}{s_2+a}) \\
&\ge \log(\frac{(s_1-s_2)+s_2+s_1}{(s_1-s_2)+s_2+a}) \\
&= \log(\frac{2s_1}{s_1+a}) \\
&\ge \log(\frac{2s_1}{1.5s_1}) \\
&= O(1)
\end{aligned}$$

其中第一处缩放利用了糖水不等式。

很好理解，$s_1$ 是 2 子树的主要成分，主要成分被砍半了自然会导致原子树下降一个常数倍数。类似的，这里再形式化的给出 1 子树的证明，如果 $2(s_1-a) \le s_2$，即 $(s_1-a) \le \frac{s_2}{2} \le \frac{s_1+s_2}{2}$：

$$\begin{aligned}
\log(s_3+s_1+s_2)-\log(s_3+s_1-a) &= \log(\frac{s_3+s_1+s_2}{s_3+s_1-a}) \\
&\ge \log(\frac{(s_1+s_2-s_3)+s_3+s_1+s_2}{(s_1+s_2-s_3)+s_3+s_1-a}) \\
&= \log(\frac{2(s_1+s_2)}{(s_1+s_2)+(s_1-a)}) \\
&\ge \log(\frac{2(s_1+s_2)}{1.5(s_1+s_2)}) \\
&= O(1)
\end{aligned}$$

故总断边数是 $O(n\log n)$ 量级的。

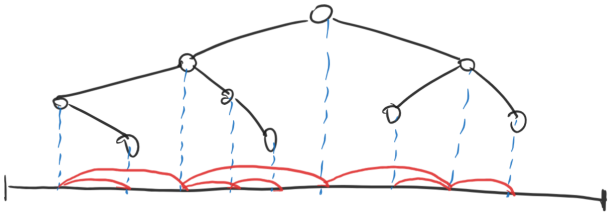
□

## 3.2 证明 2

下面我们换一种更加轻巧的方式来证明这一命题。

证明. 可以把笛卡尔树的所有边映射到 $1 \sim n$ 下标的区间上，如图4。容易发现，这些区间如果有交那么必然互相包含，这是由笛卡尔树的性质所决定的。而加入一个点实际上就是把覆盖到这个点的所有区间全部左右交替的拆分开，再新加一条边。

图 4: 将树边映射为区间



建立一颗猫树分治的线段树。把所有的区间挂到首次把这个区间劈开的节点上（即该线段树节点包含，但是两个儿子不包含）。在一个线段树节点内部，所有挂在该点的区间一定都经过某一个点，并从上往下，从大到小，堆叠在一起。如图5(a)。

(a) 线段树节点上所有区间示意图

(b) 劈开后部分区间降级示意图

图 5: (a)(b)

设势能为所有区间在这个线段树上的高度和。线段树叶子高度定义为 1，离叶子越远，高度越大。

对于一个区间来说，劈一刀，几乎必然降级（部分边界情况除外），跑到更低的线段树节点上，例如图 5(b)，红色的箭头代表这三个区间被扔到了更低的层级上去。于是势能必然降低 $O(k)$，$k$ 为断边数。

故断边数得证。

□

这个证明将引出一个简洁的 $O(n \log n)$ 维护方法。

# 4 动态笛卡尔树

## 4.1 做法 1 $O(n\log^2 n)$

$O(n\log^2 n)$ 的动态树做法较为暴力，在 [5, 6] 中已有所介绍，这里回顾一下。

考虑用 LCT 维护动态树，使得我们可以快速跳到下一个拐点。由于断边数量至多 $O(n\log n)$，暴力多一个 $\log n$ 维护 LCT 即可。复杂度 $O(n\log^2 n)$。

一个已知的公开猜想（来自 [5]）是，这个做法可以被轻微的改进，并分析到 $O(n\log n)$。笔者目前并不知道该猜想是否已被证明，如有知情读者可以联系笔者，感激不尽。

## 4.2 做法 2 $O(n\log n)$

这个做法和上述方法有较大区别。改进自证明 2。

### 4.2.1 做法 2 简单情况

简单起见，先假定每次加入的笛卡尔树的节点权值都是前缀最大值。

建立好猫树线段树后，我们在线段树的每一个节点上维护一个栈，区间长度从栈底到栈顶依此变长。当有修改操作时，直接将每一条要降级的边暴力从原来的栈中弹出，降级。注意到原本有包含关系的边，操作之后仍然有包含关系，所以在更低级别的线段树节点插入这个区间时只需在栈的末尾插入。

修改时，先从上往下定位线段树，然后从下往上逆序依此处理，从叶子到根依此处理每一个节点上的区间栈。对于每一个栈，需要被弹出的必然是一个后缀，每一个区间都暴力尝试弹出即可。

最后还会新增 $O(1)$ 条边，这些边对应的区间不被任何区间包含，直接定位到线段树上加入到栈末尾即可。

### 4.2.2 做法 2 一般情况

对于更一般的情况，每次加入的笛卡尔树的节点权值不一定是前缀最大值。

前面的绝大多数处理任然是正确的，唯一的缺点我们添加的边不一定不被任何区间包含，可能需要在栈的中间做插入，这在一般的栈中是做不到的。

于是每个节点额外维护一个平衡树，用于维护来自特殊插入的区间。通过一定的预处理使得访问平衡树的开头位置只需 $O(1)$ 的代价，删除则正常 $O(\log n)$ 的代价。删除可以摊还到插入上，复杂度不用担心。

只需保证，任何区间一旦下穿给儿子，一定被**放回**栈中即可。具体来说，每次下传前，把弹出平衡树的区间和弹出栈的区间做一下归并排序, 统一下穿给儿子即可。

值得注意的是，这个算法虽然复杂度只有 $O(n\log n)$，但是并不能维护树上的 ddp 相关信息。

# 5 应用

## 5.1 OI 例题分析

**例题 5.1.** [CF1290E] Cartesian Tree[2]

给定一个长度为 $n$ 的排列，按照排列中数字从小到大的顺序依此在排列中插入数字。每次操作结束求笛卡尔树子树的大小和。

$1 \le n \le 150000$

动态笛卡尔树模版，加一个使用 LCT 维护子树大小和的 ddp 即可。复杂度 $O(n\log^2 n)$。

不过此题由于只需要求大小和这种简单的线性信息，根据线性性可以把贡献拆开，进而使用吉司机线段树维护，详情可见 [4]。

若此题题意更改为求“子树大小的平方和”，恐怕吉司机线段树的方法就不再适用，此时便可展示动态笛卡尔树的威力。对于动态笛卡尔树来说，只需重新设计 ddp 的标记即可，复杂度依然是 $O(n\log^2 n)$。

**例题 5.2.** [Ynoi2010] Worst Case Top Tree[3]

给定序列 $a_0, a_1, a_2 \ldots, a_n, a_{n+1}$；满足 $a_0 = a_{n+1} = +\infty$，$a_1, a_2, \ldots, a_n$ 在输入中给出；

对 $1 \le x \le n$，称 $\max_{0\le i<x, a_i \ge a_x} i$ 和 $x$ 是 **相邻** 的，且 $\min_{x<i\le n+1, a_i > a_x} i$ 和 $x$ 是 **相邻** 的；如果 $x$ 和 $y$ 相邻，则 $y$ 和 $x$ 也相邻；

如果 $0 \le b_1, b_2, b_3, b_4, b_5, b_6 \le n+1$，且 $b_i$ 和 $b_{i+1}$ 相邻，$b_1$ 和 $b_6$ 相邻，$b_i$ 互不相同，则称集合 $\{b_1, b_2, b_3, b_4, b_5, b_6\}$ 是一个六元环（即判断两个六元环是否相同时，不考虑 $b_i$ 的顺序）。

共有 $m$ 次修改操作，每次修改操作给出 $x\ y$，将 $a_x$ 改为 $a_x + y$；

每次修改后要求输出六元环的个数。

$1 \le n, m \le 5 \cdot 10^5;\ 1 \le x \le n;\ 1 \le a_i, y \le 10^9$。

首先建大根笛卡尔树，根据一定的性质分析（性质分析与本题关联不大，这里略去，可见 [6]），注意到笛卡尔树上的每个大小为 4 的连通块，加上联通块根节点的左右父亲必定构成一个六元环，且不存在其他可能构成方式。而大小为 4 的连通块数量是一个局部信息，可以每次在笛卡尔树的局部爆搜更新，不需要使用 LCT 维护。于是只需要显式维护出笛卡尔树的结构即可做到 $O(n\log n)$。

[2] https://codeforces.com/contest/1290/problem/E

[3] https://www.luogu.com.cn/problem/P6107

单点加和加入一个点没有本质区别，在使用“做法 2”时可能需要在栈中删除一个点。但是这容易通过在“特殊插入平衡树”的基础上，额外维护“特殊删除平衡树”做到。复杂度 $O(n \log n)$。

**例题 5.3.** [uoj456] 许愿树和圣诞树[4]

给定一颗 $n$ 个点的动态树，你需要维护以下 *3* 种操作：

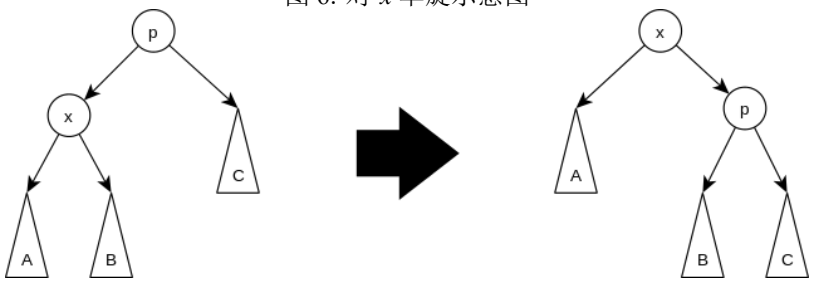
*1.* 点 $x$ 连续不断单旋向上，直至它成为点 $y$ 的儿子，保证 $y$ 是 $x$ 的祖先。

*2.* 询问点 $x$ 的第 $k$ 级祖先，保证此时点 $x$ 的第 $k$ 级祖先存在。

*3.* 询问整棵树先序遍历中的第 $k$ 个点，保证此时 $1 \le k \le n$。

单旋的定义和 *Splay* 相同，如图*6*。

图 6: 对 $x$ 单旋示意图




单旋向上实际上就是等价于在笛卡尔树中把这个点的权值增加。所以问题可以转化为笛卡尔树单点加。可以在 $O(n \log n)$ 的时间内显式维护树形态。我们额外套一个 LCT，即可在 $O(n \log^2 n)$ 的时间内回答询问。

对于询问 2 只需要在 LCT 中把该点 access 到根，查询路径上的第 $p$ 个就行了。询问 3 可以在 LCT 上从上往下定位找到这个点，找到后不要忘记将这个点 access 上去保证复杂度。

**例题 5.4.** [WC2022] 秃子酋长 & [Ynoi2024] 魔法少女网站第二部[5]

给一个长为 $n$ 的排列 $a_1, \ldots, a_n$，有 $m$ 次询问，每次询问区间 $[l, r]$ 内，排序后相邻的数在原序列中的位置的差的绝对值之和。

$1 \le n, m \le A$，$1 \le a_i \le n$，$a_i$ 互不相同，$1 \le l \le r \le n$

在《秃子酋长》中 $A = 5 \times 10^5$。

在《魔法少女网站第二部》中 $A = 2 \times 10^6$。

这个问题乍一眼看上去和动态笛卡尔树没有什么关系，但是实则两者存在隐秘且深刻的联系。

[4] https://uoj.ac/contest/47/problem/456

[5] https://www.luogu.com.cn/problem/P8078 & https://www.luogu.com.cn/problem/P11367

首先不加解释的抛出一些直觉：把询问放到二维平面（$l-r$ 平面）上，可以证明，平面上的贡献可以被拆分为 $O(n\log n)$ 个矩形加，而每个矩形加都恰好对应了一次动态笛卡尔树问题的断边。

下面将严谨说明这一事实。

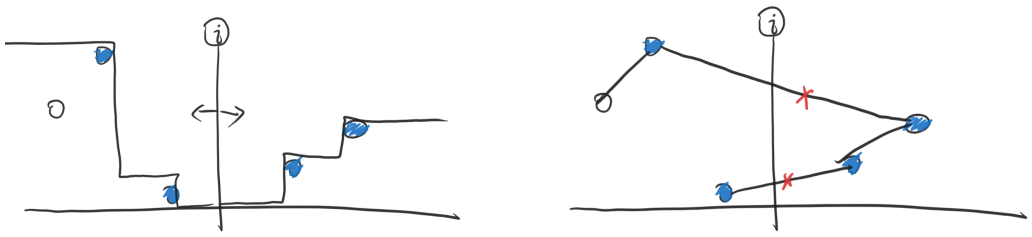
我们把一个区间的答案，看成区间中所有数和它的前驱下标差的绝对值的和（前驱指在区间中比它小的最大的数）。考虑对绝对值拆贡献。

我们从数的角度对询问做贡献。对于 $a_i$ 来说，它关心的是其前驱在排列中，下标和它的大小关系，而不关系其前驱的具体数值。即对于 $|a_i-a_j|$ 来说，我们固定住 $a_i$，只计算 $+a_i$ 或 $-a_i$ 的贡献。$a_j$ 的贡献可以再通过同样的方法统计每个数和其后继的贡献。下文只讨论 $a_i$ 和其前驱的贡献。

把排列中所有 $<a_i$ 的数留下，$a_i$ 只关心和它在一个区间中小于它的最大值，也就是从它的位置开始，往前看和往后看的前后缀 max，见图 7(a)，实心的蓝点即为 $i$ 往前往后看到的前后缀 max。

进一步的观察是（见图8），“成段”的前后缀 max 可以看做一个整体（看成更靠近 i 的那个点），“成段”的定义是一段连续在一侧的前/后缀 max 构成的值域区间中不存在另一侧的后/前缀 max。

因为我们只关心 max 在 $i$ 的哪一侧，一个连续段中，由于其值域区间不包含另一侧的前缀 max，所以不论在这一段中包含多少个都不会改变 max 在哪一侧的事实。



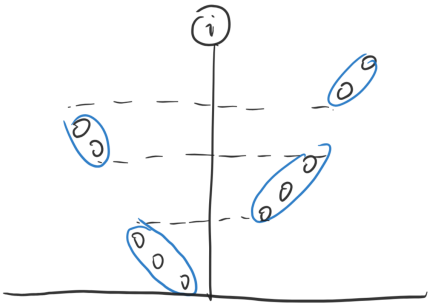

(a) $i$ 前后缀 max 示意图　　(b) $i$ 前后缀 max 笛卡尔树示意图

图 7: (a)(b)

而前后缀 max 的连续段发生变化恰好对应了 $<i$ 的节点构成的笛卡尔树跨过 $i$ 的边。（见图7(b)）每一个前后缀 max 的连续段的变化都对应了一个二维平面矩形加。将所有矩形加离线扫描线即可 $O(n\log^2 n+q\log n)$ 解决此题。当 $n,q$ 同阶时可以多叉平衡做到 $O(\frac{n\log^2 n}{\log\log n})$。

后记：秃子酋长存在一个优雅的 $O(n\sqrt{q})$ 的莫队做法，有兴趣的读者可查阅 [8]。而在魔法少女网站第二部通过扩大数据范围将该做法卡掉了，有关多叉平衡树的支线内容可以在 [7] 中找到。

图 8: $i$ 前后缀 max 连续段示意图

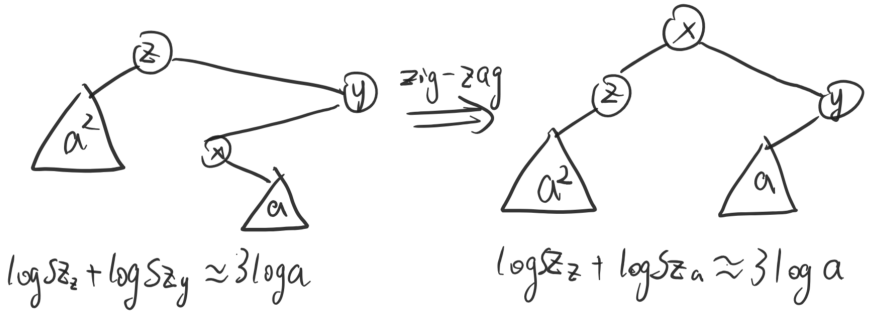


## 5.2 Splay 势能新分析

我们仍然沿用树熵作为 Splay 的势能函数。

Splay 势能分析的核心在于分析批量 zig-zig 和 zig-zag 对势能的影响。一个自然的想法是证明每次操作必然让势能下降 O(1)。可这个加强的结论实际上是假的，如果 Splay 操作花了 $O(k)$ 的时间，消耗了 $O(k)$ 的势能，并不意味着每一次旋转势能都降低了 $O(1)$（Hack 见图 9，三角中的数字表示子树大小，当 $a$ 足够大时，$z, y$ 的势能和变化小到几乎可以忽略不计）。需要把一次 Splay 看做一个整体分析 [3]，这相当的麻烦且不直观。

图 9: zig-zag 势能变化反例



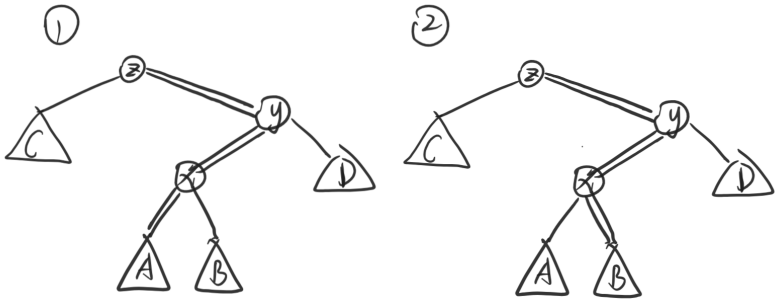

考虑对当前的 Splay 重链剖分。（这里重剖只是一个分析工具，并不是真的在算法中维护重剖）

在旋转的过程中，祖先链上的轻重链情况是不会发生变化的。故遇到最多 $\log n$ 次在 $(x, y), (y, z)$ 中存在轻边的情况。可以将这些情况直接计入复杂度，从而钦定 $(x, y), (y, z)$ 之间都是重边。接下来我们分类讨论 zig-zag 和 zig-zig 两种情况。

对于 zig-zag 再分以下两种情况讨论（图10），A 为重儿子或 B 为重儿子。

注意到，当 $A$ 是重儿子时，zig-zag 完 $\log sz_y$ 必然下降的 $O(1)$，因为 $y$ 失去了它子树的重要成分 A（由重链性质可知，$4sz_A + O(1) \geq sz_y$）。同理，当 B 是重儿子是 $\log sz_x$ 必然下降

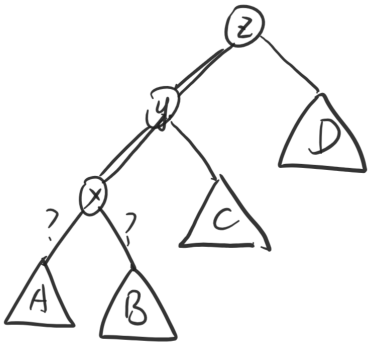
图 10: zig-zag 分类讨论图




$O(1)$，因为 $z$ 失去了它子树的重要成分 B。

类似的，讨论 zig-zig 的情况。（图11）此时不再需要分类讨论 A,B 子树的重量关系，$z$ 子树必然会同时失去他们两个，这两者由于重链性质必然是 $z$ 子树的重要成分，故 $\log sz_z$ 必然下降 $O(1)$。

图 11: zig-zag 分类讨论图




于是我们就完整证明了 Splay 的复杂度为 $O(n \log n)$！

## 6 总结

本文对数据结构中的动态笛卡尔树问题的一些笔者的新成果进行了介绍。

笔者注意到这一类问题还没有在信息学竞赛中被广泛的知晓和应用，相关算法的资料更是寥寥无几。或许是因为过去的技术无论是在理论层面还是代码层面，都过于粗暴和困难。希望本文提出的新算法可以起到抛砖引玉的作用，让读者更好地认识动态笛卡尔树问题，感受到树上摊还类问题的独特魅力。

期待这一优雅技术在信息学竞赛中有更好的应用。

# 7 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢家人对我的陪伴与支持。
感谢南京市第一中学韩英老师的关心与指导。
感谢李欣隆同学、彭亦宸同学、叶李蹊同学、吴航同学与我交流并提出建议。
感谢王一策同学对本论文写作过程中的特殊支持。
感谢其他给予我帮助的老师与同学。

# 参考文献

[1] Iwona Bialynicka-Birula and Roberto Grossi，《Amortized Rigidness in Dynamic Cartesian Trees》

[2] Oiwiki 笛卡尔树 https://oi-wiki.org/ds/cartesian-tree/

[3] Oiwiki Splay 树 https://oi-wiki.org/ds/splay/

[4] CF1290E 题解 https://www.luogu.com.cn/problem/solution/CF1290E

[5] UOJ Easy Round 8 题解 https://whzzt.blog.uoj.ac/blog/4749

[6] [Ynoi2010] Worst Case Top Tree 题解 https://www.luogu.com.cn/problem/solution/P6107

[7] [Ynoi2024] 魔法少女网站第二部题解 https://www.luogu.com.cn/problem/solution/P11367

[8] [WC2022] 秃子酋长题解 https://www.luogu.com.cn/problem/solution/P8078

[9] 刘承奥《北校门外的未来》命题报告

# 近最优的在线范围修改查询算法与莫队转移代价上下界

广州市第二中学 陈翔乐

## 摘 要

本文建立了莫队与范围修改查询问题之间的联系，并给出了近最优的在线范围修改查询算法和莫队转移代价上下界的新结果。

## 1 前言

范围修改查询问题是一个经典问题，例如 K-D tree 是解决高维矩形修改查询问题的最优范围划分树。

莫队算法（Mo's Algorithm）是一个经典算法，最初被用于解决一类静态区间查询问题，后来被拓展到更多的范围，如树链、双子树、半平面等。我们希望知道各种范围上的莫队算法的转移代价的上下界。

本文将建立上述两个问题的联系，并给出一些新的结果。

本文在第 2,3 节介绍莫队与范围修改查询问题；第 4 节建立了莫队与范围修改查询问题之间的联系；第 5 节给出了一般莫队问题的转移代价忽略对数因子的上下界；第 6 节给出了一种在线范围修改查询问题的代数结构运算次数忽略对数因子最优的算法。此前据 Optimal partition trees[3] 的作者 Timothy M. Chan 所说，在线的圆范围修改查询只有时间复杂度为 $\Theta(n^{5/3})$ 的做法，而我们得到了一个时间复杂度 $\tilde{O}(n^{3/2})$ 的近最优的算法。

## 2 莫队算法介绍

莫队算法最初被用来解决这个问题：

**例题 2.1 (小 Z 的袜子[1])** 给出一个序列 $a_{1\ldots n}$，有 $m$ 次询问，每次询问给出一个区间 $[l,r]$，计数满足 $l \le i < j \le r \wedge a_i = a_j$ 的二元组 $(i,j)$ 的数量。

假设任意两个询问区间互不相同，即 $m \le n^2$。

[1]有改动

先离线读入所有的询问，然后考虑将询问的顺序重排列并维护一个由当前区间中的所有数组成的可重集合，动态维护这个集合中相等点对的数量。依次扫描重排列后的询问，假设正在考虑第 $i > 1$ 个区间 $[l_i, r_i]$，集合此前维护的是 $[l_{i-1}, r_{i-1}]$ 中的数，因此需要删除和加入一些元素。显然，删除和加入的总量不超过 $|l_i - l_{i-1}| + |r_i - r_{i-1}|$。

维护集合是简单的，只需要维护一个数组 $cnt_x$ 表示 $x$ 的出现次数以及答案。因此，需要使得 $|l_1 - r_1| + \sum_{i=2}^{n} |l_i - l_{i-1}| + |r_i - r_{i-1}|$ 较小。

先介绍一些经典的构造方法。下面称将集合从表示一个区间修改为另一个区间的过程为“转移”或“移动”，过程中删除和加入元素的总量为“转移代价”。

**算法 2.1 (二维莫队构造方法 1)** 取 $B = n/\sqrt{m}$，将所有询问以 $l/B$ 为第一关键字，$r$ 为第二关键字排序。即对序列分块，先将询问按照左端点所属块归类，每一类再按照右端点排序。

这样左端点的移动要么在一块内，代价不超过 $B$；要么跨越出块，总代价为 $O(n)$。对于每一块，右端点的移动代价为 $O(n)$。因此，总代价为 $O(mB + n^2/B) = O(n\sqrt{m})$。

这是最常用的经典算法。同时可以发现将第偶数个块的右端点由升序排序改为降序可以避免相邻块之间右端点从最右侧直接跳到最左侧的无意义移动，减小常数。

**算法 2.2 (二维莫队构造方法 2)** 考虑一张 $m$ 个点的完全图，点 $i$ 与点 $j$ 之间的边权为其转移代价 $|l_i - l_j| + |r_i - r_j|$。不难发现找一条最短且遍历了所有点的路径是最优的，即旅行商问题。不难验证边权满足三角不等式，同时将区间看作一个平面上的点 $(l_i, r_i)$，边权就是曼哈顿距离。此时求出这张图的最小生成树，其 dfs 序的代价一定不超过最小生成树边权和的两倍，所以也不超过最优解的两倍。根据算法 2.1，最优解的代价是 $O(n\sqrt{m})$ 的，所以这个构造也是 $O(n\sqrt{m})$ 的。

**算法 2.3 (二维莫队构造方法 3[2])** 仍然考虑构造平面路径，直接将点按照其在希尔伯特曲线上的顺序排序。

[2]https://codeforces.com/blog/entry/61203

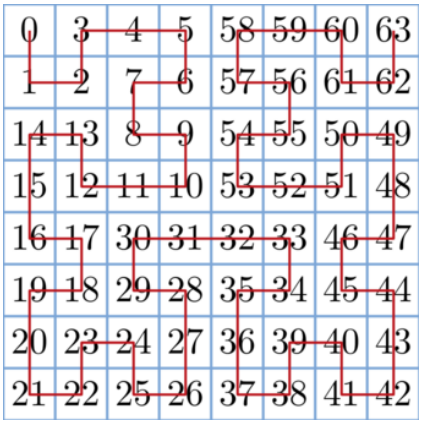



图 1: 希尔伯特曲线，图片来自原博客

取阈值 $B = \log(n/\sqrt{m})$，简单分析该分形曲线在 $2^B \times 2^B$ 上层和下层的代价可知转移总代价是 $O(n\sqrt{m})$。

同时，可以给出一组使得转移代价为 $\Omega(n\sqrt{m})$ 的数据。这只要使得任意两点之间的边权都是 $\Omega(B)$，也就是在序列上等间距取 $\sqrt{m}$ 个点，询问两两点构成的区间即可。

这样就完成了对序列区间范围，即二维莫队的转移代价上下界的分析。

## 2.1 莫队的定义

**定义 2.1 (范围)** 一个范围是 $\{1, 2, \ldots, n\}$ 的某些子集构成的集族。

每种范围的莫队都被定义在一类满足某种条件的范围上。

**定义 2.2 (莫队转移代价)** 给定范围中的 $m$ 个集合，将其重排列得到 $S_1, S_2, \ldots, S_m$，其转移代价为 $|S_1| + \sum_{i=2}^{n} |S_{i-1} \oplus S_i|$，$\oplus$ 表示对称差。

**引理 2.1** 考虑 $i, j$ 之间连边为 $|S_i \oplus S_j|$ 的完全图，该图的边权满足三角不等式即 $w(i, j) \leq w(i, k) + w(k, j)$。

证明. 这是对称差显然具有的性质。 □

因此直接求出最小生成树就可以得到转移代价达到下界的莫队构造，但时间复杂度过高无法接受。

OI 中将 $d$ 维空间中的偏序关系构成的范围的莫队简称为 $d$ 维莫队，本文将采用这一称呼。OI 中流行的所谓“dsu on tree”技巧就是子树莫队，对序列前缀的扫描线可以视作一维莫队。

接下来考虑一个更复杂的问题。

**例题 2.2 (双子树交莫队[3])** 根据定义，表述为：给出两棵编号 $1 \sim n$ 的树，范围是所有子树对中编号的交，即范围中每个集合对应一组 $(x, y)$ 使得该集合是第一棵树 $x$ 子树和第二棵树 $y$ 子树中包含的编号的交。不妨假设给出的集合互不相同。

直接给出一种转移代价上界为 $O(n\sqrt{m})$ 的构造方法。其下界 $\Omega(n\sqrt{m})$ 是容易构造的。

**算法 2.4 (双子树（交）莫队构造方法 1)** 我们直接构造双子树莫队，即集合是两子树中结点的并。这样在插入和删除结点时，可以判断集合中该编号的存在性。于是这样也保证了子树交的转移代价。

取块长 $B = n/\sqrt{m}$，利用 Top Cluster 树分块将树分解为若干大小为 $\Theta(B)$ 的块，再将每一块收缩为一个点，得到一棵规模为 $\Theta(n/B)$ 的树。可以证明，对于一棵大小为 $l$ 的树，当给定集合是范围中所有 $l^2$ 个集合时，存在构造方式的转移代价为 $O(l^2)$。对收缩后的树这样构造的转移代价是 $O((n/B)^2 \times B) = O(n^2/B) = O(n\sqrt{m})$，同时每个集合在块内的转移总代价是 $O(mB) = O(n\sqrt{m})$。

考虑证明这一点。对树轻重链剖分，设 $x$ 的重儿子为 $son_x$，特别地，若 $x$ 不存在重儿子则设 $son_x = 0$；设点 $x$ 的子树大小为 $size_x$，特别地，设 $size_0 = 0$。对于集合 $(x, y)$，选择 $(x, son_y)$ 和 $(son_x, y)$ 中到其转移代价较小的一个集合向其连边，即选择 $size_y - size_{son_y}$ 和 $size_x - size_{son_x}$ 中较小的一边。这样 $l^2$ 个点构成了一个外向树森林，对每棵树求 dfs 序即可。

此时的转移代价不超过所有 $\min(size_y - size_{son_y}, size_x - size_{son_x})$ 的和的两倍，考虑分析该和的量级。对于任意 $k \geq 0$，可以证明 $2^k \leq size_v - size_{son_v} < 2^{k+1}$ 的 $v$ 不超过 $l/2^k$ 个。这是因为若这样的 $v$ 构成祖先关系，则与重儿子的定义矛盾。因此简单计算等比数列求和可知该求和是 $O(l^2)$ 的。

## 2.2 不删除莫队

有时候撤销向集合中加入元素的影响是简单的，但任意删除元素并不简单，此时可以使用不删除莫队。

不删除莫队形如给定范围中的 $m$ 个集合，通过若干次如下操作维护一个集合，使得所有给定的集合都在某一时刻等于维护的动态集合：

1) 在集合中加入一个元素

2) 撤销上一次未被撤销的操作 1

转移代价定义为进行的操作次数。

将不删除莫队加入和撤销的过程建一棵树，即每个叶子代表一个集合，每个叶子到根的路径上的所有元素即为该集合。取叶子构成的虚树，就得到了不删除莫队的版本树。

[3] 双子树莫队的原题为 https://www.luogu.com.cn/problem/P9988，有改动

**定理 2.1** 对某一范围，若存在转移代价为 $O(T)$ 的莫队，则存在转移代价为 $O(T\log m)$ 的不删除莫队。

证明. 使用线段树分治算法即可。 □

可以证明在一些情况下该对数因子是必须存在的。如，取 $m=n/2$，每个集合是序列上形如 $[i,i+n/2]$ 的区间，其莫队的转移代价显然是 $O(n)$，但其不删除莫队的转移次数下界为 $\Omega(n\log n)$。

证明. 来自 [2]。通过调整，可以说明最优的不删除莫队每个时刻的集合都是一个区间的形式。考虑转移过程中长度为 $l\le n/2$ 的区间的出现次数，由于每个 $[i,i+n/2]$ 的区间中都要包含至少一个长度为 $l$ 的区间，因此相邻两个长度为 $l$ 的区间的左端点之间的距离不超过 $n/2-l+1$。计算调和级数求和可知转移过程至少包含 $\Omega(n\log n)$ 步。

□

## 3 范围修改查询问题

**例题 3.1 (矩形修改查询)** 给定平面上的 $n$ 个点 $(x_i,y_i)$，每个有初始权值 $v_i$。你需要维护 $m$ 次操作，每次操作为给出一个矩形，求矩形中所有点的权值和对 $2^{64}$ 取模的值；或给出一个矩形和常数 $c$，将矩形中所有点的权值乘上 $c$。

更一般地，将加法与乘法拓展为任意分配双半群。

这个问题最广为人知的解决方法是使用 K-D tree。下面将介绍两种其他的算法。

**算法 3.1 (四分树)** 该结构十分简单，但笔者此前从未得知。笔者发现其在对坐标离散化外的建树复杂度是线性，且结构简明直观，因此具有更强的拓展性。

考虑对 $x,y$ 轴分别分块，即将横/纵坐标排序后，将排序数组以块长为 $\sqrt n$ 分块。这样将平面划分为了 $\sqrt n\times\sqrt n$ 的网格，每个网格中落有一些点，用一棵四分树维护这个 $\sqrt n\times\sqrt n$ 平面。即，一个结点代表一个矩形，其四个儿子为将其从两维的中间切割开的四个小矩形。而递归到单个网格后，可以对网格内部的点任意建树。这样得到了一棵和通常使用的 K-D tree 功能相同但结构不同的树。

对于四分树，可以将其视作点集是 $\sqrt n\times\sqrt n$ 平面上所有整点构成的特殊的 K-D tree，因此在网格上递归的时间复杂度是正确的。但有另一种更好的刻画方式：将四分树每层的结点取出，每层的结点代表的矩形构成一个划分平面的网格，且每层网格的边长取出来是一个等比数列。一个矩形在某一层经过的结点数是与其有交的网格数，即其周长除以网格边长，等比数列求和自然得到其经过的总结点数为 $O(\sqrt n)$。

而在底层网格内部的点，其所在网格是两行区间加两列区间的形状。而一开始的分块方式保证了每行和每列的网格中点的总和是 $O(\sqrt n)$，因此底层点递归的时间复杂度也是 $O(\sqrt n)$。

除排序外建树的时间复杂度是线性的，当然也可以简单地拓展到高维。

沿用范围的定义，可以定义任意范围的修改查询。修改查询问题同样定义在一类范围上。下文中约定 $n$ 为元素个数，$m$ 为修改查询的集合数量总和。

这里给出对范围的进一步定义。

**定义 3.1 (对偶范围)** 对于范围中若干个集合 $S_1, S_2, \dots, S_m$，定义其对偶范围：对偶范围包含 $n$ 个集合，$\forall 1 \le j \le n$，第 $j$ 个集合为 $S'_j = \{i | j \in S_i\}$。

**定义 3.2 (等价类)** 对于范围中若干个集合 $S_1, S_2, \dots, S_m$，元素 $j, k$ 同属一个等价类当且仅当 $S'_j = S'_k$。

**定义 3.3 (范围的等价类数)** 对于一个范围，在其中选取 $x$ 个集合，将得到的不同的 $S'$ 数量的最大值记为 $P(x)$。

下文中用 $P'$ 表示某个范围的对偶范围的等价类数函数。不失一般性地，设 $P(m) = n$。

**算法 3.2 (等价类分治 [1])** 等价类分治是一种解决离线修改查询问题的算法。

考虑维护一个动态森林。森林的结构类似线段树，每个结点维护一个等价类中的信息以及懒标记。过程中会动态地加入一个根连接两棵树，或将一个点的标记下传后删除这个点。[4]

每次处理一段长度为 $B$ 的操作序列，对操作序列分治，同时在分治到区间 $[l, r]$ 的时候，森林中每棵树都是一个区间 $[l, r]$ 中集合构造出的等价类。这是因为处理这个区间时，同一个等价类中的元素受到的操作是完全相同的。递归到 $[l, mid]$ 和 $[mid + 1, r]$ 时，新建若干个根合并一些此前的等价类，处理完后再将这些根的标记下传并删除。直到递归到 $l = r$ 时只有至多两个等价类，也即森林只有两棵树，作为该集合的那棵树就是我们所想要的了。

总代数结构运算次数为 $O(m/B \times (P(B) + 2P(B/2) + 4P(B/4) + \dots))$。

例如解决上述矩形修改查询问题，取 $B = \sqrt{n}$，$P(x) = O(\min(x^2, n))$，所以 $(P(B) + 2P(B/2) + 4P(B/4) + \dots) = O(n)$，代数结构运算次数为 $O(m/B \times n) = O(m\sqrt{n})$。而其中合并等价类的过程不增加复杂度，因此时间复杂度相同。

**定义 3.4 (范围划分树)** 定义范围划分树为一类有根二叉 leafy tree。其恰好有 $n$ 个叶子，每个叶子对应范围的集合中 $\{1, 2 \dots, n\}$ 的元素，即对应一个单元素集合。每个非叶子结点代表其子树中叶子的集合（这集合不一定在范围中）。

一个范围中集合 $S$ 的代价定义为：在树上找到一个最小的包含根的连通块使得连通块的叶子对应集合的并恰好为 $S$，该连通块的大小。将此连通块的叶子称为这个范围的分解。

一个集合在分解时进行的代数结构运算次数不超过其代价乘常数。

**定义 3.5 (离线范围划分树)** 根据给出的范围中的集合 $S_1, S_2, \dots, S_m$ 构造的范围划分树，其代价总和定义为 $S_1, S_2, \dots, S_m$ 的代价总和。

[4]实际上，线段树或平衡树都可以被描述为这个模式。

**定义 3.6 (在线范围划分树)** 给出一个范围，其的一个在线范围划分树的代价定义为范围中所有集合代价的最大值。

如 K-D tree、四分树或线段树都是在线范围划分树；等价类分治不是范围划分树。

# 4 莫队与范围修改查询问题的关系

**定理 4.1** 莫队的转移代价与其对偶范围的离线范围划分树的代价总和在忽略对数因子的情况下相同。

证明. 若某一范围存在总代价为 $T$ 的离线范围划分树，那么其叶子按 dfs 序排序就是其对偶范围转移代价为 $O(T)$ 的莫队。因为考虑将在每个范围的分解上挂上这个范围，在 dfs 进入子树时加入，出子树时撤销，就可以发现转移代价不超过 $2T$。实际上，这给出的是一个不删除莫队。

若某一范围存在总代价为 $T$ 的莫队构造，那么可以得到每个元素在莫队的范围序列中出现的若干个区间，区间个数总和是 $O(T)$。所以对范围序列建线段树，就是一个对偶范围上总代价为 $O(T\log m)$ 的离线范围划分树。 □

同时还可以发现不删除莫队版本树的概念和其对偶范围的离线范围划分树是相似的，但代价为连通块的叶子个数而非大小，这里可能会产生对数代价。如一维不删除莫队的转移代价为 $O(n)$，但其对偶范围的前缀范围修改查询问题需要用线段树解决。

线段树分治是时间维上的区间对偶范围的不删除莫队。

根据这一理论，可以给出二维莫队和双子树（交）莫队使用其对偶范围的范围划分树的构造算法。

由于等价类分治并不是范围划分树，所以无法据此构造莫队。

**算法 4.1 (二维莫队构造方法 4)** 序列区间（二维）范围的对偶是平面的矩形范围，其中区间 $[l_i, r_i]$ 对应平面点 $(l_i, r_i)$，第 $i(1 \le i \le n)$ 个矩形的左下角是 $(1, i)$，右上角是 $(i, n)$。

于是可以使用 K-D tree 或四分树。但不对 $x, y$ 分块，直接对平面构建四分树并保留范围内存在点的四分树结点的复杂度仍然正确。因为范围中矩形的特殊性，按 $\sqrt{m} \times \sqrt{m}$ 划分格子后，每行和每列的格子只会被经过 $\sqrt{m}$ 次。因此总代价是 $O(n\sqrt{m})$。

注意到四分树存在 dfs 序恰好是希尔伯特曲线！

**算法 4.2 (双子树交莫队构造方法 2)** 这是一种比方法 1 更加简单的做法。

考虑其对偶范围的修改查询：元素是若干双树的点对，对于每个编号，其对应的修改查询集合是所有在两棵树上的点都是其祖先的点对。

考虑对两棵树带权轻重链剖分，每个点的权值为包含这个点的点对数。对于每对重链，若这对重链上存在至少一个点对，则对其建立 K-D tree，横纵坐标为点对在两棵树上距离链

顶的距离。最后把 K-D tree 按照重链链顶在 dfs 序中的位置构成的二元组的字典序排序连接起来建线段树，可以得到一棵在线范围划分树。

可以证明：对于一个范围，在其 $O(\log^2 m)$ 对重链（即其在两棵树上到根的重链的笛卡尔积）的 K-D tree 上分解的代价总和为 $O(\sqrt{m})$。根据（带权）轻重链剖分的性质，与自上而下第 $i$ 条重链有交的点对数不超过 $m/2^{i-1}$。因此在第一棵树的第 $i$ 条重链与第二棵树的第 $j$ 条重链上的点对数不超过 $m/\max(2^{i-1}, 2^{j-1})$，也即其 K-D tree 的点数上界。有

$$\sum_{0\le i,j\le \log m} \sqrt{m/\max(2^{i-1}, 2^{j-1})} = O(\sqrt{m})$$

从而分解代价总和为 $O(\sqrt{m})$。

# 5 莫队的通用构造算法与转移代价的上下界

**定理 5.1 (莫队转移代价下界)** 莫队的转移代价下界为 $\Omega(n \max_{1\le i\le n} P'(i)/i)$。

证明. 设上式在 $i = B$ 时取到最大值。

存在其对偶范围的大小为 $B$ 的集合族（即原范围的元素）满足其能将元素，即原范围的集合划分为 $P'(B)$ 部分。

将这些元素复制 $n/B$ 遍，任意两个等价类之间的转移代价都 $\ge n/B$。得证。 □

下面将给出一种忽略对数因子代价总和最优的离线范围划分树。

笔者将这类数据结构命名为 RT 树，以纪念笔者的一位朋友。如下的数据结构称为原型 RT 树，原型 RT 树主要用于构造莫队和分析莫队的转移代价，且实现简单。

## 5.1 原型 RT 树

**算法 5.1 (原型 RT 树的构造)** 初始保留所有的集合。将所有元素代表的叶子结点作为最底层，视作初始的 $n$ 个等价类。

接下来进行若干轮，每轮将每个集合以 $1/2$ 概率保留进入上一层。对每个新的等价类建立结点，将其内部的下层等价类建线段树并将其作为根。直到不存在剩余的集合。

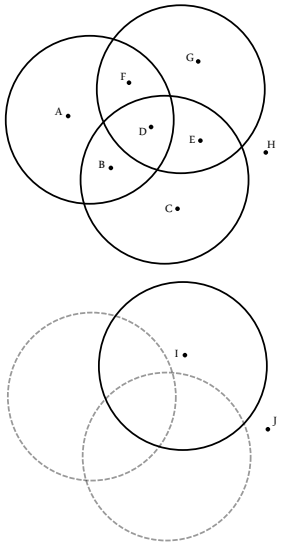



图 2: 原型 RT 树的构造过程

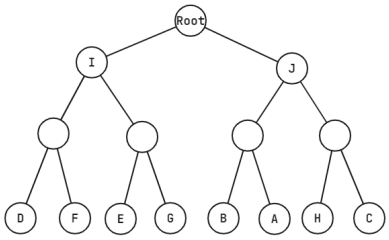



图 3: 图 2 过程对应的原型 RT 树

考虑分析原型 RT 树的代价总和。

类似四分树的分析方法，一个集合在原型 RT 树上递归时，会递归到与之有交但不包含的等价类上，再在以这个等价类为根的线段树上递归。

事实上有

**引理 5.1** 对于每个集合以 $1/x$ 的概率保留构成的等价类，一个等价类的代价以高概率不超过 $O(cnt \times x \log^2 n)$，$cnt$ 表示该等价类内部的下层等价类个数。

证明. 考虑线段树叶子的 dfs 序，也就是下层等价类建线段树前的序列顺序。

可以证明：包含相邻两个等价类的不同的集合个数以高概率不超过 $Cx \ln n$。因为这两个等价类在本层同属一个等价类，而若有 $Cx \ln n$ 个不同的集合，那么随机选中任何一个都

会导致这两个等价类在本层被划分开。没有选中任何一个的概率为 $(1-\frac{1}{x})^{Cx\ln n}\le n^{-c}$。

根据 union bound，每个集合在 dfs 序上出现和消失的总次数以高概率为 $O(cnt\times x\log n)$，再加上线段树拆分区间，一个等价类的代价即以高概率不超过 $O(cnt\times x\log^2 n)$。

□

设第 $i$ 层保留了 $c_i$ 个集合。那么从第 $i$ 层递归到第 $i-1$ 层（设叶子是第 0 层）的总代价就是 $O(P(c_{i-1})2^i\log^2 n)$。我们知道 $c_{i-1}\sim m/2^{i-1}$，因此可以将上式写为 $(P(c_{i-1})/c_{i-1})\times m\log^2 n/2^{i-1}\times 2^i=O(m\log^2 n(P(c_{i-1})/c_{i-1}))$。

但是在构造莫队时，引理 5.1 的证明中线段树带来的对数因子实际上不存在。因此构造莫队的代价以高概率为 $O(n\log m\sum_{i\ge 0}P'(c_i)/c_i)$，显然也是 $O(n\log n\log m\max_{1\le i\le n}P'(i)/i)$。

根据概率方法的道理，由于这组构造的代价以高概率为 $O(n\log n\log m\max_{1\le i\le n}P'(i)/i)$，因此存在构造方法使得实际代价为 $O(n\log n\log m\max_{1\le i\le n}P'(i)/i)$。结合定理 5.1 给出的下界，可以确定莫队转移代价的上下界。

**定理 5.2 (莫队之理)** 莫队的转移代价为 $\tilde{\Theta}(n\max_{1\le i\le n}P'(i)/i)$。

经过测试，该构造可以通过双子树莫队的原题。

**例题 5.1 (圆对偶莫队)** 给出平面上的 $n$ 个圆 $C_i$，范围中的集合 $S_j=\{i|(x_j,y_j)\in C_i\}$。

这是曾经 open 的莫队构造问题，该问题此前只有将其转化为三维单纯形的代价为 $O(nm^{2/3})$ 的构造方法，但直接使用利用原型 RT 树的构造算法即可得到一组转移代价为 $O(n\sqrt{m}\log n)$ 的构造。

随机化带来的对数因子在一些情况下可以被证明必须存在，如任意子集范围的莫队；在一些情况下可以被证明不存在，如高维莫队。因此对此对数因子的分析仍需要研究。

# 6 在线范围修改查询算法

**引理 6.1 (范围修改查询问题的代数结构运算次数下界)** 范围修改查询问题的代数结构运算次数的下界为 $\Omega(m\max_{1\le i\le m}P(i)/i)$。

该引理源于 [1]，原论文中给出的界是取 $P(i)=n$ 的情形，这里做了一点加强。证明只需要将原定理的证明的最后一部分的参数稍加修改。

这也说明了等价类分治在任何情况忽略对数因子的代数结构运算次数最优性。

原型 RT 树难以被简单的重构等方法在线化，且理论上关于下界有高达三个对数因子。这里给出一种在线化、去随机化且理论上关于下界只有两个对数因子的 RT 树。注意这棵树并不符合在线范围划分树的定义，而是一棵动态结构的范围划分树。

## 6.1 在线化与去随机化的 RT 树

维护一棵动态结构的范围划分树。

沿用类似原型 RT 树的结构，对于第 $i$ 层，维护一个集合族 $V_i$ 满足 $|V_i| \le m/2^i$，$V_0 \supset V_1 \supset V_2 \supset ...$，初始为空。第 $i$ 层的结点即代表 $V_i$ 构成的等价类。而对于每个等价类包含的若干下层等价类结点构成的树，建动态开点线段树的结构，使得其高度不超过 $\log n$。

在分解一个范围，从一个等价类遍历如上的动态开点线段树结构向下层递归时，记录递归时的非终止结点数量，将第 $i$ 层的非终止结点数记为 $e_i$。显然，该层的代数结构运算次数不超过终止结点数，即不超过 $e_i + 1$。

分解后，找到最大的 $i$，满足 $e_i > P(m/2^{i-1}) \log n/(m/2^{i+1})$，将该范围插入 $V_{0,1,...,i}$。插入时会进行一些线段树分裂以分裂等价类。

此时大于 $i$ 的层的代数结构运算次数不超过 $\log n \sum_{j>i} P(m/2^{j-1})/(m/2^{j+1})$，所有操作的此类代价总和不超过 $O(m \log n \sum_{i\ge 0} P(m/2^i)/(m/2^i))$；不超过 $i$ 的层的代数结构运算次数被线段树分裂控制，不会带来额外贡献。

第 $i-1$ 层至多有 $P(m/2^{i-1})$ 个等价类（$i = 0$ 的情况则认为每个元素本身是一个等价类，即有 $n$ 个等价类），因此线段树的结点总数不超过 $P(m/2^{i-1}) \log n$。线段树分裂会增加至少 $e_i$ 个结点，所以每个 $i$ 作为满足条件的最大的 $i$ 的次数不超过 $P(m/2^{i-1}) \log n/(P(m/2^{i-1}) \log n/(m/2^{i+1})) = m/2^{i+1}$。$V_i$ 中的集合只会来自不低于 $i$ 的层，因此有 $|V_i| \le \sum_{j\ge i} m/2^{j+1} \le m/2^i$，因此每次操作后仍然限制条件！

因此该算法的代数结构运算次数为 $\tilde{O}(m \max_{1\le i\le m} P(i)/i)$。

可以看出底层较为特殊。一个优化是并不需要维护多个满足 $P(m/2^i) = n$ 的层，因此可以删去从零开始的若干层。

## 6.2 在线圆范围修改查询

从莫队转移代价的视角看，该 RT 树的构造严格优于原型 RT 树的构造，构造不删除莫队时也不存在额外的对数因子。但其需要维护树结构上动态的集合分裂，比维护静态集合困难。一些情况下，可以在树结构发生变化时暴力重构。

对于圆，有 $P(x) = O(\min(x^2, n))$。因此只需要维护 $m/2^i \le \sqrt{n}$ 的层，这样加入的圆的数量为 $\tilde{O}(m/\sqrt{n})$。而在递归时，需要判断是否子树中所有点都在圆内或圆外，这相当于给定平面点集合，每次给一平面点，查询距其最近和最远的点，这可以用 Voronoi 图和最远点 Voronoi 图实现，同时需要支持在线平面图点定位，其构造时间复杂度为线性对数。

每次暴力重构最坏情况下要重构整棵树，由于树高为 $O(\log n \log m)$，重构的时间复杂度即为关于子树大小和的对数，即 $\tilde{O}(n)$。

总时间复杂度为 $\tilde{O}(m\sqrt{n})$。

# 7 总结

本文的关键结论是定理 5.2（莫队之理）和存在一种在线代数结构运算次数近最优的数据结构。定理 5.2 确定了任意范围的莫队转移代价的一个较紧的上下界，并给出了一种简单高效的构造算法，将曾经 open 的莫队问题做到了近最优；在线化的 RT 树作为一种代数结构运算次数近最优的范围划分数据结构，将圆范围修改查询问题的时间复杂度做到了近最优，并成为理解任意范围修改查询问题的有力工具。

# 8 致谢

感谢李欣隆学长为本文提供的一些例题与思路，给予我启发。
感谢蔡承泽学长的帮助，感谢陈奕帆、范璟阳同学为本文审稿。
感谢 Timothy M. Chan、莫涛等前辈的贡献。
感谢中国计算机学会提供学习和交流的平台。
感谢伴我走过 OI 之路的所有人。

# 参考文献

[1] Chengze Cai and Xinlong Li. Offline optimal range query and update algorithm. 2021.

[2] Chengze Cai and Xinlong Li. 数据结构艺术. 2022.

[3] Timothy M. Chan. Optimal partition trees. In 26th Annual Symposium on Computational Geometry, pages 1–10, 2011.

# 浅谈图论问题中的 Johnson 思想

重庆南开（融侨）中学校 何宇翔

## 摘 要

图论 Johnson 思想的核心是设置节点势能，通过节点势能调整边权，将复杂图论问题转化为更易求解的形式。本文探讨该思想的核心内涵，并研究其拓展应用。首先基于经典的 Johnson 全源最短路算法归纳核心思想；随后重点研究其在动态图问题中的应用与拓展，包括费用流、动态负环判定、边权修改量较小情况下的最短路优化等场景，并总结或提出相应的高效求解方法；最后针对单次负环判定问题介绍了 Goldberg 算法，该算法在效率上显著优于传统 SPFA 算法，且实现较为简便，在信息学竞赛中具有较高的实用价值。本文通过具体例题，结合笔者自身的思路，展示了 Johnson 思想在上述场景与应用中的灵活性与优越性。

## 1 Johnson 思想

### 1.1 全源最短路 Johnson 算法

Donald B. Johnson 在 1977 年发表论文，论文中提出一种适用于有负权图的全源最短路（即求出所有点对之间的最短路长度）的算法 [1] [2]，其核心流程如下：

首先对于全图执行一次 SPFA 算法，以所有节点为起点，初始的距离数组 $d$ 的值任意。若图中不存在负环，则可以根据 SPFA 算法的结果调整距离数组 $d$。图中任意单向边 $u \to v$ 权值为 $w$，简记为 $(u, v, w)$，SPFA 算法结束后的距离数组 $d$ 满足关系：

$$d_u + w \geq d_v$$

由于所有关系都是相对的，后文称每个节点的 $d$ 值为该节点的势能。此处的势能可以类比物理学中的势能，其物理意义是点 $u$ 的基础距离为 $d_u$，在势能 $d$ 下的距离加上 $d_u$ 即为真实距离。因此，在当前势能 $d$ 下，边 $(u, v, w)$ 的调整后的边权称为势能边权 $w'$，定义如下：

$$w' = d_u + w - d_v \geq 0$$

相应的，当前势能 $d$ 下，节点 $u$ 到节点 $v$ 的最短路长度 $\mathrm{dis}'(u,v)$ 和原图最短路长度 $\mathrm{dis}(u,v)$ 的关系满足：

$$\mathrm{dis}(u,v) = \mathrm{dis}'(u,v) - d_u + d_v$$

由于调整后的边权 $w'$ 非负，在当前势能下计算最短路可以使用比 SPFA 更加高效的 Dijkstra 算法，再用势能下的最短路长度反推得到原图的最短路长度。

综上，Johnson 全源最短路算法通过“单次 SPFA 求势能 + 多次 Dijkstra 求最短路”的流程实现了全源最短路的求解。本文中 Dijkstra 算法复杂度默认是 $O(n\log n + m)$，则 Johnson 算法的时间复杂度是 $O(n^2\log n + nm)$。相较于其他全源最短路算法，如 Floyd 算法、枚举起点多次执行 SPFA 算法等，该算法效率更优，可应用于最小直径生成树等问题场景。

## 1.2 Johnson 思想

Johnson 算法核心思想在于，通过势能的设置调整边权，将含负权的图转化为非负权图，从而简化求解过程。尽管该算法最初是针对全源最短路问题而提出，但其思想具有极强的通用性，可拓展至更多图论问题中。

除最短路求解外，Johnson 思想同样适用于负环判定问题：若能计算出一组势能 $d$ 使每条有向边 $(u,v,w)$ 都满足 $d_u + w \geq d_v$ 的关系，则图中没有负环；反之，若无法找到一组满足条件的势能 $d$，则图中一定存在负环。

# 2 动态图 Johnson 思想

在静态图场景下，Johnson 思想的应用空间相对有限。而在图会动态变化的问题中，Johnson 思想只需要调整势能即可适配变化，此时更能体现出 Johnson 思想的广泛适用性。

## 2.1 费用流

下文所述费用流问题 [3] 中，默认源点记为 $s$，汇点记为 $t$，默认求解目标为最小费用最大流。

### 2.1.1 原始对偶费用流

传统 EK 费用流算法需多次通过 SPFA 算法求解最短路以寻找增广路，效率较低。原始对偶费用流借助 Johnson 思想，通过势能调整将边权转化为非负值，再用 Dijkstra 算法代替 SPFA 算法，无论是理论时间复杂度还是实际运行效率都得到了显著优化。

与 Johnson 全源最短路算法类似，在原始对偶费用流算法的初始阶段同样需要在残量网络上执行一次 SPFA 算法以得到每个点的初始势能 $d$；算法后续流程与 EK 费用流算法类

似，每次需要在残量网络上寻找到一条 $s$ 到 $t$ 的最短路作为增广路，减少增广路上每条边的容量、增加每条反向边的容量；重复执行这个过程直到无法找到新的增广路。与 EK 费用流的区别在于，原始对偶费用流每次找最短路的过程将 SPFA 算法替换为在势能 $d$ 下使用 Dijkstra 算法。

但是每次增广后，反向边的容量可能从 0 增加至正数，即残量网络原本不包含这条边，增广后会添加该边。此时若不调整势能 $d$，添加的这条边的边权在原势能下可能为负数，导致后续增广无法继续使用 Dijkstra 算法。为此，每次增广后需要按如下策略调整势能：设增广边为 $(u, v, w)$，其反向边权值为 $-w$，新势能 $d'$ 需使增广边和反向边都满足条件，即：

$$d'_u + w \geq d'_v$$
$$d'_v - w \geq d'_u$$

联立两式可得：

$$d'_u + w = d'_v$$

由于增广的边一定在 $s$ 到 $t$ 的最短路上，即 $d'_u$ 恰为 $d'_s$ 加上 $s$ 到 $u$ 在原图上的最短路长度。由于势能是相对的，令 $d'_s = 0$，而 $d'_u$ 即为增广前原图上 $s$ 到 $u$ 的最短路长度。不难发现这样调整能满足所有边的限制。

设 $\mathrm{dis}(s, u)$ 为势能 $d$ 下 Dijkstra 算法得到的从 $s$ 到 $u$ 的最短路长度，结合势能定义，增广前的最短路长度即为原势能 $d$ 的值加上在势能 $d$ 下的最短路长度，因此新势能可表示为：

$$d'_u = d_u - d_s + \mathrm{dis}(s, u)$$

若无需保证调整后 $d'_s = 0$，其他节点势能为从 $s$ 出发的最短路长度，则上式可以省略 $-d_s$ 项。也可以在算法最开始时的 SPFA 过程中通过初值的设置来保证初始势能 $d_s = 0$，进而在后续增广过程中 $d'_s$ 也恒为 0。

每次增广后按此式调整势能，即可持续使用 Dijkstra 算法完成增广。该算法时间复杂度是 $O(nm + k(n \log n + m))$，其中 $k$ 表示增广次数。

### 2.1.2 ZKW 费用流

ZKW 费用流同样基于 Johnson 思想，流程与原始对偶费用流类似，都是先计算初始势能，每次增广后调整势能。

初始阶段 ZKW 费用流也需要执行一次 SPFA 算法得到每个点的初始势能 $d$。与原始对偶算法不同的是，此处需要以 $s$ 作为唯一起点，此时 $d$ 即为 $s$ 到各节点的最短路长度。此举作用在于，在此时的势能 $d$ 下从 $s$ 沿调整后边权为 0 的边到达 $t$ 的路径即为增广路，增广后添加的反向边的势能边权也为 0，也就是增广操作不会产生负权边。

当 $s$ 无法通过当前势能边权为 0 的边到达 $t$ 时，ZKW 调整势能的方式与原始对偶完全不同。设节点集合 $S$ 为从 $s$ 出发通过边权为 0 的边能到达的所有点，节点集合 $T$ 为不可达

的所有点，即 $T = V \setminus S$ 。为了使汇点 $t$ 变得可达，需要将一条从 $S$ 中节点指向 $T$ 中节点的边的势能边权降为 0 。若存在多条这样的边则选择势能边权最小的一条，若不存在这样的边则说明已经满流。形式化地，令

$$\Delta d = \min_{(u,v,w):u\in S,v\in T} w$$

将 $T$ 集合中的点势能整体加上 $\Delta d$ ，调整后的边权依然非负，且新的可达节点集合 $S$ 一定有新节点加入。重复执行该过程，直到 $t$ 被加入到集合 $S$ ，则可以进行下一轮增广。

ZKW 费用流的时间复杂度为 $O(nm + k(n + m))$ ，其中 $k$ 为势能调整次数与增广次数的总和。在极端情况下，每次增广后可能需要 $O(n)$ 次势能调整才能进行下一轮增广。实际运行中，ZKW 费用流、原始对偶费用流、EK 费用流各有优劣。

需要补充说明一点，以上两个费用流算法的核心优化集中于最短路维护，找到增广路后的增广过程本质是最大流问题，可以通过多路增广等策略进一步优化，与 Johnson 思想的关系较小，故不展开论述。

### 2.1.3 动态加删边费用流

**例题 2.1.** 给一个 $n$ 个点，$m$ 条有向边的图，边有容量和费用，容量和费用均为整数。执行 $q$ 次修改操作，每次会加入或删除一条容量为 1 的边。每次修改后都需要计算以 1 为源点 $n$ 为汇点，总流量为 $k$ 的最小费用，$k$ 为一个固定的整数。题目保证任意时刻源点到汇点的最大流不小于 $k$ ，任意时刻不存在负环。

数据范围：$n \le 10^3$ ，$m, k, q \le 10^4$ 。

本题数据范围较大，可使用原始对偶费用流解决。

在执行第一次修改前，先求初始状态的最小费用，时间复杂度是 $O(nm + k(n \log n + m))$ 。

对于删除一条边 $(u, v, w)$ 的操作：因为其容量为 1 ，若删除前该边有流量经过，删边后需在残量网络上以 $u$ 为源点 $v$ 为汇点执行总流量为 1 的增广，即使用 Dijkstra 算法求最短路，增广并调整势能。这里的源汇点虽有变化，但势能调整方法不受影响。

对于添加一条边 $(u, v, w)$ 的操作，一个错误的方法是：先从 $t$ 到 $s$ 执行一次增广，使总流量减少 1 ，即退流操作；加边后再从 $s$ 到 $t$ 执行一次增广，使总流量增加 1 。该做法在加边后可能会遇到负环导致无法找出增广路，其核心在于，费用流算法依赖增广路长度单调不减的性质，以确保每次寻找增广路时不会出现负环，若在算法运行过程中直接添加边，可能导致残量网络中出现负环。

加边操作正确的做法是：不执行退流操作，加边前先以 $v$ 为起点执行 Dijkstra 算法并调整势能。调整后可以通过势能直接计算从 $v$ 到 $u$ 的最短路长度为 $l = d_u - d_v$ ，考虑新加入的边权 $w$ 与 $l$ 的关系：

- 若 $l + w \ge 0$ ，那么该边当前的势能边权 $w' = d_u + w - d_v = l + w \ge 0$ ，图中所有边的流量都不需要修改；

- 否则，该边与上述最短路形成的流量为 1 的环流，需要增广这个环流，使原题意义下源点汇点间流量保持为 $k$ 不变而总费用减小。增广后，该边容量为 0 不影响势能，该边的反向边 $(v,u,-w)$ 当前的势能边权为 $w'=d_v-w+d_u=-l-w>0$，所以增广后不用再调整势能。

这个算法通过上述势能调整策略，可以时刻保证每条边的势能边权都非负。每次修改操作需要执行 1 次 Dijkstra 算法，时间复杂度是 $O(n\log n+m+q)$；$q$ 次修改操作的总时间复杂度是 $O(q(n\log n+m+q))$，可以通过此题。

本题作为例题，仅展示了 Johnson 思想在费用流动态调整中的一种应用场景，在费用流相关的其他问题中 Johnson 思想还能有更广泛的拓展空间。需注意的是，费用流问题本身具有特殊性：一条边和其反向边的权值互为相反数，这使得本题中的势能调整策略不一定能迁移到其他动态图问题当中。

## 2.2 动态判负环

**例题 2.2** (QOJ3223 稍有修改和加强)**.** 给定整数 $k$，设 $p$ 是长度为 $2^k$ 的 *01* 串（下标从 0 开始，下同）。如果所有长度为 $10^9$ 且前 $k$ 位和后 $k$ 位都是 0 的 *01* 串 $S$，均满足：

$$\sum_{i=0}^{10^9-k} p[f(S,i)]-\sum_{i=0}^{10^9-1} S_i=0$$

则称 $p$ 是好的。其中：

- $S_i$ 表示 $S$ 中的第 $i$ 位，$S_i\in\{0,1\}$
- $f(S,i)$ 表示 $S_i,S_{i+1},\ldots,S_{i+k-1}$ 作为由高到低位拼成的 $k$ 位二进制数，即

$$f(S,i)=\sum_{j=0}^{k-1}2^{k-j-1}S_{i+j}.$$

现在有一个长度为 $2^k$ 的 *01* 串 $q$，求字典序最小的长度为 $2^k$ 的 *01* 串 $s$，使 $p=s\oplus q$ 是好的。其中 $\oplus$ 表示逐位异或。

数据范围：$2\le k\le 20$。

本题从 01 字符串的角度难以入手，需要先将原问题转化成一个图论问题。根据 01 串 $p$ 构建一张有向图，图中有 $2^{k-1}$ 个节点，编号 0 到 $2^{k-1}-1$。节点 $i$ 向外连两条出边，一条指向节点 $(2i \bmod 2^{k-1})$ 边权为 $p_{2i}$；另一条指向 $((2i+1) \bmod 2^{k-1})$ 边权为 $p_{2i+1}-1$。这两条边的意义分别是 01 串 $S$ 中的考虑宽度为 $k$ 的窗口，窗口当前二进制数是 $i$，右移一步后窗口新的二进制数作为指向的节点，这一步对原题等式左侧产生的贡献为 $p_i$ 和下个二进制位的差，作为边权。

图 1 中是 $k=4$ 的例子。这样得到的有向图，边权只可能是 $\{-1,0,1\}$ 中的值，且 $p$ 序列每一项和图中每条边一一对应。全图是强连通的，且任意两点 $i$ 到 $j$ 之间都存在边数不超过 $k$ 的路径。

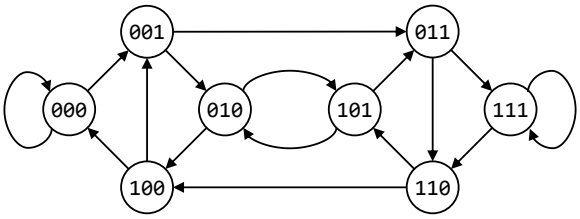



图 1: $k = 4$ 时如图所示，图中有 8 个节点 16 条边。

建图后原题对 $p$ 序列的限制等价于：从图中节点 0 出发走任意步数后回到节点 0 的路径长度一定为 0。结合图强连通的性质，所以又等价于图中所有环长均为 0 。此时再对图中每条边构建一条边权为相反数的反向边，则原问题又等价于新图上不存在负环。

对于符合题目要求的序列 $p$ ，图中从节点 0 到每个节点 $i$ 的所有路径长度都相等，设为 $d_i$ ，则任意有向边 $(u, v, w)$ 都满足

$$d_v - d_u = w$$

序列 $p$ 中未确定的项满足 $0 \le p_i \le 1$ ，可以视为差分约束限制。具体而言，$p_i$ 对应的边是

$$u = \left\lfloor \frac{i}{2} \right\rfloor$$
$$v = i \bmod 2^{k-1}$$
$$w = p_i - (i \bmod 2)$$

那么差分约束的限制就是

$$0 - (i \bmod 2) \le d_v - d_u \le 1 - (i \bmod 2)$$

题目求解要求字典序最小，需要按 $i$ 从小到大逐位确定 $s_i$ 。当某一位 $s_i$ 确定时，$p_i$ 也随之确定，$p_i$ 对应的正反两条边的权值也由此确定，这会改变 $d_v - d_u$ 不等式的上界或下界，使差分约束的限制从不等式变成等式，图中与之对应的变化则是将 $p_i$ 对应边自身权值或者反向边权值减少 1 。若此时图中出现负环，则说明将 $s_i$ 设置为当前值时会导致 $p$ 序列无合法解。对每一位 $s_i$ 先尝试令 $s_i = 0$ ，如果出现负环则改成 $s_i = 1$ ，最终确定整个 01 串 $s$ 。这个过程中需要执行 $O(2^k)$ 次负环判定算法，图的节点数和边数规模也是 $O(2^k)$ ，若每次都重新判负环则效率太低无法通过此题，需要设计效率更高的动态判负环的算法。

借用 Johnson 思想，设置势能 $d_i$ 为 0 号点到点 $i$ 的最短路长度。由该图的性质可知 $d_i \in [-k, k]$ 。接下来对于每一次边权变化，都需要快速判断能否执行修改，若执行修改则还需要高效维护势能的变化。

当有向边 $(u, v, w)$ 边权要减少 1 时，需要先判断出减少后是否形成负环，若出现负环，则此次修改不能执行。由于势能调整后的边权都非负，修改这条边产生负环需要同时满足以下两个条件：

1. 这条边的势能边权从 0 变成 $-1$；
2. 图中存在从 $v$ 到 $u$ 只经过势能边权为 0 的边的路径。

这等价于，只考虑势能边权为 0 的边时，节点 $u$ 和 $v$ 属于同一个强连通分量。

由于本题构图具有特殊性，按 $i$ 从小到大的顺序确定边权，恰好是从节点 0 逐渐向外扩展；每当确定一个 $i$，这条边的不等式限制变成等式，其正反两条边的势能边权都会被调整为 0。因此修改 $(u,v,w)$ 时节点 $u$ 与 $v$ 中至少有一个点和节点 0 同属于一个强连通分量。若能动态维护节点 0 所属强连通分量 $S_0$ 包含的点，即可快速判定修改是否执行。

由于势能即为从 0 出发的最短路，所以从起点 0 出发只经过势能边权为 0 的边可以到达图中所有点，因此，若从节点 $i$ 出发只经过势能边权为 0 的边也能到达 0，则 $i \in S_0$。而节点在加入 $S_0$ 后不会退出，所以维护 $S_0$ 的具体方法是：每当出现势能边权为 0 的边 $(u,v)$ 时，先判断 $v$ 是否在 $S_0$ 中，若 $v \in S_0$，则从 $u$ 开始在反图上沿势能边权为 0 的边搜索，忽略已加入 $S_0$ 的点，将遍历到的点加入到 $S_0$。每个点只会被搜索遍历一次，所以维护 $S_0$ 的总时间复杂度为 $O(2^k)$。

若边 $(u,v,w)$ 修改后的势能边权为 $-1$，则需要调整势能：从 $v$ 出发在图中正向搜索，只经过势能边权为 0 的边，将本次遍历到的每个点的势能都减 1。不难发现这样调整后的势能仍然符合本题势能定义，等于从 0 出发到 $i$ 的最短路长度。注意到整个算法流程中每个点的势能只会减少不会增加，而势能的值域为 $[-k,k]$，整个算法势能变化总量为 $O(2^k \cdot k)$。

本题最终算法的总时间复杂度为 $O(2^k \cdot k)$。本题做法利用了题目本身的许多特殊性质，从而使最终复杂度较低。本题作为铺垫，后续的一些例题虽然没有这些特殊性质，但求解的算法是相似的。

**例题 2.3.** 给一张 $n$ 个点 $m$ 条有向边 $(u_i,v_i,w_i)$ 的图，保证图没有负环。执行 $q$ 次修改，每次给出 $x,y$，表示将第 $x$ 条边的权值 $w_x$ 加上 $y$（$y$ 可能为负数），每次修改后都需要在线的判定图中是否存在负环。

数据范围：$n \le 10^3$，$m,q \le 10^4$，保证任意时刻 $|w_i| \le 10^{14}$。

为了应用 Johnson 思想，第一步需先执行一次 SPFA 算法，得到每个节点的初始势能，此时所有边的势能边权都非负。

在图中首次出现负环前，只需要在每次修改边权后调整势能。若修改操作的 $y \ge 0$，即边权增大，修改前的势能仍然满足要求，无需修改势能。若修改操作的 $y < 0$，则势能可能需要修改。设本次操作修改的有向边为 $(u,v)$，以 $v$ 为起点执行 Dijkstra 算法，将每个点的势能调整为 $v$ 到该点的最短路长度。调整势能后节点 $v$ 出发到任意点的最短路长度都等于势能之差，且最短路上所有边的势能边权均为 0。设势能调整和修改边权后，边 $(u,v)$ 的势能边权为 $w'$。若 $w' < 0$ 则图中一定出现了负环；否则执行本次边权修改操作后，所有边的势能边权仍然非负。

当图中产生负环后，无论如何设置势能也不能使所有边的势能边权都非负，上述势能调整后本次修改的边的势能边权仍然是负数，不再额外处理。这样的负边会持续存在，随

着后续操作依次执行而不断积累，也可能在后续操作中被消除，这不仅会影响判定图中是否存在负环，甚至会影响上述势能调整过程中 Dijkstra 算法的执行，因此，上述势能调整方法需要改进。

若当前图中存在势能边权为负数的边，基于边 $(u,v)$ 调整势能的时候，从 $v$ 出发的 Dijkstra 算法过程要忽略掉所有负边，并调整势能。忽略所有负边可能会使一些本该可达的点变得不可达，在本次势能调整中，可达点 $x$ 的新势能 $d'_x$ 等于该点原势能 $d_x$ 减去 $v$ 的原势能 $d_v$ 再加上原势能边权下的最短路长度 $\mathrm{dis}(v,x)$，即 $d'_x = d_x - d_v + \mathrm{dis}(v,x)$。不可达点的最短路长度应视为正无穷，因为可达点势能的值域绝对值不超过 $(n-1)\cdot\max(|w_i|)$，所以可以设置常量 $\mathrm{inf} = \max(n)\cdot\max(|w_i|)$，不可达点的新势能为 $d'_x = d_x - d_v + \mathrm{inf}$。

以上改进后的势能调整方法不会新增负边。可以通过分类讨论证明，具体考虑每条调整前非负的边 $(x,y)$：

- $x,y$ 本次都是可达点：则该边会被 Dijkstra 中使用，调整后仍然非负；
- $x,y$ 本次都不可达：两个点的势能增量相等，调整前后的势能边权不变；
- $x$ 可达 $y$ 不可达：调整前不可能出现这类非负边；
- $x$ 不可达 $y$ 可达：调整后的势能边权相比调整前增加 $\mathrm{inf} - \mathrm{dis}(v,y)$，调整后必然非负。

题目中判定负环的需求，可以使用改进后的势能调整方法对当前势能下的负边进行逐一消除，若能全部消除则图中无负环，若不能全部消除则图中仍存在负环。具体而言：遍历图的边集，判断每条边是否在当前势能下为负边；若找到任意一条负边，就基于这条边执行一轮势能调整，若调整后该边仍是负边则调整失败，图中仍存在负环，本次判定结束；否则调整成功，消除一条负边，再次遍历边集找是否存在其他负边，重复这个过程直到消除所有负边或者某条负边无法被消除。

以上判定过程中执行势能调整总次数至多为 $2q$，因为：

- 调整失败至多 $q$ 次：调整失败就结束本次判定；
- 调整成功至多 $q$ 次：调整成功会消除一条负边。

每次势能调整 Dijkstra 算法时间复杂度为 $O(n\log n+m)$，再加上初始势能需要执行一次 SPFA 算法，这个算法的总时间复杂度为 $O(nm + q(n\log n + m))$。

这个问题是一般图动态判负环最基础的模型，更多动态操作例如加边、删边等都可以等效为边权的修改，使用本问题的方法解决。

**例题 2.4.** 给定有向图，包含 $n$ 个点和 $m$ 条有向边 $(u_i,v_i,w_i)$，保证图中存在环。执行 $q$ 次修改操作：每次修改包含两个正整数 $x,y$，表示将第 $x$ 条边的边权 $w_x$ 减少 $y$，并保证任意时刻都不存在负环。

定义环的比率为环上所有边的边权之和与环上的边数的比值向下取整后，即 $\left\lfloor \frac{\sum w}{len} \right\rfloor$。在每次修改后在线求解图中最小比率环的比率。

数据范围：$1 \le n,m,q,|w|,y \le 5\times 10^3$。

先将最小比率环转化为判负环问题：设最小比率环比率为整数 $x$，当且仅当将每条边

的权值减 $x$ 后图中没有出现负环，且将每条边的边权减 $(x+1)$ 后图中出现负环。

设 $W_0 = \max(w_i)$，则答案满足 $0 \le ans \le W_0$。由于每次修改只会使边权减少，答案一定单调不增。所有修改开始前先计算初始状态的答案，将所有边权减 $(W_0+1)$，此时一定有负环；接下来令边权逐渐增加，每次令所有边的权值都加 1，直到图中无负环，即得到初始答案。每次修改一条边后也采用类似的方法，基于上次答案逐渐增加边权，每次都直到图中无负环为止。

基于例题 2.3 中动态判负环方案，整个过程中所有边权加 1 的修改至多有 $W_0+1$ 次，这种修改只可能使势能边权为负数的边数减少，不可能产生新的负边。$q$ 次将边权减小的操作每次可能会产生一条负边，加上初始所有边的权值减去 $W_0$ 所产生的负边，负边总数不超过 $m+q$。因此总时间复杂度为 $O((m+q)(n\log n+m)+W_0\cdot m)$。该方案虽未完全满足数据范围要求，但已达到较优复杂度，进一步优化可利用边权修改量 $y$ 较小的特性，用桶替代 Dijkstra 中的堆，参考下文的例题 2.6。

## 2.3 修改量较少的最短路优化

当边权修改量较小时，可在 Johnson 势能调整过程中采用桶结构替代 Dijkstra 算法中的堆，显著提升效率。

**例题 2.5** (CF843D). 给定带权有向图，包含 $n$ 个点和 $m$ 条边，边权非负。有 $q$ 次操作：

- 1 $v$：查询从起点 1 到终点 $v$ 的最短路径长度。
- 2 $c$ $l_1$ $l_2$ … $l_c$：编号为 $l_1, l_2 \ldots, l_c$ 的边的权值各加 1。

数据范围：$n, m \le 10^5$，$q \le 2000$，$\sum c \le 10^6$。

如果每次查询都执行 Dijkstra 算法，时间复杂度为 $O(q(n\log n+m))$，无法达到本题要求。其原因是完全没有利用到边权修改量较小的特性。

将每个节点的势能设置为从起点 1 出发到该点的最短路长度，在当前势能下从起点 1 出发到每个点的最短路长度都为 0。执行修改操作使 $c$ 条边的权值加 1 后，当前势能下每个点最短路长度不超过 $c$。此时调整势能可以在 Dijkstra 过程中用 $O(c)$ 个桶来代替堆：更新节点 $i$ 距离后若距离为 $d_i$ 则将点 $i$ 放入第 $d_i$ 个桶中；每次从首个非空桶中取节点更新其他点的距离。除了堆换成桶外其他步骤保持不变，Dijkstra 过程结束后将每个点的势能更新为当前最短路的长度。

改进后的 Dijkstra 算法单次势能调整的时间复杂度为 $O(n+m+c)$，所以算法的总时间复杂度为 $O(q(n+m)+\sum c)$。

**例题 2.6.** 给定有向图，包含 $n$ 个点和 $m$ 条边 $(u_i, v_i, w_i)$，保证初始图中无负环。执行 $q$ 次修改，每次修改参数 $x, y$，表示将第 $x$ 条边权值 $w_x$ 加上 $y$（$y$ 可能为负），修改后先判定图是否存在负环，若无负环则还要计算 1 号点到每个点的最短路长度，输出这些最短路长度的哈希值。每次修改后的判负环和最短路都需要在线解决。

数据范围：$1 \le n, m, q, |y| \le 5 \times 10^3$，$|w_i| \le 10^9$。

若直接沿用例题 2.3 动态判断负环的方法，并结合势能在无负环时从起点 1 使用 Dijkstra 求出单源最短路，总时间复杂度为 $O(nm + q(n \log n + m))$，在本题数据规模下无法通过。本题也无法直接沿用例题 2.5 中的做法，因为本题修改后的势能边权可能为负数。

考虑从势能的定义上入手，将节点势能直接定义为起点 1 到该点的最短路长度。这样定义的优势在于询问时答案即为势能，无需重新求最短路。而劣势在于势能维护的难度增大，需要重新推导势能维护的过程。

计算初始势能的方法不变，即以节点 1 为起点执行一次 SPFA 算法求出每个点的最短路。接下来面临修改，先考虑图中无负权边情况下的势能调整方法。

当边权增加，参考例题 2.5 的原理，起点 1 到每个点的最短路增量不会超过 $y$，即在当前势能下最短路长度不超过 $y$，用桶优化的 Dijkstra 算法更新势能，每次时间复杂度为 $O(n + m + y)$。

当边权减少，即 $y < 0$。设本次操作修改的边是 $(u, v)$，修改前的势能边权为 $w$。若修改后势能边权非负，即 $w + y \ge 0$，则所有势能不用调整，最短路答案不变。否则 $w + y < 0$ 需要调整势能。若修改边权后图中无负环，则起点 1 到节点 $u$ 的最短路长度 $d_u$ 不会变化，这意味着 $d_v$ 必须减小使 $(u, v)$ 的势能边权从负数 $w + y$ 增加到 0。设 $T = -(w + y)$，将这个过程拆分为先将 $w$ 减到 0，再进行 $T$ 轮，每轮将边权减少 1 并调整势能使边权重新变成 0。每一轮，所有从 $v$ 出发只经过势能边权为 0 的边能到达的点，势能都必须减 1，将这些点记为点集 $S$。点集 $S$ 的势能变化会导致从 $V \setminus S$ 到 $S$ 的所有边的势能边权增加 1，从 $S$ 向 $V \setminus S$ 的所有边的势能边权减少 1。如果 $T$ 轮都顺利进行，则调整结束；如果中途发现 $u \in S$，即点 $v$ 到点 $u$ 的最短路长度已经为 0，而 $(u, v)$ 的势能边权仍为负，说明本次修改产生了负环。

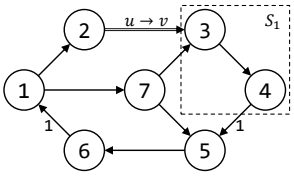



图 2: 原图与集合 $S_0$

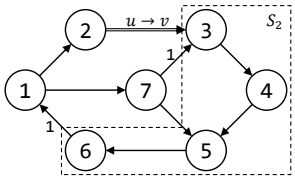



图 3: 一轮后与集合 $S_1$

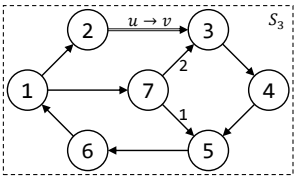



图 4: 两轮后与集合 $S_2$

例如图中所示的调整过程，其中标注的边权为势能边权，未标注则势能边权为 0。调整前的原图如图 2 所示，当前基于边 $2 \to 3$ 调整势能。第一轮将 $S_0$ 的每个点势能减 1，则 $7 \to 3$ 边权加 1，$4 \to 5$ 边权减 1。因此第二轮调整势能的点集 $S_1$ 新增节点 5 和 6，如图 3 所示。第二轮调整后，$S_2$ 新增三个节点 $1, 2, 7$，如图 4。此时 $u = 2$ 已加入点集，因此再执行一轮调整将会出现负环。

按以上方法调整后的势能，在图中不存在负环的情况下，仍然符合本题对势能的特殊要求，即势能等于从起点 1 出发到每个点的最短路，换言之从起点 1 出发只经过势能边权

为 0 的边可以到达图中所有点。要证明这一点，可用数学归纳法，一轮调整前的势能符合要求，需要说明调整后势能等于再执行边权 −1 修改后的最短路长度。考虑这轮的集合 $S$，对于 $S$ 集合外的节点 $x$，从 $S$ 内出发只走势能边权为 0 的边无法到达 $x$，但从 1 出发只走势能边权为 0 的边可以到达 $x$，所以从 1 到 $x$ 的路径与 $S$ 交集为空，调整后 1 仍然可以沿这条路径到 $x$。而对于 $S$ 集合内的点，调整后从 1 出发可以先到点 $u$ 再经过 $(u,v)$ 边后沿 $S$ 内的点到达，这条路径所有势能边权都等于 0。因此，调整后的势能仍然满足本题对势能的特殊要求。

在以上势能调整的过程中，每一轮的点集 $S$ 一定不断变大，节点被加入到 $S$ 以后不会退出。设节点 $x$ 是第 $t_x$ 轮被加入到 $S$ 中，若 $T$ 轮后仍未加入 $S$ 则规定 $t_x = T$，那么点 $x$ 的势能减少量 $\Delta d_x = T - t_x$。只要算出每个节点的 $t$ 就能一次性完成 $T$ 轮调整，而计算 $t$ 本质上也是最短路问题。具体而言，起点 $v$ 设置 $t_v = 0$。对于一条边 $(x,y,z)$，其中 $z$ 是势能边权，已知点 $x$ 在第 $t_x$ 轮被加入加入 $S$，若 $y$ 一直不被加入 $S$，那么这条边的势能边权从第 $t_x$ 轮开始每轮减 1，在第 $t_x + z$ 轮时已经被减为 0 从而点 $y$ 被加入到 $S$。不难看出 $t$ 即为势能调整开始前 $v$ 到每个点的最短路长度和 $T$ 取较小值，即 $t_x = \min(T, \mathrm{dis}(v,x))$。而 $T = -(w+y) \le -y$，所以使用桶优化的 Dijkstra 算法调整势能的时间复杂度为 $O(n+m+|y|)$。

如果 $T$ 轮调整的中间某一轮，点 $u$ 被加入到 $S$，调整就应立即停止。对应到 Dijkstra 计算 $t_x$ 的过程中，若算出 $t_u < T$ 则说明有负环，一旦从桶中取出节点 $u$ 且 $t_u < T$，Dijkstra 就应立刻停止，并将 $T$ 改为 $t_u$ 后再调整 $t_x \le T$ 的所有点 $x$ 的势能。这样，无论是否有负环，当前负权边的势能边权增加量都恰为 $T$，同时 Dijkstra 过程中桶的使用量也是 $T$。这样才能确保所有调整中桶的枚举量之和不超过 $\sum \max(-y, 0)$。

现在还需解决负边积累的问题，将本题的势能调整方法与例题 2.3 的负边消除方法相结合。通过遍历边集找到一条当前势能下的负权边 $(u,v,w)$，尝试调整势能将它消除，同样是以 $t_v = 0$ 为起点，限制 $T = -w$ 执行 Dijkstra，再根据 $t$ 数组调整势能。Dijkstra 中遇到其他负边就将其势能边权视为 0，这样做可以确保调整后其他负权边不会变得更小。

按上述方法消除负边，直到发现全部消除则图上不存在负环，或者发现一条无法消除的负边则停止。但是，若通过这个过程消除了所有负边，结束时每个点的势能不一定为 1 到该点的最短路长度，其原因在于从最近一次图中无负环到现在，可能进行了很多次边权增大的操作，这些操作没有调整势能。设这期间，修改增大边权的总增加量为 $Y = \sum \max(y, 0)$，则当前势能下起点 1 到每个点的最短路一定不会超过 $Y$，只需再执行一次桶优化的 Dijkstra 即可将势能调整正确。

在这个算法中，所有 Dijkstra 算法的桶的用量为 $\sum |y|$，而 Dijkstra 次数不超过 $O(q)$，因此时间复杂度为 $O(nm + qn + qm + \sum |y|)$。

上文例题 2.4 最小比率环问题末尾遗留的如何进一步优化的问题，在这里已经解决。该问题只需要判负环无需计算最短路，所以算法流程可以更加简略，边权增加后不用处理，只需要在图上存在负环时尝试调整即可。

## 2.4 小结

本章节用多个例题，分别讨论了各种动态图场景下的 Johnson 势能调整方法。费用流问题中每条边都存在权值为相反数的反向边，原始对偶算法和 ZKW 算法都根据这个特殊性以非常简单的操作就实现了势能调整。而一般的动态图上判负环和动态最短路问题，则需要用 Dijkstra 算法调整势能，且在边权变化量较小时可以将 Dijkstra 中的堆换成桶以提升效率。以上各个场景下，使用 Johnson 势能调整的根本目标都是消除负权边产生的影响，将原本需要 SPFA 算法解决的问题转化为可以用 Dijkstra 解决的问题，从而提高算法效率。

# 3 单次判负环

基于动态图相关算法的铺垫，下文回归最原始的单源最短路问题的探讨：给定一个有向图，求一个起点到其余各点的最短路径长度，或判定图中存在负环。下文将介绍时间复杂度优于传统 SPFA 算法的算法。

## 3.1 边权不小于 −1 的算法

**例题 3.1.** 构造一个长度为 $n$ 的正整数序列，满足 $m$ 个约束条件，其中第 $i$ 个约束条件为，区间 $[l_i, r_i]$ 内奇数下标的元素和与偶数下标的元素和相同。

先判断是否有解，若有解，需要求出一组解使序列的最大值最小。

数据范围：$n, m \le 2 \times 10^5$ 。

先将原约束条件进行转化：将偶数下标的元素的正负取反，再求前缀和，则原约束条件等价于 $r_i$ 为止与 $l_i - 1$ 位置的前缀和相等。

对转化后的约束条件进行差分约束建图：从 0 到 $n$ 每个位置对应图中一个节点，节点的权值为这个位置前缀和。对于每个约束条件 $i$，将节点 $r_i$ 与节点 $l_i - 1$ 合并即可满足该约束。解决题目要求的“最大值最小”的问题一般用二分答案，设当前二分的答案是 $ans$，即原序列中每个元素的值域限制为区间 $[1, ans]$ 。在差分约束构图中，每个元素对应图中两个节点的权值之差，在两点间构建正反两条边，正向边权 $-1$，反向边权 $ans$ 。现在题目中所有约束条件都已经等价的转化到了图上，在当前二分答案限制下原问题有解等价于图中无负环。图中包含的点数和边数均为 $O(n)$，单次 SPFA 时间复杂度为 $O(n^2)$ 。注意到图中负边权值均为 $-1$ 且数量不超过 $n$，如果负边本身构成环则无解，否则令环上正边权值 $ans = n$ 一定可以消除负环，所以二分答案的上界也是 $n$ 。所以总时间复杂度为 $O(n^2 \log n)$ 。

观察图的形态。当 $m = 0$ 时，仅考虑边权为 $-1$ 的边，图的形态如下图 5 所示。站在出题人角度考虑，合理利用节点合并操作，可以构出各式各样的图，卡满 SPFA 算法及其各种变体，例如图 6 。

注意到图中仅存在权值为 $-1$ 的负权边，这使得 Johnson 算法的势能调整思想具备良好的可行性。因为借用上文例题 2.6 中介绍的将负数边权视为 0 的方法在势能调整中不会使负

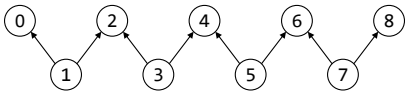



图 5: 图中为 $n = 8, m = 0$ 时建图的形态。每条边正向权值为 $-1$ 反向权值为 $ans$ 。

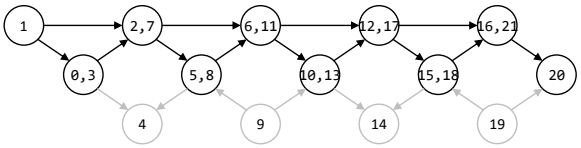



图 6: 卡满 SPFA 算法的经典方法之一是连续多个“三角形”，图中展示了 $n = 21$ 时构造 5 层“三角形”的例子，该结构可以向右不断延伸。

权边权值进一步减少，这能使势能调整过程更简单。接下来要介绍的是 Goldberg 提出的一种高效算法 [4]。

初始时所有顶点的势能均设为 0 。在图中仅保留当前势能边权不超过 0 的边，当势能调整导致边权变化时自动修改边集。下文中“可达性”的描述也表示只使用当前不超过 0 的边时一个点能否到达另一个点。基本的势能调整策略是：从图中每次取一条当前势能边权为负的边 $(u, v, -1)$ 尝试调整为非负，若 $v$ 可达 $u$ 则说明图中存在负环；否则将 $v$ 可达的所有点的势能减 1 ，此时该边的势能边权变为 0 ，且调整过程中不会使其他边的势能边权变负。

以上的基本势能调整策略每次只能消除一条负权边，考虑优化，使其能够批量的消除负权边。

第一步，先对于每条负权边 $(u, v, -1)$ 判断 $v$ 是否可达 $u$ ：在当前仅保留势能边权不超过 0 的边构成的图中求强连通分量，若存在某条负权边的两个端点同属一个强连通分量，则说明图中有负环，算法终止；否则图中每条负权边的 $v$ 都不可达 $u$ 。

若第一步未检测到负环，则执行第二步调整势能。考虑按第一步的强连通分量缩点后得到的有向无环图，在图上选择若干两两不可达的负权边，将这些边的可达点集的并集中所有节点的势能都减 1 。如此调整势能时间复杂度为 $O(n)$ ，调整完成后，所有被选择的负权边都变为 0 ，且不会产生新的负权边。

在负权边可达性形成的偏序集中，上述势能调整方法可以一次性消除一条反链的负权边。设 $m_0$ 为当前图上势能边权为负的边数，根据 Dilworth 定理，最长反链的长度和最长链的长度至少有一个不小于 $\sqrt{m_0}$ 。如果能再设计一个势能调整算法，支持消除一组构成链的负权边，那么将两种势能调整算法相结合就能保证每次至少消除 $\sqrt{m_0}$ 条负边，可以通过计算证明 $O(\sqrt{n})$ 轮即可全部消除。

假设选取了 $k$ 条负权边，其中第 $i$ 条边 $e_i = (u_i, v_i, -1)$ ，且 $e_i$ 可达 $e_{i-1}$ 。按照逐一消除的方法，需要按 $i$ 从小到大处理 $e_i$ ，当枚举到 $e_i$ 时，设 $S_i$ 为当前图中 $v_i$ 的可达点集，若 $u_i \in S_i$ 则有负环，否则将 $S_i$ 集合中的点势能减 1 。

现在要批量消除负边，因为 $e_i$ 可达 $e_{i-1}$，所以 $S_{i-1} \subset S_i$ 。注意不能直接按第一步的强连通分量来计算这些集合，因为一些权值严格大于 0 的边在前几轮消除时可能变成 0 然后被加入到图中。正确做法是：对于每个点 $x$ 算出 $t_x = \max\{i \mid x \notin S_i\}$ ，则 $x$ 的势能减少量为 $k - t_x$ 。设置 $k$ 个起点 $t_{v_i} = i - 1$ ，使用桶优化的 Dijkstra 算出所有节点的 $t$ 值。若存在 $t_{u_i} < i$ 则判定图中存在负环。这种势能调整方法的时间复杂度同为 $O(n)$。

结合两种势能调整方法，势能调整的迭代论述已经可以控制在 $O(\sqrt{m_0})$ 轮内，但还遗留了一个问题：每轮如何快速选出一组可以批量消除的负权边。找一组构成链的负权边可以在强连通分量缩点后的有向无环图上 DP，给每条负边设定 DP 状态为“从这条边出发最多可以经过多少条负边”，所有负边的 DP 状态的最大值即为最长链的长度。若最长链长度 $\geq \sqrt{m_0}$，则根据 DP 转移倒推可以得到最长链上的所有负边，然后执行第二种势能调整方法；否则，DP 值的众数出现次数必然 $\geq \sqrt{m_0}$，且对应的这些负权边一定互相不可达，找出这些边然后执行第一种调整方法。

无论是求解强连通分量、DP 计算最长链，还是两种调整方法，时间复杂度都为 $O(n)$（本题边数 $m = O(n)$）。至此，该算法能够在 $O(n\sqrt{n})$ 时间内完成负环判定，即为 Goldberg 算法，相比传统 SPFA 算法取得了显著优化。

本题外层还需要二分答案，总时间复杂度是 $O(n\sqrt{n}\log n)$，仍无法满足题目要求。由于负环判定部分已难以进一步优化，因此需尝试去除二分操作，采用动态判定的方式求解。

考虑到若将正权边的统一权值 $ans$ 降低，可能会产生大量新的负边，因此只能允许 $ans$ 增大。从 $ans = 1$ 开始不断增大 $ans$，每次增大后需要判断是否仍存在负环，直到图中不存在负环时算法结束。每次判断时若结果为图中仍有负环，则会增加 $O(n)$ 额外时间复杂度。

为减少此处增加的时间复杂度，可以采用类似 BSGS 的策略验证 $ans$。具体操作时：先从 $ans = 1$ 开始以步长为 $step = O(\sqrt{n})$ 增大 $ans$，当图中负环全部被消除时设为 $ans_0$；然后全部清空，再从 $ans = ans_0 - step + 1$ 开始以步长 1 增大 $ans$，直到算出最终答案。这样可以保证枚举 $ans$ 带来的额外时间复杂度也是 $O(n\sqrt{n})$，最终本题总时间复杂度为 $O(n\sqrt{n})$。

本题中 Goldberg 算法可以在负权只有 $-1$ 时高效的通过调整势能将图中的所有边变得非负，从而判负环或求最短路。本题边数 $m = O(n)$，而对于 $n$ 个点 $m$ 条边的场景，Goldberg 算法时间复杂度为 $O((n+m)\sqrt{m_0})$，其中根号下的 $m_0$ 表示图中负权边的数量。另外，若负权边的权值为任意负整数，则 Goldberg 算法还需要增加一步额外的问题转换流程。

## 3.2　一般图判负环

对于图 $G_0 = (V, E_0)$ 中的任意一条边 $(u, v, w_0)$，设 $w_1 = \left\lceil \frac{w_0}{2} \right\rceil$，并向新图 $G_1 = (V, E_1)$ 的边集 $E_1$ 中添加一条边 $(u, v, w_1)$。新图 $G_1$ 的每条负权边绝对值不超过原图 $G_0$ 对应边的一半，且原图中非负权的边在新图中也非负。假如已经算出一组节点势能 $d$，在该势能下图 $G_1$ 中所有边都非负。将每个点的势能都乘 2 后，图 $G_0$ 中的每条边 $(u, v, w_0)$ 的边权：

$$\begin{aligned} w_0' &= 2d_u + w_0 - 2d_v \\ &\geq 2d_u + (2w_1 - 1) - 2d_v \\ &\geq -1 \end{aligned}$$

即图 $G_0$ 中负权边的权值均为 $-1$，接下来再使用上一节中的算法消除这些负边，时间复杂度为 $O((n+m)\sqrt{m})$。关于负环的判定：由于 $G_0$ 可看作是将 $G_1$ 的各边边权翻倍后，对部分边的权值减 1 得到的图，因此若图 $G_1$ 中存在负环，则该环在图 $G_0$ 中仍为负环。

若 $G_1$ 中仍存在小于 $-1$ 的边，可以按上述方法递归处理。每层递归都会将负数边权的值域折半，直到图中不存在负权边；在递归返回阶段调整势能消除当前途中所有负边，若某一层发现负环，则说明原图存在负环。这样的方法可高效解决一般有向图的负环判定问题，整体时间复杂度为 $O((n+m)\sqrt{m}\log W)$，其中 $W$ 表示原图中负权边绝对值的最大值。若还需计算单源最短路，可以在递归判负环的过程结束后基于最终的势能用 Dijkstra 算法计算。

**例题 3.2.** 给一张有向图，包含 $n$ 个点和 $m$ 条边 $(u_i, v_i, w_i)$，保证图中存在环。定义环的比率为该环的边权和与边数的比值向下取整，即 $\left\lfloor \frac{\sum w}{len} \right\rfloor$。要求找出图中任意一个最小比率环。

数据范围：$1 \le n, m \le 5 \times 10^4$，$|w_i| \le 10^9$。

该问题为最基本的最小比率环问题，常规思路是采用二分答案结合负环判定算法。设二分过程中当前需要验证的答案 $ans$，需要将原图每条边的边权减 $ans$ 后判断是否存在负环。若此时存在负环，则答案应小于 $ans$；若此时无负环，则答案不小于 $ans$。若采用 SPFA 算法进行负环判定，整体时间复杂度为 $O(nm\log W)$，即使替换为一般图的 Goldberg 算法，时间复杂度为 $O((n+m)\sqrt{m}\log^2 W)$。

时间复杂度中的两个“log”因子，一个源于二分答案，另一个来自将负数边权折半过程的递归层数。类比 OI 中常见的“二分答案嵌套倍增验证”可优化为“倍增过程直接求解答案”的逻辑，此处可将这两层步骤合并以降低时间复杂度。

从原图 $G_0$ 开始将负数边权折半，递归 $k$ 层后的图 $G_k$ 上，边权数值中每 1 的权值等效于原图 $2^k$ 的权值。设递归层数为 $p = O(\log W)$ 时，原图 $G_0$ 的每条权值 $\le 0$ 的边在 $G_p$ 中都变成 0，$G_0$ 的每条权值 $> 0$ 的边在 $G_p$ 中都变成 1，即图 $G_p$ 中仅包含权值为 0 和 1 的边。

若将 $G_p$ 中所有边的权值减 2 后，图中所有边均为负权，图中必然存在负环。这等效于 $G_0$ 中所有边权值减 $2^{p+1}$ 后图中必然存在负环，说明原问题答案严格大于 $-2^{p+1}$。又因为原图负数边权不小于 $-(2^p-1)$，所以若将原图所有边的权值增加 $(2^p-1)$ 后图中一定没有负环。因此，原问题的答案被限定在区间 $(-2^{p+1}, 2^p)$ 内。

对于 $k = 0, 1, \cdots, p$，设 $g_k$ 表示当前 $G_k$ 的边权相对于原图 $G_0$ 的总修改量，当前还没有做任何修改所以 $g_k = 0$。当前答案范围是区间 $ans \in (-2^{p+1}, 2^p)$，考虑将这个长度为 $3 \times 2^p$ 的答案区间缩小到长度为 $2 \times 2^p$。可以先将图 $G_p$ 中所有边权都减 1，这会使 $g_p$ 减去 $2^p$，然后判定 $G_p$ 中是否存在负环。若出现负环，则撤销 $G_p$ 的边权减 1 操作，不保留本次判定过程对 $G_p$ 计算的势能，并将 $g_p$ 重置为 0。完成这一步后，$g_p$ 与正确答案之差的绝对值一定小于 $2^p$，即答案范围 $|ans - g_p| < 2^p$。

接下来从 $G_p$ 开始逐层反向迭代至原图 $G_0$，同时逐渐缩小答案区间。假设当前已经处理完的图 $G_{k+1}$，修改量为 $g_{k+1}$，满足 $|ans - g_{k+1}| < 2^{k+1}$。现在要处理图 $G_k$，在本轮迭代中将这个长度为 $4 \times 2^k$ 的答案区间缩小到长度为 $2 \times 2^k$，这需要两次负环判定。

先直接将 $G_{k+1}$ 的势能乘 2 作为 $G_k$ 的势能，令 $g_k = g_{k+1}$。此时图 $G_k$ 中负数边权都等于 $-1$，执行一遍负环判定算法。若图中有负环，则 $ans > g_{k+1}$，即答案区间缩减为 $ans \in (g_{k+1}, g_{k+1} + 2^{k+1})$，令 $g_k = g_{k+1} + 2^k$，将图 $G_k$ 将所有边权都加 1，节点势能不变，本轮迭代结束。若图中无负环，则答案区间上界变为 $ans < g_{k+1} + 2^k$，此时答案区间长度缩小

到 $3\times 2^k$，还需要再缩小一次。尝试将 $G_k$ 所有边权都减 1 后再执行一遍负环判定，如果无负环，则 $ans < g_{k+1}$，令 $g_k = g_{k+1} - 2^k$，本轮迭代结束；否则撤销 $G_k$ 的边权减 1 操作，不保留本次判定过程对 $G_k$ 计算的势能，令 $g_k = g_{k+1}$，本轮迭代结束。以上三种迭代结束的情况都满足图 $G_k$ 无负环且 $|ans - g_k| < 2^k$，达成了缩小答案区间的目的。按照该过程 $k$ 从大到小迭代 $p$ 轮后，最终 $|ans - g_0| < 2^0$，即 $ans = g_0$ 就是本题的正确答案。

上述过程仅求解出最小比率环的比率 $ans$，题目还需要算法能够定位图中某一个最小比率环：将每条边的权值减去 $(ans + 1)$ 后，图中的任意一个负环即为所求的最小比率环。

负环的定位过程较为简洁，在 Goldberg 中算法的执行流程中，有两处步骤可判定负环并定位具体环。其一，在仅保留非正权边的子图中求强连通分量，若某条负权边的两个端点属于同一个强连通分量，则此处形成负环。定位负环可以从这条负边 $(u, v)$ 的节点 $v$ 出发在强连通分量中寻找一条到节点 $u$ 的路径。其二，在处理链状分布的负权边时，若 Dijkstra 过程中发现 $t_{u_i} < i$ 则此处形成负环，定位负环只需在 Dijkstra 算法时记录路径转移关系，即可从 $u_i$ 回溯得到负环。

以上求最小比率环的算法的总时间复杂度为 $O(n\sqrt{n}\log W)$，此处 $W$ 为原图中边权绝对值的最大值。

## 3.3 小结

本章节的核心是 Goldberg 算法，该算法可以判定一张图中是否存在负环，同时还能求出一组节点势能。Goldberg 算法有两个版本，本章节先在例题 3.1 中展示了在负数边权都等于 $-1$ 的特殊图场景下的 Goldberg 算法，其时间复杂度为 $O((n + m)\sqrt{m})$。对于一般图，Goldberg 算法时间复杂度为 $O((n + m)\sqrt{m}\log W)$。笔者在学习和研究过程中发现，在最小比率环问题中可以改进 Goldberg 算法得到更低的时间复杂度，于是在本章节最后部分的例题 3.2 中详细展示。

除了本文中的算法之外，学术界还存在一些理论算法，在本文重点研究的判负环等问题上可以做到更低的理论时间复杂度。这些算法大部分都要利用 Johnson 的势能调整思想，但一些算法适用场景有局限性，一些算法程序实现难度较大不适用于 OI 中，本文不再展开介绍。

# 4 总结

本文较为系统探讨了 Johnson 思想在图论问题中的应用，包括全源最短路、费用流算法优化、动态图、最短路和负环判定等多个场景。针对动态图问题，提出了一类通用性较强的势能调整的解决方法。对于负环判定问题，介绍了信息学竞赛中容易实现且可以高效运行的 Goldberg 的算法。

在查阅相关文献与竞赛题目时，笔者发现信息学竞赛圈内缺乏 Johnson 思想在上述领

域的系统研究，应用该思想的具体题目案例也较少。本文所述各种情形下的势能调整思路（如势能总变化量较小时、边权修改量较小时等）仍有进一步拓展的空间。本文能够起到抛砖引玉的作用，帮助读者深入理解 Johnson 思想的核心价值，推动更多相关题目与高效算法在信息学竞赛中的出现。

## 致谢

感谢中国计算机学会提供学习和交流的平台，感谢国家集训队教练的指导。
感谢家人对我的陪伴与支持。
感谢董又铭老师、杜瑜皓老师在论文上为我提供的帮助。
感谢重庆南开（融侨）中学校对我的栽培。
感谢其他给予我帮助的老师与同学。

## 参考文献

[1] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. Journal of the ACM, 24(1):1–13, 1977.

[2] 最短路. https://oi-wiki.org/graph/shortest-path/, 2026.

[3] 费用流. https://oi-wiki.org/graph/flow/min-cost/, 2026.

[4] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. SIAM Journal on Computing, 24(3):494–504, 1995.

# 一类费用流问题的优化

深圳市耀华实验学校 胡晓虎

## 摘 要

网络流问题是计算机科学领域的经典问题，其中费用流问题因其广泛的应用和较高的求解难度而在信息学竞赛中备受关注。本文旨在整理并引入一系列针对费用流算法的优化方法。

本文对一般的费用流算法进行了分析，包括消圈算法、SSP（Successive Shortest Path）算法、Capacity Scaling 算法以及 Network Simplex 算法，探讨了它们在随机图与特殊构造图中的时间复杂度表现及适用场景。

本文还列举了一些性质良好的费用流图并且展示了其相关优化，并深入剖析了其与反悔贪心策略的联系。通过对若干经典例题的建模分析，展示了如何在此类问题中将通用费用流算法的高复杂度降低。

最后，本文总结了不同算法的实际应用策略，为解决此类问题提供了系统的理论参考与实践指导。

## 目 录

# 1 引言

网络流问题是计算机领域的一个经典问题。在信息学竞赛中，网络流常常被用于解决种类繁多的最优化问题。其中，费用流是网络流问题的一个经典变种，在适用范围更广的同时拥有较低的效率。这些题目经常出现在各大信息学竞赛的比赛中。

然而，大部分这类题目均以“建立正确的费用流图”为考察目标，而没有对费用流算法本身进行要求；有些标程使用的费用流算法是伪多项式的，只能处理一些较小的数据。也

有部分题目考察了对费用流问题的优化，但题目较为分散，且大多可以使用其他算法通过，这对学习这类算法不利。

本文整理并引入了一系列对费用流算法的优化方法。第三节介绍了在大部分费用流图中均可使用的算法，第四节例举了一些常见的性质良好的费用流图并展示了其相关优化。本文不会重点讲解一些题目的建模转化过程。

## 2 约定

网络（network）是一个特殊的有向图 $G = (V, E)$。一条边 $e \in E$ 具有四个属性，被记为 $(u, v, c, w)$，表示起点，终点，容量和费用。这种定义不否认重边（$u, v$ 相等，但 $w$ 不等）的存在，但是重边对下文中的全部证明均不影响，下文不显式地考虑重边，在实际出现重边的地方算法会隐式地区分重边。

可以使用几个矩阵来描述网络：对 $(x, y, z, p) \in E$ 使用 $c(x, y) = z$ 来表示流量，$w(x, y) = p$ 表示费用。对于其他的 $(x, y)$，可以认为 $c(x, y) = 0$。

对于一个网络，再对其进行处理前，一般会添加反向边，这有多种本质相同的实现方式，这里只介绍其中一种。具体来说，对于 $c(x, y) \neq 0$，维持 $c(y, x) = f(x, y)$，$w(y, x) = -w(x, y)$，且试图修改 $f(y, x)$ 时改为反向修改 $f(x, y)$。容易发现，一单位流量流经反向边相当于撤回了原边的一单位流量，而在分析时反向边的 $c$ 会更有利于简化情况。

对于一个网络，若不指定两个特殊点源点和汇点，默认为无源汇的图。在这样的图中，需要满足条件 $\sum_i f(x, i) = \sum_i f(i, x)$ 和 $f(x, y) \leq c(x, y)$。无源汇的最小费用可行流即求在满足这个条件下 $\sum f(x, y)w(x, y)$ 的最小值。

对于一个网络，若指定两个点 $s, t$ 分别为源点和汇点，那么 $s$ 和 $t$ 不需要作为 $x$ 满足 $\sum_i f(x, i) = \sum_i f(i, x)$。求满足 $\sum_i f(s, i)$ 最大的情况下 $\sum f(x, y)w(x, y)$ 的最小值被称为有源汇最小费用最大流。

对于一个无源汇的网络，进行一次推流需要选择一个环 $p_1, p_2, \ldots, p_k, p_1$ 和一个正整数 $v$，使得 $c(p_i, p_{i \bmod k+1}) - f(p_i, p_{i \bmod k+1}) \geq v$，将所有 $f(p_i, p_{i \bmod k+1})$ 增加 $v$。对于一个有源汇的网络，还可以选择 $p_1, p_2, \ldots, p_k$ 和正整数 $v$ 使得 $c(p_i, p_{i+1}) \geq v$ 且 $(p_1 - s)(p_1 - t) = (p_k - s)(p_k - t) = 0$，将所有 $f(p_i, p_{i+1})$ 增加 1。若无额外说明，$v$ 一般取路径上 $c$ 的最大值。容易发现任意推流后答案不变，因为所有操作都是可逆的。

本文中，未额外说明的情况下，使用 $n = |V|$，$m = |E|$，所有数均为整数，容量和流量非负。未额外说明时，所有操作的费用均为 $10^9$ 以内的非负整数。

本文中，随机图或随机数据指在所有符合限制条件的图或数据中均匀随机生成的图或数据。本文默认图连通，对于不联通的图可以对每个连通块分别处理。

本文中，任意图指全部符合限制条件的图，一般在最坏情况出现。性质良好的图指可以进一步分析，可能分析出更优秀复杂度的图。

本文算法中的 $\infty$ 表示一个极大的数。

部分题目题面及其数据范围被略微改写，但在无额外说明的情况下不影响做法。

本文默认带负权的单源最短路使用 SPFA 算法，在任意给出的图上有 $O(nm)$ 的时间复杂度，在随机图上有 $O(m)$ 的时间复杂度。

本文默认非负权值的单源最短路使用斐波那契堆优化的 Dijkstra 算法，在所有图上具有 $O(n\log n+m)$ 的时间复杂度。

# 3 一般的费用流算法

## 3.1 题目

**例题 1：【模板】最小费用最大流**[1] 给出一个 $n$ 个点 $m$ 条边的有源汇的网络，求出它的最小费用最大流。

$n\le 5\times 10^3, m\le 5\times 10^4$，$0\le c,w\le 10^3$。保证图随机生成。

**例题 2：最小费用最大流**[2] 给出一个 $n$ 个点 $m$ 条边的有源汇的网络，求出它的最小费用最大流。

$n,m\le 500$，$0\le c\le 10^9, |w|\le 2\times 10^6$。

这两个题目十分相似，都是最小费用最大流的模板问题，其中的差异在于数据的保证。例题 2 不保证数据随机，对算法的最坏情况的考验更大，而例题 1 的数据范围更大，要求在随机图上更快的算法。

## 3.2 问题的转化

例题 1 和例题 2 均为有源汇最小费用最大流问题，而一些算法基于无源汇最小费用可行流；有些能解决有源汇最小费用最大流的算法需要应用于无源汇的图；还有些题目需要求最大费用或有源汇的最小费用可行流。这时就需要对这些问题之间的转化。

无源汇的最小费用可行流改成有源汇的相当简单：加两个不与任何点相连的点作为源点和汇点即可；有源汇改成无源汇只需要加上边 $(t,s,\infty,-\infty)$ 即可用于求最小费用最大流，将边权改为 0 即求最小费用可行流，将边权改为 $-\epsilon$ 即求最小费用情况下的最大流。

一些算法无法处理负边权，一些题目需要要求一些边的容量有下界。事实上，这两个问题是类似的，对于无法处理负边权的边，可以令其下界为容量，然后主动加入反向的边即可，于是只需要求上下界问题。这是一个经典建模问题，可以使用一次有源汇最小费用最大流解决。

[1] 来源: P3381 【模板】最小费用最大流 https://www.luogu.com.cn/problem/P3381

[2] 来源: UOJ487 最小费用最大流 https://uoj.ac/problem/487

## 3.3 消圈算法

消圈算法是一个朴素的贪心算法，可以用来解决无源汇最小费用可行流，由于其极低的效率，只有理论价值，而无实用价值。

消圈算法的流程为不断在有容量的边上找一个负环，沿着负环推流，直到图上没有负环。接下来证明它的正确性。

> 考虑消圈算法求出的答案，设其流量矩阵为 $f(i,j)$，最优解的流量矩阵为 $f'(i,j)$，有 $\sum_{i,j} f(i,j)w(i,j) \geq \sum_{i,j} f'(i,j)w(i,j)$。若取到等号，那么消圈算法求出的答案是正确的。否则建立新图，对于所有 $i,j$，若 $f(i,j) < f'(i,j)$，加入边 $(i,j,f'(i,j)-f(i,j),w(i,j))$，否则加入边 $(j,i,f(i,j)-f'(i,j),w(j,i))$，容易验证在这个新图上推流的效果和在原图上是一样的。由于对于每个点都有出入容量和相等，这个图可以被看成若干个环的和；而由于总费用小于 0，那么至少有一个环的费用小于 0，也就是原图上仍然存在负环，消圈算法没有结束，矛盾。于是消圈算法可以得到无源汇最小费用可行流的正确答案。

消圈算法非常简单，与之相应的，效率非常低下。它是下面几个算法的思想来源。

## 3.4 SSP 算法

### 3.4.1 算法简介

SSP（Successive Shortest Path）是一个贪心的算法，是信息学竞赛中最常用的算法。由于和 EK 算法流程类似，也有很多人称它为 EK 费用流。目前的大多数考察建模的费用流题都可以用这个算法通过。它贪心地每次选择费用最小的可行增广路并进行推流，于是可以顺便求出限制流量时的答案。

### 3.4.2 算法流程

基础的 SSP 算法十分简单。

1. 忽略目前容量为 0 的边，使用 SPFA 求出从 $s$ 到 $t$ 的最短路，若 $s$ 无法到达 $t$，算法终止；
2. 对这条最短路进行推流，回到 1。

它无法处理负环且较为缓慢。由于这样的原因，有下面这些优化：

对存在负权的边运行单源最短路的消耗相当大，于是我们希望尽量减少这样的次数。借用 Johnson 全源最短路算法的思路，为每个结点设置一个权值 $h_x$ 使得对于所有的容量不为

0 的边 $(u,v,w)$ 有 $h_v - h_u + w \geq 0$，那么以 $h_v - h_u + w$ 作为边权就不存在负权边了，可以运行更加高效的 Dijkstra 算法。$h_x$ 一般设置为 $s$ 到 $h_x$ 的最短路，而这是容易更新的。这个技巧也被称为 Primal-Dual 原始对偶算法，可以将多次负权最短路优化为最多 1 次负权最短路和多次非负权的单源最短路，在图较大且对 SPFA 不友好时时效果明显，但在随机的图上 SPFA 并不慢，于是在随机的图上没有作用。

对于费用较小的图，最短路可能不止一条，最短路的并形成了一个图。此时可以仿照 Dinic 算法的多路增广优化，在图上进行 dfs 进行多路增广。不同点在于，Dinic 运行时图被严格分层，但在 SSP 算法进行增广时，常常出现一个混乱的边权全部为 0 的连通块，此时需要精细处理才能兼顾复杂度和常数。这是一个常数优化，在费用较小的图上一般表现良好，但是在费用范围较大时具有反效果。

### 3.4.3 算法分析

这个算法的朴素版本在无负环的有源汇最小费用最大流问题正确性上的证明可以参考 Dinic 算法的证明 [1] 和 3.3 中的证明。这个证明也说明了在增广的过程中可以求出所有流量时的最小费用，也可以求出最小费用可行流。

在一般的稀疏图上，SSP 算法是 $O(nm + (n\log n + m)f)$ 的，其中 $f$ 表示总流量。这是一个伪多项式的时间复杂度，说明在某些数据上十分缓慢。但是由于和 $f$ 有关，在流量较小时可以直接使用。

将 SSP 算法卡到 $O(nm + (n\log n + m)f)$ 相当困难。文章 [2] 给出了一种在 $m = O(n^2)$，$f = O(2^{\frac{n}{2}})$ 的情况下将时间复杂度卡到 $\Omega(n^2 2^{\frac{n}{2}})$，达到上界，但这个构造相当精巧，在大部分题目中无法出现。论文 [3] 指出，当边的费用生成得相对平滑（指定 $[0,n]$ 的整数部分，小数部分随机生成）时，即使图的形态和容量是任意的，时间复杂度仍有 $O(nm^2\log m)$ 的上界。这说明在大多数题目中，并不会出现让 SSP 算法到达指数级的情况。与此同时，一些常用的特殊的图上 SSP 算法可以分析出更优秀的时间复杂度，尤其是在使用了多路增广的常数优化之后，经常可以分析出优于 $O(mf)$ 的时间复杂度，有时甚至可以处理 $m \leq 2\times 10^5$ 的数据。在随机生成的图上表现同样良好，不使用原始对偶算法时，可以在 0.05 秒内通过图随机生成的例题 1。

但是，一些费用流问题的图是任意的，这让出题人将算法卡到指数级并不困难。这个算法难以通过例题 2。

## 3.5 Capacity Scaling 算法

### 3.5.1 算法简介

这是一个基于对容量二进制分解的算法，被用于解决无源汇最小费用可行流问题。因为在大部分图上性能不及 SSP 算法，在 OI 中使用的人不多。

### 3.5.2 算法流程

1. 令 $p$ 为 $\lfloor \log_2 U \rfloor$，其中 $U$ 是容量的上界；建立一个 $n$ 个点的空费用流图 $G$。
2. 枚举所有原费用流图上的边 $(u, v, w, c)$，若 $2 \nmid \lfloor \frac{c}{2^p} \rfloor$，则在 $G$ 上加入边 $(u, v, w, 1)$，加入后检查是否存在负环。如果存在，沿着权最小的简单负环推流即可。
3. 将 $G$ 的所有边的容量乘 2（同时流量和答案也乘 2），将 $p$ 减小 1。
4. 若 $p \geq 0$，则回到 2。

算法过程中，$G$ 中加入的边的总和和原图等价；由于每次只会加入一条容量为 1 的边，不会因为增广而出现新的负环。实际实现中，无需建立新图，只需要记录原图上每个边的容量，然后直接对它进行更改即可。这样总边数为 $O(m)$ 而非 $O(m \log U)$。

加入边后检查是否存在负环这一步骤相当于在加入边前求 $v$ 到 $u$ 的最短路。这可以直接套用 3.3.3 种的原始对偶优化技巧，同样可以只进行一次 SPFA。

### 3.5.3 算法分析

算法正确性十分平凡：算法过程中的操作只有将所有边容量乘 2 或将一条边的容量加 1，若进行一次推流后仍存在负环，那么它就不是权最小的简单负环。

这是一个 $O(m \log U(n \log n + m))$ 的算法，也就是说，这是一个弱多项式做法，对于图较小（从而 $m$ 较小）但流量较大（$f$ 较大，从而 SSP 算法较慢）是有显著优势。但是，这个算法的流程是相对固定的，无论是在随机生成的图上还是在有优秀特殊性质的图上，相比 SSP 算法并没有优势。Capacity Scaling 算法更多的是给出了一个伪多项式时间复杂度的上界，同时比较容易实现，它可以通过例题 2。在更小的数据范围，可以直接使用 SPFA 而不使用原始对偶优化，有 $O(nm^2 \log U)$ 的时间复杂度，但实际运行速度更快。

## 3.6 Network Simplex 算法

### 3.6.1 算法简介

在 3.4.3 的时间复杂度分析中，可以发现网络流算法的最坏情况相当难以分析，虽然有着较劣的理论最劣复杂度，但是在实际运行中效率一般很高。在线性规划中，一个算法有着类似的性质：单纯形法（Simplex）：即使它的最劣时间复杂度是指数级的，即使多项式时间复杂度的椭球法（Khachyian method）被很早地发明出来，单纯形法在不大的随机数据中依然是最优的选择。一些研究指出，这得益于较低的期望转轴次数和较小的常数 [6]。

费用流问题也是一种线性规划问题，于是一些人对线性规划进行了特化，创造了单纯形费用流（Network Simplex）算法。它是对单纯形法在网络流图上的优化，所以拥有单纯

形法的一些优点，同时和单纯形法一样拥有最坏指数级的时间复杂度和较高的效率，还不需要初始的一次代价较大的 SPFA。下面将 Network Simplex 简称为 NS。

### 3.6.2 算法流程

NS 的流程和单纯形法非常类似，可以参考这里 [7] 来理解单纯形法。这里给出一种理解和实现方式。

无源汇最小费用可行流模型容易改写成一个线性规划问题：

- $\min \sum_i f_i w_i$
- $\sum_i ([u_i = x] - [v_i = x]) f_i = 0$
- $f_i \le c_i$，可以写成 $f_i + g_i = c_i$
- $f_i \ge 0$，$g_i \ge 0$。

这个线性规划问题有 $n$ 个限制形如 $\sum t_i f_i = 0$，总共有 $2m$ 个变量。我们可以直接抄一遍单纯形法的变量有上限版本：

1. 在原版线性规划中，基变量定义为非零的变量；在这里，当 $f_i = c_i$ 或 $g_i = c_i$ 时，认为它们是非基变量。
2. 任意选择一组基。共有 $n + m$ 条限制，但是其中一条是冗余的，于是有 $n + m - 1$ 条有效限制，在原版线性规划中会有 $n + m - 1$ 个基变量。考虑到当 $f$ 和 $g$ 有一个为 0 时，会多算一个基变量，于是在上面的定义下，有 $2n - 2$ 个基变量。在实现中，只需要处理 $f$，而 $f$ 和 $g$ 同时是或不是基变量，于是最终只需要处理 $n - 1$ 个 $f$ 即可。容易发现其实它是任意一棵生成树，其他的所有边都在一个方向上被满流。
3. 选择初始基本可行解。所有 $f$ 取 0 满足条件。
4. 转轴。选择一个非基变量，对应一条从一个方向上满流的边；考虑将它改变后的影响，对应从另一个方向上推流；由于基变量形成一颗生成树，于是对基变量的影响相当于在生成树上的一条链上推流。推流时只需要求出路径上剩余流量最小的边即可，判断推流是否更优只需要求链上的和即可。
5. 终止条件。显然无法更优就终止。
6. 选择转轴规则。最朴素的实现是枚举每条边，然后一旦能转轴做到更优就进行转轴并不断循环。

我们可以进行朴素的实现，暴力维护生成树然后每次求距离时直接暴力求 LCA。这已经基本上是目前在随机图上最快的算法了，下文会分析其原因。

### 3.6.3 算法优化

对 NS 算法的优化有几种不同的思路。

维护生成树时，如果图相对随机，那么生成树的期望高度不大，于是暴力维护由于其较小的常数而优于其他的方法。但是，当图的形态较为特殊时（如一条主链加一些边），生成树的高度到达 $\Omega(n^{1-\epsilon})$ 时，暴力维护生成树成为瓶颈。生成树需要支持的操作比较简单，可以使用 LCT 维护。在随机图上，这样的优化并不会显著提高效率，但是在特殊的图上是一个有效的优化。

选择转轴规则也是一个可以优化的地方。有两种基本策略：第一种是不断循环枚举非基变量（边），然后取第一个转轴变优的；第二种是每次考察所有边，取转轴收益最大的。这两种之间还可以平衡出第三种：设定阈值 $B$，在每 $B$ 条边之间使用策略二，然后全局使用策略一；一般取 $B = O(\sqrt{n})$。

### 3.6.4 算法分析

NS 算法的本质是每次消去一个负环，也就是说，可以被看做消圈算法的优化，其所有其他部分都是为了运行速度而进行的优化。和前身单纯形法一样，朴素的 NS 的最劣时间复杂度是指数级的，并且同样难以触及上界。由于其极高的效率，是工程中最常用的算法。

是否存在简单的转轴规则使得转轴次数被控制在多项式复杂度内是一个开放问题，目前所有的简单策略都存在指数级的最劣复杂度 [6]。但是，NS 算法的适用范围并不是这些刻意构造的图，而是相对随机的图。平滑分析给出了 $\Omega(m\phi \min(n, \phi))$ 的下界和 $O(n^2 m \log \phi \log n)$ 的转轴次数上界 [6]，说明了即使是恶意构造的反例也是十分脆弱的。

NS 算法的主场是相对随机的图。文章 [5] 中给出了朴素实现下 $O(nm)$ 的期望运行时间（不使用 LCT，转轴使用策略三），同时暴力的实现也让它的常数因子很小。有多种方法说明 NS 高效的原因：NS 作为单纯形法的特化，同样有单纯形法“转轴次数少”的优点；在随机的图上，随意挑选的生成树一般直径很小，于是暴力实现动态树同样高效；相当简单的流程使得算法易于修改，可以很简单地做在链上的特化，等等。

### 3.6.5 在 OI 中的意义

在信息学竞赛中，知道这个算法的人相当少。这导致很少有人会专门为了让这样一个生僻的算法无法通过而研究如何卡这个算法，于是目前大部分图对这个算法来说于随机图具有类似的性质，从而有着极高的效率。除了例题 1 和例题 2 之外，一些数据范围极大的网络流题也可以通过。

一个重要的事实是一个算法如果具有高推广价值，那么了解它的人会越来越多，于是总有一天大家都知道如何构造它的最劣情况。例如，SPFA 在刚被发明有着类似的特性，常数较小，在任意生成的图上具有近似 $O(m)$ 的时间复杂度。但是，随着 SPFA 的推广，越来

越多的人知道网格图和菊花图是 SPFA 的缺点之一，而网格图相当常见，于是便有人说“关于 SPFA：它死了”。随机增广的一般图匹配算法同样类似，在卡它的方法出现之前，它全面优于带花树，还更易学，但一旦对应的构造被提出，这个算法在 OI 中就被逐渐淘汰。

NS 有着和 SPFA 类似的性质：常数极小，在不刻意卡的情况下比各种费用流算法甚至网络流算法优秀，于是在尚未推广的情况下可以用来获取意料之外的分数。但当它在 OI 内部推广开后，NS 也许会迎来它在 OI 内的谢幕。但是，在 OI 之外的地方，只要 NS 保持了在相对随机的图上最优秀的运行效率，它就会保持在工程上的最高的应用频率。

### 3.7 总结

费用流问题是学术上的经典问题，在最近学术界的研究中还提出了一些近线性的算法[7]，但在 OI 界对其了解并不多。消圈算法和 SSP 可以说是最朴素的贪心算法，剩下的算法都基于这两个算法的思维基础优化而来。它们的思想分别是消除负环和一次增加尽可能少的费用。在其他算法和下面的优化中，处处都有这两个思路的影子。

在实用层面，SSP 算法是 OI 中最经典的算法。对于复杂度要求不高的数据，可以使用 Capacity Scaling 的 SPFA 版本，而对于随机生成的图，NS 算法是最好的选择。当复杂度要求变高时，一些活在论文里的算法也开始出现应用。

## 4 模拟费用流和反悔贪心

### 4.1 引入

在上面的算法中，所有的普适的费用流算法都是 $\Omega(nm)$ 的。这很好理解，因为即使只有 1 的流量，我们也需要求出从源点到汇点的最短路，而这是难以更快的。

但是实际做题时所遇到的图往往更加特殊。这是因为很少有题只是给你一张图然后询问最小费用最大流的，更多的题是根据题目的性质建立一张极为特殊的图然后求最小费用。

类似 NS 对单纯形法的特化，我们也可以对费用流进行特化让它能解决更大范围的问题，例如下面这一类题。根据对费用流算法的选择，可以分为对消圈算法的优化和对 SSP 算法的优化。

### 4.2 一类链上问题的模型

#### 4.2.1 消圈的模型

[3] 来源: CF865D Buy Low Sell High https://codeforces.com/contest/865/problem/D

**例题 3：Buy Low Sell High**[3] 有一种物品，接下来 $n$ 天内，$i$ 天后的价格为 $a_i$，每天你可以买入或卖出一件物品，初始没有物品，问最多能获利多少。$n \le 2 \times 10^5$。

本题对应的网络十分简单：总共 $n+2$ 个点，$s = n+1, t = n+2$，有边 $(i, i+1, \infty, 0)(i < n)$ 和 $(s, i, 1, a_i), (i, t, 1, -a_i)$，需要求出最小费用。补上边 $(t, s, \infty, 0)$ 后，我们可以使用 NS 算法（作为消圈算法的优化），有可能直接通过，但这取决于具体的实现细节，这里不多讲。注意到当 $s \to i$ 的边和 $i \to t$ 的边都有流量时，可以选择环 $s \to t \to i \to s$ 的总代价为 0，进行增广不影响正确性。于是默认要求 $s \to i$ 和 $i \to t$ 的边至多有一条有流量。接下来考虑对这个图运用消圈算法并进行优化。

对于本题，对于 $i : 1 \to n$，依次考虑经过点 $i$ 且不经过 $(i, n]$ 的点的增广环。此时 $i$ 只有 $i \to t$ 一条出边，于是一定会经过这条边；同时 $s \to i$ 的边权和 $i \to t$ 的边权和为 0，于是不会选择 $s \to i \to t$ 的路径，那么找到的增广环一定经过 $i-1 \to i$ 这条边，形如找到一个 $x < i$ 且 $s \to x$ 无流量且有 $a_x < a_i$，增广环即为 $s \to x \rightsquigarrow i \to t \to s$，费用为 $a_x - a_i < 0$；增广后若 $x \to t$ 有流量，此时存在环 $x \to s \to t \to x$ 费用为 0，可以保持 $s \to i$ 和 $i \to t$ 的边至多有一条有流量。按照顺序进行增广，每次选择最小的 $a_x$，可以证明增广后不会出现新的负环，这也是这类题的重要特性，具体证明可以参考 4.2.3。

对上面的思路进行简化。实际上，只需要记录 $s \to x$ 的流量和 $x \to t$ 的流量之差，记为 $c_x$。当考虑加入点 $i$ 时，$c_i = 0$；若对于 $i$ 找到了一个 $x < i$ 使得 $c_x < 1$ 且 $a_x - a_i < 0$ 最小，那么进行增广可以被简化为记录答案并令 $c_x \leftarrow c_x + 1$，$c_i \leftarrow c_i - 1$。当处理完 $i$ 后，$c_i$ 不会减少，于是可以用 $1 - c_x$ 个 $a_x$ 来记录可行的决策。使用堆维护即可。时间复杂度为 $O(n \log n)$。

这题可以很简单的用反悔贪心来理解：到第 $i$ 天时，贪心的选择一个之前价格最低的日期买入并在第 $i$ 天卖出；如果可行，那么第 $i$ 天相当于可以买入两次：第一次撤销卖出，第二次才是买入。于是反悔贪心和消圈特化殊途同归，得到了一样的做法。

**例题 4：【UER #8】雪灾与外卖**[4] 数轴上有 $n$ 个点，第 $i$ 个点在 $d_i$ 处，有 $a_i$ 个球，有一个容量为 $b_i$，进洞费用为 $w_i$ 的洞。球移动一单位距离有 1 的费用，问所有球都进洞的最小费用。

$n \le 2 \times 10^5$，$\sum a_i \le \min(2 \times 10^5, \sum b_i)$。

本题的网络同样简单：共 $n+2$ 个点，$s = n+1, t = n+2$，有边 $(i, i+1, \infty, d_{i+1} - d_i), (i+1, i, \infty, d_{i+1} - d_i), (s, i, a_i, -\infty), (i, t, b_i, w_i)$。同样补上边 $(t, s, \infty, 0)$ 后可以使用 NS，不过这里难以通过。为了方便，在输入时默认在 $-\infty$ 出有 $+\infty$ 个洞。由于保证了 $\sum b \ge \sum a$，最终最优解一定不会使用这个洞。

首先同样从左到右进行考虑，考虑点 $i$ 时只考虑编号 $\le i$ 的点和 $s, t$ 构成的图。考虑边 $(s, i, a_i, -\infty)$ 时，新增的点 $i$ 和 $s, t$ 组成了 $s \to i \to \cdots \to t$ 的结构，只要把 $i \to \cdots \to t$ 填充即可。这相当于要找到一个 $x$ 使得从 $i$ 走到 $x$ 的边权和最小且 $x \to t$ 的边有容量（由于 $x \to s$ 的边边权为 $+\infty$，无法进入）。可以证明，在左侧没有负环的情况下，进行单一的一次增广不会出现新的负环，具体的证明参考 4.2.3。本题中，还可以证明当一个端点被增广路跨过

[4]来源: UOJ455【UER #8】雪灾与外卖 https://uoj.ac/problem/455

后就可以在最优解中不再被选择作为端点，证明只需要考虑跨过的一次增广不选这个端点说明这个后缀不优，下次选这个端点时可以调整至这次的左端点即可，同样参考 4.2.3。

这样考虑用堆维护每个增广经过的段，每次当一个负环跨过两个负权段时将它们合并即可，在实现上容易发现被合并前更左侧的增广路一定不优，于是可以不用动态更新那一部分。容易发现每一次增广最多新增一个负权段，而有效增广次数是 $O(\sum a_i)$ 的，所以总时间复杂度为 $O(n\log n)$。这一部分的实现细节可以参考下文的 4.5.2 部分，其中描述的算法与这个本质相同。

### 4.2.2 SSP 的模型

**例题 5：k-Maximum Subsequence Sum**[5] 维护一个长度为 $n$ 的整数序列 $a$，有 $q$ 次操作，每次操作为以下两种之一：

- 给出 $i, v$，令 $a_i \leftarrow v$；
- 给出 $l, r, k$，求在 $[l, r]$ 中选出 $k$ 个不交的子区间 $l_1 \le r_1 < l_2 \le r_2 < \cdots < l_k \le r_k$，求 $\sum_{i=1}^{k}\sum_{j=l_i}^{r_i} a_j$ 的最大值。

$n, q \le 10^5, k \le 10, |a_i|, |v| \le 10^9$。

本题的网络也大体在一条链上：总共 $n+5$ 个点，$s = n+2, t = n+3$，有边 $(i, i+1, 1, a_i)(i \le n)$，$(n+5, t, n, 0)$；对于一次询问 $l, r, k$，有边 $(s, n+4, k, 0)$，$(n+4, i, 1, 0)(i \in [l, r])$，$(i+1, n+5, 1, 0)(i \in [l, r])$，每次求最大费用最大流。

由于链上的边容量为 1，只有两种状态，容易想到用线段树来维护。设 $c_i$ 表示边 $i \to i+1$ 的流量，那么选择一条增广路径 $s \to x \to y \to t(x < y)$ 的条件是所有 $i \in [x, y)$ 有 $c_i = 0$ 且 $s \to x, y \to t$ 有剩余容量，权值为 $\sum_{i=x}^{y-1} -a_i$，进行增广就是将 $[x, y)$ 的 $c$ 改为 1；选择增广路径 $s \to y \to x \to t(x < y)$ 的条件是所有 $i \in [y, x)$ 有 $c_i = 1$ 且 $s \to y, x \to t$ 有剩余容量，权值为 $\sum_{i=y}^{x-1} a_i$，进行增广就是将 $[y, x)$ 的 $c$ 改为 0。这些都是容易使用线段树维护的，且支持修改，于是可以直接套用 SSP 算法，每次求出权值最小的增广路径并进行增广，单次操作为 $O(\log n)$，总时间复杂度为 $O(nk\log n)$。

**例题 6：April Fools' Problem (hard)**[6] 有两个长度为 $n$ 的正整数序列 $a, b$，你需要选出 $k$ 对下标 $i_1 \le j_1, i_2 \le j_2, \ldots, i_k \le j_k$，最小化 $\sum_{l=1}^{k} a_{i_l} + b_{j_l}$。对每个 $k \in [1, n]$ 求出答案。

$n \le 2 \times 10^5$。

本题原题为对单个 $k$ 求答案。这里存在进行 WQS 二分并使用类似于例题 3 解法的做法。但这个做法无法通过修改后的题目。

[5]来源: CF280D k-Maximum Subsequence Sum https://codeforces.com/contest/280/problem/D

[6]来源: CF802O April Fools' Problem (hard) https://codeforces.com/contest/802/problem/O

本题的网络与例题 3 高度类似，仅仅是将边 $(i,t,1,-a_i)$ 改为 $(i,t,1,b_i)$。但是本题需要限制流量，于是 NS 算法无能为力，考虑和例题 5 一样优化 SSP 算法。

由于只有向右的边，和例题 5 同样可以记录 $c_i$ 为边 $i \to i+1$ 的流量。选择增广路径 $s \to x \to y \to t(x<y)$ 的条件仅有 $s \to x, y \to t$ 有容量，权值为 $a_x + b_y$，进行增广将 $[x,y)$ 的 $c$ 加 1；选择增广路 $s \to x \to y \to t(x>y)$ 要求 $s \to x, y \to t$ 有容量且 $\prod_{i=y}^{x-1} c_i \neq 0$，权值同样为 $a_x + b_y$，进行增广将 $[y,x)$ 的 $c$ 加 $-1$。

使用线段树维护它来优化 SSP 算法。具体来说，由于对 $c$ 的修改只有区间加，且 $c$ 始终非负，且只需要关心 $c$ 是否为 0，对于每个线段树区间只需记录 $c$ 的最小值和假设进行全局减令区间最小值为 0 时的答案即可转移，而维护 $\min_{x<y} a_x + b_y$ 并非难事。单次操作时间复杂度为 $O(\log n)$，总时间复杂度为 $O(n \log n)$。

**例题 7：【UER #8】雪灾与外卖改 1**[7] 数轴上有 $n$ 个点，第 $i$ 个点在 $d_i$ 处，有 $a_i$ 个球，有一个容量为 $b_i$，进洞费用为 $w_i$ 的洞。球移动一单位距离有 1 的费用，对于每个整数 $i$，移动 $i \to i+1$ 和 $i \to i-1$ 均只能进行 $m$ 次。对于每个 $k \in [1,n]$，求出让至少 $k$ 个球进洞的最小费用或报告无解。

$n \leq 10^5, m \leq 20$。

本题相较于例题 4 需要对每个 $k$ 求出答案，且有限制每条边的经过次数，于是 SSP 算法相当自然。

例题 6 只有向右的边，于是我们可以维护最小值并更改，而本题的结构更加复杂。由于每条边只有 $O(m)$ 中情况，使用线段树维护时每个区间只需要对每个 $x \in [-2m, 2m]$ 维护进行区间加 $x$ 之后的答案即可。这样单次时间复杂度为 $O(m \log n)$，总时间复杂度为 $O(nm \log n)$。

**例题 8：【UER #8】雪灾与外卖改 2**[8] 数轴上有 $n$ 个点，第 $i$ 个点在 $d_i$ 处，有 $a_i$ 个球，有一个容量为 $b_i$，进洞费用为 $w_i$ 的洞。球移动一单位距离有 1 的费用。对于每个 $k \in [1,m]$，求让至少 $k$ 个球进洞的最小代价。

$n \leq 2 \times 10^5, m \leq 2 \times 10^4$。

当 $m$ 不再有限制时，线段树难以为继。和例题 6 类似，我们只需要处理从源点出发，经过一条边，在链上向一侧走一段，最后走连向汇点的边。具体来说，维护 SSP 算法的过程可以转化为如下数据结构问题：

有四个长度为 $n$ 的整数序列 $a, b, c, d$，有如下几种操作：

- 对 $a$ 和 $b$ 进行单点修改；
- 给出 $[l,r] \subseteq [1,n], v \in \{-1,1\}$，令 $c_{[l,r]}$ 加上 $v$；
- 求 $\max_{l \leq r} a_l + b_r + \sum_{i \in [l,r]} ([c_i \geq 0] - [c_i \leq 0]) d_i$

[7] 来源: UOJ455 【UER #8】雪灾与外卖 https://uoj.ac/problem/455

[8] 来源: UOJ455 【UER #8】雪灾与外卖 https://uoj.ac/problem/455

在本题中，$c_i$ 表示 $i$ 和 $i+1$ 之间的流量，于是转化显然。考虑进行分块。将序列 $B$ 个一块分成 $O(\frac{n}{B})$ 块，对于每一块都记录进行 $c$ 的任意加减后的结果。由于一块内只有 $O(B)$ 个 $c_i$，($[c_i \geq 0] - [c_i \leq 0]$) 只会变化 $O(B)$ 次，重构时只需要使用类似例题 11 使用线段树进行维护即可。单次推流的时间复杂度为 $O(\frac{n}{B} + B\log B)$，总时间复杂度为 $O((m+\frac{n}{B})(\frac{n}{B} + B\log B))$。取 $B = \frac{n}{\log n}$，总时间复杂度为 $O(n\log n + m\sqrt{n\log n})$。

本题也给出了一类链上模拟费用流问题的通解，缺点是时间复杂度较高。

### 4.2.3 为什么可以这样做

对 SSP 的特化是朴素的：只是把其中的一个部分使用数据结构维护，并没有改变算法的本质，而链十分适合使用数据结构进行维护；而对消圈的特化就不是很显然，因为在这里每次只考虑了部分的图，难以考虑不包含新加入点的负环。例题 3 和例题 4 也代表了两大类题：单向匹配和双向匹配。

考虑归纳证明。当 $[1, i-1]$ 中没有负环时，需要证明加入点 $i$ 并处理所有包含点 $i$ 的费用最小的负环后仍然没有负环。这里和 3.3 类似，如果存在新的负环，那么它一定和经过点 $i$ 的一个负环相交（否则 $[1, i-1]$ 中没有负环，也不会凭空出现新的负环），而将与之相交的负环调整后可以得到费用更小的负环，于费用最小矛盾。于是可以证明处理完成后不会出现新的负环。

上面的证明的核心点在于考虑到点 $i$ 时，$[1, i-1]$ 的内部已经计算处了一种最优的子结构，在费用流的意义下，它里面没有自环；于是加入点 $i$ 相当于加入 $O(1)$ 条边并在残量网络上运行最小费用流，根据前面的分析，这与直接对整张图运行是等价的。

这是一个相当普适的证明。不仅仅在链上，在一些性质良好的图上，只要能给出一个维持子结构的图，都可以套用这个证明。

## 4.3 更一般的模型

### 4.3.1 增广路较短的图

**例题 9：[USACO12FEB] Cow Coupons G**[9] 有 $n$ 个物品，第 $i$ 个物品的价格是 $a_i$，你可以使用 $m$ 次魔法，一次魔法选择一个 $i$，将物品 $i$ 的价格改为 $b_i$。问 $k$ 的钱最多能买多少物品。

$m \leq n \leq 10^5, a_i > b_i$。

本题的网络是一个二分图：有 $n+4$ 个点，$s = n+1, t = n+2$，有边 $(s, n+3, n, 0), (s, n+4, m, 0), (n+3, i, 1, a_i), (n+4, i, 1, b_i), (i, t, 1, 0)$，求费用不超过 $k$ 时的最大流量。强化为求每个流量时的最小费用。

[9]来源: P3045 [USACO16DEC] Cities and States S https://www.luogu.com.cn/problem/P3045

在这里的增广路很短，长度最多为 5。由于一侧只有两个点 $n+3, n+4$，使用堆记录它们出边的最小权值和反向边的最小权值和即可。如果进行讨论，那么所有可能的增广路对应以下几种操作：

- 用 $a_i$ 价格购买 $i$；
- 用 $b_i$ 价格和一次魔法购买 $i$；
- 用 $a_j - b_j + b_i$ 价格撤回在 $j$ 物品上的魔法并改为对 $i$ 使用魔法，购买 $i$。

使用堆维护即可。

**例题 10：Poor Students**[10] $n$ 个不同的物品放进 $k$ 箱子里，第 $i$ 个物品放入第 $j$ 个箱子有代价 $a_{i,j}$，第 $j$ 个箱子有容量 $c_j$。求最小代价。

$n \le 5 \times 10^4$，$k \le 10$，$\sum c_j \ge n$。

本题的网络也是一个二分图：有 $n+k+2$ 个点，$s = n+k+1, t = n+k+2$，有边 $(s, i, 1, 0), (i, j+n, 1, a_{i,j}), (j+n, t, c_j, 0)$，求最小费用最大流。这其实时例题 9 的拓展。

由于这是一个二分图，那么增广路一定每两步经过一次右部点，而右部点只有 $k$ 个，于是考虑在右部点之间建边，边权表示两点间的距离。那么每次增广只需要在右部点内部跑一次最短路即可，而最终的增广路唯一地对应着右侧的一条路径，选择费用最小的增广即可。

考虑维护右部点之间的边。由于右部点之间的每次移动都形如 $j_1 + n \to i \to j_2 + n$，只需要对每个 $(j_1, j_2)$ 维护 $i$ 的集合使得 $j_1 \to i$ 和 $i \to j_2$ 有容量。每次增广只会改变 $O(k)$ 个信息，于是同样使用堆即可维护。

### 4.3.2 树上问题

**例题 11：征服世界**[11] 有一颗 $n$ 个点的树，点 $i$ 上有 $a_i$ 个球，一个容量为 $b_i$ 的洞。一个球可以沿着一条边 $(u_i, v_i)$ 移动并付出 $c_i$ 的代价，求让所有球进洞的最小代价。

$n \le 2.5 \times 10^5$，$\sum a_i \le \sum b_i \le 10^6$。

本题的网络为例题 4 中的链改为树，且 $w_i = 0$ 的情形，于是我们也有类似例题 4 的结论。这里仍然需要以一个顺序进行增广，且需要保证前面的点不会走到后面的点，而在一颗树上，这是难以做到的，其原因在于一个点可以有大于两个邻居，从而无法简单处理多方均需要增广时的问题。

需要注意到一次增广一定沿着一条链走了一段，然后两端分别连向源汇点，于是可以考虑在链的最浅处处理这次增广。具体来说，仿照例题 4 的处理方式，dfs 到 $x$ 时维护两个

[10] 来源: QOJ7185 Poor Students https://qoj.ac/problem/7185

[11] 来源: LOJ6405「ICPC World Finals 2018」征服世界 https://loj.ac/p/6405

堆表示 $x$ 子树内的合法流的集合；当两个堆都不为空时，显然我们正在对应的 LCA 处，于是可以仿照例题 4 将两个堆的内容合并，直到其中一个堆为空。容易可以证明它的正确性。

**例题 12：征服世界改**[12] 有一颗 $n$ 个点的树，点 $i$ 上有 $a_i$ 个球，一个容量为 $b_i$ 且进洞费用为 $w_i$ 的洞。一个球可以沿着一条边 $(u_i, v_i)$ 移动并付出 $c_i$ 的代价。对于所有 $k \in [1, m]$，求出让至少 $k$ 个球进洞的最小代价。

$n \le 10^5, m \le 10^4$。

本题和例题 8 一样要求对每个 $k$ 求出答案，仿照例题 8 进行处理。本题中，我们同样只需要求从源点出发，经过一条边，在链上走一段，最后走连向汇点的边。与例题 8 相似，各种链剖分难以处理这样的问题，于是我们可以进行树分块。

具体来说，运用 Top Cluster 树分块，将树分成 $O(\frac{n}{B})$ 个大小为 $O(B)$ 的块，每个块只有两个界点与其他块公用。对每个块，记录链上所有 $c$ 和对应 $c$ 变化时的信息。由于界点信息变化次数巨大，所以需要单独记录两个界点并将路径按照选择的界点分类，对每类维护一段在内部（不含界点）任选，两端都在内部（不含界点）任选，两端都在外部的三种情况，还需要分两类处理。这样，每次询问最短路时，可以在压缩树上直接 dfs 得到答案，做到 $O((m+\frac{n}{B})\frac{n}{B})$ 的总复杂度。当更新涉及到簇路径之外的边时，需要更新整个块。此时，进行一次 dfs 即可类似于在压缩树上 dfs 地得到每个点上挂的信息，于是问题和例题 12 是类似的，同样可以做到单次 $O(B\log B)$ 的时间复杂度。总时间复杂度为 $O((m+\frac{n}{B})(\frac{n}{B}+B\log B))$，取 $B=\sqrt{\frac{n}{\log n}}$ 有 $O(n\log n + m\sqrt{n\log n})$。

本题给出了一个树上模拟费用流的通解，它也可以拓展到更多的图上。

### 4.3.3 算法的边界

SSP 算法和消圈算法的特化方式迥异，它们能处理的问题的边界也不尽相同。接下来，我对这两个算法的边界进行讨论。

SSP 算法有一个很明显的边界，那就是无论如何改写，推流次数都是不变的。也就是说，对于推流次数极大时，SSP 算法一定无法解决问题。在一些问题中，推流次数达到了 $10^9$ 甚至 $10^{18}$ 量级，如果它还需要对多个流量询问最小费用，那么无论是 SSP 还是消圈，都对此无能为力，一般需要分析更好的性质并使用其他算法来解决。除了推流次数外，单次推流的时间复杂度同样需要关注。在例题 8 和例题 12 里，即使图只是一个有双向边的链或树，已经让维护的难度飙升至根号级别且带有不小的常数。对于更加复杂的图来说，广义串并联图可以在收缩树上进行类似 Top Cluster 分块的算法并对块内的使用类似例题 12 的处理方式，但此时的常数因子已不可忽略，让它没有实用价值；对于更加一般的图来说，无论如何都会面临需要求出最短路的问题，而这很难比 Dijkstra 更简洁，从而无法存在普适的算法进行优化。

[12]LOJ6405 「ICPC World Finals 2018」征服世界 https://loj.ac/p/6405

消圈算法相当灵活，但它的硬伤在例题 8 中体现出来，即使对于例题 8 的原题（即对单个 $k$ 求答案）存在使用 WQS 二分的算法，但是仍然会多出一个 log 因子，让它相较于 SSP 不优。如果不考虑需要对每个 $k$ 求答案的问题，消圈算法几乎是 SSP 算法的上位：对于可以以合并为界分割的图，消圈算法都可以仿照例题 11 进行操作，如在广义串并联图上同样可以做到多对数复杂度且常数没有显著增加；对于任意的图，3.6 中的 NS 算法已经相当优秀。在一些简单的情况下，如果题目固定了流量，由于费用流算法的凸性是基本的，问题可以通过 WQS 二分多一个 log 因子的方式转化为不固定流量的问题。

需要注意的是，对于不属于广义串并联图的图，仍然可能存在可接受的模拟费用流算法，但这些图和对应的算法难以总结共性，所以这里无法列出。在广义串并联图上，最短路的性质较为复杂，于是例题 11 中的合并操作也会难以实现，并不朴素。

## 4.4 对特化的分析

SSP 算法本身并不随意，增广路的限制严格，这让动态维护增广路十分困难，于是在问题上有着更高的时间复杂度。但是，SSP 算法中，增广路的性质更加简单，也能自然地对每个 $k$ 求出答案，这是其他算法不具备的能力，这也是 SSP 算法的优势。

对于不需要求出所有答案的题目来说，更加灵活的消圈算法一般更加适用。这种情况下，一般维持一个最优的子结构，然后逐渐加入并维护它使其保持最优，而类似于 4.2.3 的证明，在几乎所有这样的情况下都可以证明类似的结论，即“增广后不存在负环”。由于维护前缀信息一般相对简单，对消圈算法的特化会更快一些。

网络流是费用流在费用为 0 时的特例，而对于模拟网络流，很少需要专门指出它在对哪个算法进行特化。那么为什么费用流需要进行区分呢？我们可以从求出的答案来找到一些端倪。

对于网络流问题，无论是哪个算法，最终需要求的只有最大流一个数字。从消圈算法的角度来理解，当所有其它边的费用都是 0 时，任意增广得到的答案时不变的，且不需要考虑达到最大流之后的问题。于是，对于单纯的模拟运行最大流算法的算法而言，实际上并不需要受限于 Dinic 或 HLPP 的增广顺序，而只是以某种特定顺序更快地实现了类似 FF 算法的推流而已。对于其他的算法，同样不需要拘泥于对任意图均有效的各个算法，可以用不同的角度，如最小割，来直接计算，同样与具体的算法无关。

对于费用流而言，任意增广的 FF 不再正确，而消圈算法扛起了任意增广的大旗。在费用流这边，只有 SSP 和消圈算法（包括 NS）只需要考虑一个子图的性质，而其他的算法均从全局考虑。于是只要不求出全局的最短路，而是求局部最优解，那是消圈算法，反之沿着全局的最短路进行，那是 SSP 算法，这十分易于区分。与此同时，SSP 算法求出的答案不仅仅是两个数，而是一个凸包，这让给它比 NS 算法能解决的问题更多，尤其是“对所有 $k$ 求出问题的答案”类；与之对应的，维护全局的最短路相当不便，于是一般需要更高的时间复杂度。而消圈算法难以处理“对所有 $k$ 求出问题的答案”类，但即使固定流量 $k$，也可以通过 WQS 二分解决，在 SSP 算法维护复杂度过高时优于 SSP 算法。

## 4.5 模拟费用流的局限性

### 4.5.1 模拟费用流和反悔贪心之间的关系

反悔贪心并没有一个明确的定义，正如模拟费用流也没有，这里给出本文中的定义。反悔贪心指带有取消过去操作的操作（可能取消过去的取消）的贪心算法，这里取消过去的操作一般被称作反悔。这类算法一般先让所有操作均可取消（这样正确性是显然的），然后证明一些操作无需反悔，从而证明整个算法的正确性。

本文中的所有费用流算法及其优化都是反悔贪心，其根本原因在于第二章约定处的反向边的意义：经过反向边代表反悔操作，而反向边的反向边取消了反悔操作，以此类推。将反向边的反向边和原边一同处理，就得到了约定处推流的意义。同样，消圈算法的正确性可以用上文的方式证明：先让所有操作均可反悔（即让所有有流量的环均可推流），再证明一些操作无需反悔（即说明正环无意义，与最优解之间的差全部为负环），然后正确性得证。同样的，上面一些对 4.2.1 的题目也是类似的操作，4.2.3 的证明相当于证明了在最优子结构中插入 $O(1)$ 个元素只会改变 $O(1)$ 个位置（不一定只有 $O(1)$ 个值发生更改，但维护信息只更改 $O(1)$）即可调整至新的最优子结构。所有这样的题目均可以直接用反悔贪心的方式解决。这里对 4.2.3 的证明进行略微推广即可得到这样的结论。

一些反悔贪心问题无法使用费用流描述，这是因为费用流要求了费用是凸的，但一些题目进行反悔不要求凸性，典型的例如“第二个半价”类代价最小的决策可能无法进行的题目。这类题目费用流无法处理，但在反悔贪心的意义下相当普通。

### 4.5.2 反悔贪心问题

4.2.1 的例题 3 中，模拟费用流和反悔贪心殊途同归，得到了相同的做法，其思路与 4.2.3 中的证明并不相同。这里也可以仿照例题 3 来说明其他几题的思路。

回到例题 5，使用反悔贪心的思路讲解。当 $k = 1$ 时答案容易维护，考虑 $k = 2$。$k = 2$ 的答案相比 $k = 1$ 的答案不同的位置一定形成一段，证明考虑调整，如果超过一段可以移动一侧的答案让其更优，那么从 $k = 1$ 到 $k = 2$ 的问题相当于求将一段进行“取反”的最大贡献，将 $k = 1$ 时答案部分取反就仍然是最大子段和问题。同理，从 $k = i$ 到 $k = i + 1$ 修改的位置同样可以证明连续，于是每次只需要区间取反求最大子段和即可。容易发现和模拟费用流写法完全相同，区别在于思路和证明。

对于例题 4，反悔贪心可能更好理解：对于一个洞，它可以匹配前面未匹配的球，也可以将一个之前的匹配拆散并加入一个洞；对于一个球也是同理。写出来代码发现它和模拟费用流甚至可能逐字符相同，这说明它们只是同一种算法的两种不同理解思路。例题 3 和例题 6 也是类似的，这是因为这几题都使用堆维护一段可行流的端点，而一段可行流的端点直接对应（而不是表示）一个可行决策的一端，那么即使计算代价的思路不同，算出来仍然应该是同样的东西。

根据上面的例题 4 和例题 5 可以看出，反悔贪心在大多数时候思路比模拟费用流更加简洁。但反悔贪心的严格证明并不简洁，对于更加复杂的路径如例题 10，同样比模拟费用流难，对于如例题 8 和例题 12 一类的题，模拟费用流相比反悔贪心能更好地从其弱化半拓展到它。这并不是说反悔贪心比模拟费用流更差，而是说同时使用两者的思路对解题会有很大的帮助。而在实现方面，模拟费用流一般会成为证明中“能建立费用流模型，于是能证明答案的凸性或局部最优”的部分。

对于一些不凸的模型，它们可以使用反悔贪心，无法用模拟费用流描述，但对于这类题，4.2.3 的证明经常依然有效，这是因为局部子结构的最优性仍然存在，负环只需要换成别的东西就可以完成修正，于是可以用相同的思路解决。

# 5 总结

本文从费用流问题出发，总结并介绍了消圈算法，SSP 算法，Capacity Scaling 算法和 NS 算法四个算法，分析了在不同数据下其性能和实用性。在对特殊图的讨论中，从链上的模型出发，介绍了链，树，度数小等多种情况的相关算法，给出了链上问题的一个大致同解并探讨了这类模拟费用流算法的局限性和于反悔贪心的关系，也从模拟费用流出发证明了一类反悔贪心的正确性。

费用流问题是线性规划的经典问题，无论是从建模还是优化的角度都有相当深入的研究空间。本文着重介绍了费用流的优化问题，在一些问题中给出了更强的做法，但笔者没有能力证明其最优性或找到更强的做法。希望本文起到抛砖引玉的作用，启发感兴趣的读者能够进一步深入研究，拓展一些结论的适用范围或提出更优的解法，从而得到更有趣的做法和应用。

# 6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢学校的栽培，刘溯老师，黄晶晶老师教导和帮助。

感谢家人、朋友对我的支持和鼓励。

感谢所有在论文编写过程中提出建议的同学。

# 参考文献

[1] oiwiki 最大流层次图层数单调性的证明 https://oiwiki.com/graph/flow/max-flow

[2] min_25. A Bad Example for the Successive Shortest Path Algorithm. https://min-25.hatenablog.com/entry/2018/03/19/235802

[3] Tobias Brunsch and Heiko Röglin. Smoothed Analysis of the Successive Shortest Path Algorithm. In Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '13), 1173–1190, 2013.

[4] Eleon Bach and Sophie Huiberts. Optimal Smoothed Analysis of the Simplex Method. arXiv preprint arXiv:2504.04197, 2025.

[5] A.I. Cash. Network Simplex Algorithm. Codeforces Blog Entry 94190. https://codeforces.com/blog/entry/94190

[6] Kamiel Cornelissen and Bodo Manthey. Smoothed Analysis of the Minimum-Mean Cycle Canceling Algorithm and the Network Simplex Algorithm. *Journal of Graph Algorithms and Applications*, 24(3):397–421, 2020.

[7] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), 612–623, 2022

# 浅谈正则图上的一些问题

广东实验中学 黄建恒

## 摘 要

正则图是一种所有点度数相同的无向图，近年来出现了一些相关的题目，例如二分正则图匹配。本文介绍了两种特殊正则图的性质。在第二章研究二分正则图上的匹配算法，第三章介绍距离正则图的代数性质。

## 1 基本定义与约定

为了方便描述，以下给出一些定义。

如果没有特殊说明，本文中的图均指无重边无自环的无向图，有时也可以拓展到有重边有自环的情况。

对于 $G = (V, E)$，本文中将使用 $n = |V|$ 来表示点集 $V$ 的大小，用 $m = |E|$ 来表示边集 $E$ 的大小。

定义点 $u$ 的度数 $deg(u)$ 为与其相连的边条数，重边重复计算，自环 $(u, u)$ 计算两次。定义 $N(u)$ 表示与 $u$ 相邻的顶点集合。对于点集 $S$，定义 $N(S) = \bigcup_{u \in S} N(u)$

**定义 1.1** ($k$-正则图)**.** 若每个点的度数均为 $k$，则称这个图是 **$k$-正则图**（*$k$-regular graph*）。

## 2 二分正则图

本章的图允许重边自环。

**定义 2.1** (匹配)**.** 图 $G$ 的一个匹配 $M$ 为边集 $E$ 的一个子集，满足 $M$ 中任意两条边没有公共点，称 $|M|$ 为匹配的大小，

**定义 2.2** (完美匹配/1-因子)**.** 如果 $|M| = n/2$，即覆盖了所有点，那么称这是一个完美匹配，此时 $G' = (V, M)$ 满足所有点的度数均为 1。

不难发现 1-正则图的边集本身是一个完美匹配。

## 2.1 二分正则图匹配

**定义 2.3** ($k$-正则二分图)**.** 左右部点数均为 $n/2$，所有点度数均为 $k$ 的二分图。

**定理 2.1.** 任意 $k$-正则二分图均可分解为 $k$ 个完美匹配。

考虑归纳，$k = 1$ 时成立，对于任意 $k$-正则二分图，使用 Hall 定理，对于任意左部点集 $S$，其相关的边数为 $k|S|$，如果 $|N(S)| < |S|$，则说明与 $N(S)$ 相关的边数至少为 $k|S|$，但与 $N(S)$ 相关的边数只有 $k|N(S)|$，矛盾，因此存在完美匹配，删除这组完美匹配后变成 $(k-1)$-正则二分图，证毕。 □

由上述证明，我们不难想到一个算法求解方案：每次使用 dinic 算法求出一个完美匹配，将其删除，重复 $k$ 次，时间复杂度 $O(km\sqrt{n}) = O(k^2 n\sqrt{n})$。

该定理也可以使用 König 线着色定理推出。

**定理 2.2** (König 线着色定理)**.** 对于无向二分图，其边染色的最小颜色数（即色数）等于图的最大度数。

其中边染色定义为给每条边一个颜色，使得任意两条有公共顶点的边不同色。

考虑如下构造性算法：依次加入 $m$ 条边，对于边 $(u, v)$，设 $C(u)$ 为 $u$ 相邻的边颜色集合，定义 $\operatorname{mex}(S)$ 为集合 $S$ 中最小的没出现过的正整数，令 $x = \operatorname{mex}(C(u))$，$y = \operatorname{mex}(C(v))$，将这条边权值赋为 $x$，如果 $x$ 在 $C(v)$ 中出现，设对应的边为 $(u_2, v)$，则将 $(u_2, v)$ 的边改为 $y$，此时 $C(u_2)$ 中没有 $x$，如果 $C(u_2)$ 中原本有 $y$，设为 $(u_2, v_2)$，则将这条边权值改为 $x$，递归下去，即走出一条 $x/y$ 交替的增广路，直到没有矛盾为止。

可以证明，这条增广路不会重复走到一个点 $z$。如果 $z = u$，即存在边 $(u, v_i)$ 的边原本颜色是 $x$，与 $x = \operatorname{mex}(C(u))$ 矛盾。否则说明 $z$ 有两条同色的相邻边，矛盾。因此可以在 $O(n)$ 的时间复杂度内加入一条边，时间复杂度 $O(mn) = O(kn^2)$。对于正则二分图，每种同色的边构成一组完美匹配。当 $k > O(\sqrt{n})$ 时优于前述算法。 □

## 2.2 随机化

考虑优化找匹配的复杂度，对匈牙利算法加入随机化因素，可以给出如下算法：

随机选出一个未匹配的左部点，以其为起点找一条增广路，向右走时随机选取一个相邻的点，向左走时走匹配边，直到走到一个右部未匹配点。删除这条增广路上的环，即如果走到重复点，则删除两次走这个点之间的路径，然后将增广路上每条边的匹配状态取反，即可使匹配大小加一。可以证明，除去读入和存储图的部分，该算法找到一个完美匹配的时间复杂度期望为 $O(n \log n)$

假设当前匹配了 $c$ 个未匹配点，设左右部点集为 $L, R$，匹配点集为 $L_m$，$R_m$，未匹配点集为 $L_u, R_u$，$N(x)$ 为 $x$ 相邻的点 $\forall x \in L_m \cup R_m$，记 $M(x)$ 为其匹配点。设 $f(x)$ 为从 $x$ 出发找到未匹配右部点期望经过的匹配边数，则

$$f(x)=\begin{cases}0 & x\in R_u\\ 1+f(M(x)) & x\in R_m\\ \frac{1}{k}\sum\limits_{y\in N(x)}f(y) & x\in L_u\\ \frac{1}{k-1}(-f(M(x))+\sum\limits_{y\in N(x)}f(y)) & x\in L_m\end{cases}$$

$$\forall x\in L_u, kf(x)=\sum_{y\in N(x)}f(y)$$

$$\forall x\in L_m, (k-1)f(x)=-f(M(x))+\sum_{y\in N(x)}f(y)$$

$$kf(x)=-1+\sum_{y\in N(x)}f(y)$$

$$k\sum_{x\in L}f(x)=-c+k\sum_{x\in R}f(x)$$

$$k\sum_{x\in L_m}f(x)+k\sum_{x\in L_u}f(x)=-c+k\sum_{x\in R_m}f(x)$$

$$k\sum_{x\in L_u}f(x)=(k-1)c$$

于是增广路期望经过匹配边数为 $\frac{(k-1)c}{k(n-c)}\le\frac{1}{n-c}$，路径长度不超过 $2\frac{1}{n-c}+1$，总长度 $\sum\limits_{c=0}^{n-1}(\frac{2}{n-c}+1)=O(n\log n)$，证毕。 □

使用邻接表存储边，记录每个点匹配的是哪一条出边，在删除边时将其与表中最后一个元素交换并弹出，可以做到 $O(1)$ 随机选取出边。因此找 $k$ 个匹配的时间复杂度为 $O(nk\log n)$，优于前述算法。

## 2.3 欧拉回路优化

注意到，当 $k$ 为偶数时，可以使用欧拉回路算法，将每条边按欧拉回路走过的方向定向，则每个点的入度和出度均为 $k/2$，所有方向为左部点指向右部点的边构成 $(k/2)$-正则二分图，因此可以分治递归解决。

当 $k$ 为奇数时，只需找出一组完美匹配，即可变成 $k$ 为偶数的情况，继续分治。欧拉回路部分单次时间复杂度 $O(nk)$，使用主定理分析，总时间复杂度 $O(nk\log k)$，只要在不超过 $O(nk)$ 的时间复杂度内求出一组匹配，就能实现 $O(nk\log n)$ 的求解匹配方案。下面给出 $O(nk+n\log^3 k)$ 的做法。

### 2.3.1 另一种思路

考虑给每条边赋权值为 1，则每个点相关的边权值之和为 $k$，我们希望能将每条边的权值变为 0 或 $k$，并维持每个点相关的边权值之和不变，这样所有边权 $k$ 的边形成一组完美匹配。

维持所有边的边权在 $[0,k]$ 之间，并删除所有边权为 0 或 $k$ 的边，删除边权为 $k$ 的边时加入答案。每次找到图中的一个环，将环上的边交错正负 1，我们可以让边权和较大的那一侧边权增大，较小的一侧边权减小。设环长为 $2l$，第 $i$ 条边权为 $w_i$，满足 $\sum_{i=1}^{l} w_{2i-1} \geq \sum_{i=1}^{l} w_{2i}$，记势能为 $\sum w_i^2$，则势能增加量：

$$\sum_{i=1}^{l}(w_{2i-1}+1)^2+(w_{2i}-1)^2-w_{2i-1}^2-w_{2i}^2=2l+\sum_{i=1}^{l}2(w_{2i-1}-w_{2i})\geq 2l$$

我们可以使用类似 dfs 生成树的方式找环，那么非树边一定是返祖边，每次找到一条非树边，对环上的边进行操作，然后倒退到环上深度最小的点，不影响 dfs 树性质。使用当前弧优化，每次找到一个环的复杂度为 $O(l)$，那么时间复杂度为势能增加量 $nk^2-nk=O(nk^2)$。

可以发现，当目前还没有找到匹配时，每个点的度数至少为 2，因此一定会存在环，所以任何状态总能求出方案。

使用主定理分析，总时间复杂度 $O(nk^2)$。

### 2.3.2 数据结构优化

考虑对前一个做法进行优化，首先，我们可以先只对所有边权为 1 的边找环，将所有边边权变为 $0,1,2$，其中边权为 1 的边数不超过 $n-1$，然后再只对边权为 2 的边找环，以此类推，$\forall 0\leq i\leq \lfloor \log_2 k\rfloor$，只对边权为 $2^i$ 的边找环，生成边权 $0,2^{i+1}$ 的边，那么最后会剩下 $O(n\log k)$ 条边，再继续处理。这一部分的复杂度为 $\sum_{i=0}^{\log k} n+\frac{nk}{2^i}=O(n\log k+nk)$

接下来优化更新边权的过程，首先每次边权变化的值可以取偶数边权的最小值与 $k-$ 奇数边权的最大值中的较小者，这样每次操作至少减少一条边。注意边权的变化比较简洁，我们设置阈值 $B$，用平衡树维护一些长度不超过 $B$ 的链，满足这些链点不交，并支持如下操作：

- 拼接两条链
- 从某个点处切割
- 翻转整条链
- 计算链上偶数边和奇数边的权值之和/最小权值/最大权值
- 对偶数边/奇数边加上某个值

- 删除最小/最大权值对应的边

不难发现每个操作都可以做到 $O(\log B)$。

仍然使用 dfs，与之前不同的是，我们将 dfs 过的点组织成每段长为 $B$ 的链，最后一段可能不满 $B$，每次加入一条边，如果没有访问到之前的点，就将这条边加入到最后一条链后面（或者最后一条链满了，新开一条）。如果访问到了之前的点，就统计环上的信息，进行相应的操作。确定奇数边和偶数边权总和的关系，确定能变化的最大值进行变化，删除其中边权为 0 或 $k$ 的边，对链进行分割，然后倒退到环上深度最小的点。不难理解其正确性。

加一个优化，如果新加入的点属于某条链，那么我们将这条链从这个点处分割，然后连续走较长的部分。可以证明，取 $B = k^2$，加了这个优化后这一部分的时间复杂度为 $O(n \log^3 k)$。

对于所有在 dfs 栈里的边和链，定义其势能为 0，对于每条不在 dfs 栈里也不在链里的边，定义其势能为 1，对于每条不在 dfs 栈里的链，设其长为 $h$，定义其势能为 $\log_2 h$，

每次加入一条边，势能减少 1，如果走到的点在某条链上，那么需要分割这条链，会产生一条新的不在链上的边，势能增加 1，但因为我们连续走较长的部分，所以 $\log_2 h$ 减少 1，那么总体势能仍然会减少 1。将新加入的部分并入 dfs 栈里的链复杂度不超过 $O(\log B)$，则单次时间复杂度 $O(\log B)$，每次处理一个环，会将环上的链和边分离出来，数量至多为 $\lfloor \frac{l}{B} \rfloor + 2$ 再加上减少的边数，注意 $\sum l \le nk^2$，且环个数不超过边数 $n \log k$，因为每次减少至少一条边，那么总势能增加量不超过 $O((\frac{nk^2}{B} + n \log k) \log B)$，时间复杂度 $O((\frac{nk^2}{B} + n \log k) \log^2 B)$。

取 $B = k^2$ 得 $O(n \log^3 k)$ 不超过 $O(nk)$，使用主定理，总时间复杂度 $O(nk \log k)$，达到基于欧拉回路分治的最优复杂度。

## 2.4 二分图边染色

回看边染色问题，要给每条边一个颜色，使得任意两条有公共顶点的边不同色。设二分图中点的最大度数为 $k$，借助 2.3.2，我们可以在 $O(m \log k)$ 的时间复杂度内解决。

顺次排列左部点，每次加入一个点 $u$，如果其度数与上一个点 $pre$ 的度数之和不超过 $k$，就将 $u$ 与 $pre$ 合并，即将所有 $(u, v)$ 变为 $(pre, v)$，否则不合并。不难发现，经过合并操作，相邻两个点的度数之和超过 $k$，因此左部点数不超过 $O(m/k)$。同理对右部点进行合并操作，则总点数不超过 $O(m/k)$。然后加入一些孤点使左右部点个数相等。接下来，每次找一个度数不为 $k$ 的左部点与一个度数不为 $k$ 的右部点，在他们之间连一条边，可以变为 $O(m/k)$ 的 $k$-正则图。套用 2.3.2 的做法，时间复杂度 $O(m \log k)$。

**例题 2.1** (SNOI 2024 拉丁方)**.** 称一个 $n \times n$ 的矩阵为拉丁方当且仅当其每行每列都是一个 $1 \sim n$ 的排列。给定 $n, r, c$ 和一个 $r \times c$ 的矩阵，构造一个以其为左上角的 $n \times n$ 的拉丁方或声称无解。

$1 \le r, c \le n \le 500$

考虑 $r = n$ 的情况。构造一张左右各 $n$ 个点的二分图，每个左部点表示一行，每个右部点表示一种数。每行向未在该行出现的数连一条边。若前 $c$ 列每列都是一个排列。则该图所有点度数均为 $n - r$，一定可以找到完美匹配。每次按一组匹配填入一列即可。

对于 $r < n$ 的情况。只需将前 $c$ 列填成 $c$ 个排列。构造一张左边 $c$ 个点，右边 $n$ 个点的二分图。每个左部点表示一列，每个右部点表示一种数。每列向未在该列出现的数连一条边。则每个左部点度数均为 $n - r$。如果能找到一组 $n - r$ 边染色，就能按边染色分配每一行填的数。这又等价于每个右部点的度数不超过 $n - r$。反之如果超过，则由于每行只能填入一个数，总是无法填为 $c$ 个排列。对该图运行边染色算法，变为 $r = n$ 的情况，再求解匹配即可，时间复杂度 $O(n^2 \log n)$。由于 $k = O(n)$，随机化算法与欧拉回路优化算法同复杂度。

## 2.5 $2k$-正则图

**推论 2.1.** 可以将 $2k$-正则图分解为 $k$ 个 $2$-正则图。

考虑对每个连通块运行欧拉回路算法，将每条边按欧拉回路走过的方向定向，则每个点都恰有 $k$ 条出边和入边。

建出 $2n$ 个点的二分图。对每条有向边 $(u, v)$，从左部点 $u$ 连向右部点 $v$，那么每个点的度数均为 $k$，形成了一张 $k$-正则二分图。将其分解为 $k$ 个完美匹配，对于每个完美匹配，其对应一个原图的一个 2– 正则图，那么按这 $k$ 个完美匹配分解原图即可。 □

**例题 2.2** (QOJ 10045 Permutation Recovery)**.** 有 $k$ 个长为 $n$ 的排列，将其与其逆排列写成一个 $2k \times n$ 的表格，然后打乱每一列，给你打乱后的结果，尝试还原一个表格，保证有解。

$1 \le k \le 7, 1 \le n \le 4 \times 10^4$。

考虑对于第 $i$ 列的每个数 $j$，第 $j$ 列也会有一个对应的数 $i$，在 $i$ 和 $j$ 之间连一条边，那么每个点的度数是 $2k$，要求将边集分成 $k$ 组，使得每组是一个置换环排列。则相当于将一个 $2k$-正则图分解为 $k$ 个 2-正则图。套用前述算法即可。时间复杂度 $O(nk \log k)$。 □

在本题中，图可能会出现自环或重边，但不难证明其对匹配的性质没有影响。

# 3 距离正则图

除了在匹配上的良好性质，正则图也有许多代数性质，由于所有点的度数相等，其拉普拉斯矩阵与邻接矩阵之差为 $kI$，将邻接矩阵的特征值取反后平移 $k$ 即可得到拉普拉斯矩阵的特征值。可用于计算生成树个数等。

在本章中，我们将讨论一类更特殊的正则图——距离正则图的性质。本章的图默认无重边自环。

## 3.1 定义

对于不带权的简单无向图，定义 $d(u,v)$ 为 $u,v$ 之间最短路径经过的边数，定义图的直径 $D=\max\limits_{u,v} d(u,v)$，定义 $A_i$ 为矩阵满足 $A_{i,u,v}=[d(u,v)=i]$，即所有距离为 $i$ 的点对形成的矩阵。特别地，$A_0=I$ 为单位矩阵，$A_1=A$ 为邻接矩阵。

**定义 3.1** (距离正则图)**.** 对于直径为 $D$ 的无向图，如果存在系数 $a_i,b_i,c_i$，使得 $\forall 0\le i\le D$，$AA_i=b_{i-1}A_{i-1}+a_iA_i+c_{i+1}A_{i+1}$，则称其为 **距离正则图**，

换句话说，如果对于所有距离为 $i$ 的点对 $(u,v)$，在 $v$ 的所有邻居 $w$ 中，$d(u,w)=i-1$ 的点恰有 $c_i$ 个，$d(u,w)=i$ 的点有 $a_i$ 个，$d(u,w)=i+1$ 的点恰有 $b_i$ 个，那么这个图是距离正则图。这是因为，$(AA_i)_{u,v}$ 表示从 $u$ 到 $v$ 先走一步再走 $i$ 步的方案数，枚举第一步后的距离变化，可以得到三部分贡献。

考虑 $i=0$，可以推出所有点的度数均为 $b_0$，且 $a_0=c_0=0$，于是 $\forall 0\le i\le d$，$a_i+b_i+c_i=b_0=k$，这个图是一个 $k$-正则图。

**定义 3.2** (交叉数组)**.** 定义 $\{b_0,b_1,\dots,b_{D-1},c_1,c_2,\dots,c_D\}$ 为距离正则图的 **交叉数组**。

其他系数满足 $b_D=c_0=0$，$a_i=k-b_i-c_i=b_0-b_i-c_i$ 可以由交叉数组导出。

定义 $k_i$ 表示对所有 $u$，满足 $d(u,v)=i$ 的点 $v$ 个数，可以发现 $b_i\times k_i=c_{i+1}\times k_{i+1}$，二者均表示连接了 $d(u,v)=i$ 和 $d(u,w)=i+1$ 的边 $(v,w)$ 数量。由此可以计算出 $k_i=\prod\limits_{j=1}^{i}\frac{b_{j-1}}{c_j}$，不难验证 $k_0=1$，$k_1=\frac{b_0}{c_1}=b_0=k$。

## 3.2 邻接代数

定义图 $G$ 的邻接代数为其邻接矩阵的矩阵代数 $\mathbb{A}=\mathbb{R}[A]$，由 $I,A,A^2\dots$ 张成的线性空间。对于距离正则图 $G$，我们尝试用三组基描述它：

**引理 3.1.** 设 $A$ 有 $d+1$ 个不同特征值 $\theta_0,\theta_1,\dots,\theta_d$，则 $\dim\mathbb{A}=d+1$。

由于 $A$ 是实对称矩阵，因此其可以被对角化，$A=P^{-1}\Lambda P$，其中 $\Lambda$ 为对角矩阵，且对角元为 $A$ 的特征值。则有 $A^k=P^{-1}\Lambda^kP$。对于任意多项式 $p(x)$，满足 $p(A)=P^{-1}p(\Lambda)P$，于是 $p(A)=0$ 当且仅当 $p(\Lambda)=0$。则 $A$ 的一个最小多项式为 $p(x)=\prod\limits_{i=0}^{d}(x-\theta_i)$。$\{I,A,A^2,\dots,A^d\}$ 为 $\mathbb{A}$ 的一组基，$\dim\mathbb{A}=d+1$。 □

**引理 3.2.** 令 $E_i=\prod\limits_{j=0,j\neq i}^{d}\frac{A-\theta_jI}{\theta_i-\theta_j}$，则 $\{E_0,E_1,\dots,E_d\}$ 是 $\mathbb{A}$ 的一组基。

只需证明 $E_i$ 线性无关。对于 $\theta_j$ 的任意特征向量 $v$，有 $E_iv=\delta_{i,j}v$，其中 $\delta_{i,j}=[i=j]$。证明考虑若 $j\neq i$，则 $(A-\theta_jI)v=0$，否则 $j=i$，$(A-\theta_kI)v=(\theta_i-\theta_k)v$，于是左式为 $Iv=v$。

不妨设 $\sum\limits_{i=0}^{d}c_iE_i=0$，对于任意 $i$，考虑 $\theta_i$ 的特征向量 $v$，则 $(\sum\limits_{i=0}^{d}c_iE_i)v=c_iv=0$，由于 $v\neq 0$，于是 $c_i=0$ 对于所有 $i$ 成立。$E_i$ 线性无关，$\{E_0,E_1,\dots,E_d\}$ 是 $\mathbb{A}$ 的一组基。 □

**引理 3.3.** $\{A_0 = I, A_1 = A, \ldots, A_D\}$ 是 $\mathbb{A}$ 的一组基。

首先 $\{A_0, A_1, \ldots, A_D\}$ 有值的位置互不相同，因而线性无关。

只需证明 $A^c$ 可以被 $\{A_0, A_1, \ldots, A_d\}$ 线性表出。首先 $I = A_0$，设 $A^c = \sum_{i=0}^{D} f_i A_i$，由 $AA_i = b_{i-1}A_{i-1} + a_i A_i + c_{i+1}A_{i+1}$，得 $A^{c+1} = \sum_{i=0}^{D} f_i(b_{i-1}A_{i-1} + a_i A_i + c_{i+1}A_{i+1})$ 仍为 $\{A_0, A_1, \ldots, A_d\}$ 的线性组合。$\{A_0, A_1, \ldots, A_D\}$ 是 $\mathbb{A}$ 的一组基。 □

由此可以推出 $d = D$。

## 3.3 谱

**定义 3.3** (谱)**.** 对于一个简单无向图 $G$，定义其**谱**为邻接矩阵的特征值集合（考虑重数），记作 $Spec(G) = \{\theta_0^{(m_0)}, \ldots, \theta_d^{(m_d)}\}$

定义一个图的**交叉矩阵**为：

$$L = \begin{bmatrix} a_0 & b_0 & & & & \\ c_1 & a_1 & b_1 & & 0 & \\ & c_2 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & b_{D-1} \\ & & & & c_D & a_D \end{bmatrix}$$

则其特征多项式可以 $O(D^2)$ 求出，设 $F_i$ 为其前 $i$ 行 $i$ 列子矩阵的特征多项式。满足 $F_0 = 1, F_i = (\lambda - a_{i-1})F_{i-1} - c_{i-1}b_{i-2}F_{i-2}$。

注意到其 $D+1$ 个特征值是互不相同的，由于 $b_0 \sim b_{D-1}$ 非零。因此代入任何一个特征值 $\theta$ 得到 $\theta I - L$，该矩阵前 $D$ 行总是线性无关的。秩为 $D$。我们也有简单方法计算其特征向量。

对于特征值 $\theta$，欲求特征向量 $u$ 使得 $Lu = \theta u$。设 $u_0 = 1$，有递推式 $c_i u_{i-1} + a_i u_i + b_i u_{i+1} = \theta u_i$，可以由 $u_{i-1}, u_i$ 推出 $u_{i+1}$。

**引理 3.4.** $L$ 的每个特征值都是 $A$ 的特征值。

对于任意 $L$ 的特征值 $\theta$ 和其如上定义的特征向量 $u$。任取 $G$ 中一点 $x$，令 $\alpha_i$ 表示 $A_i$ 的第 $x$ 列。有 $A\alpha_i = b_{i-1}\alpha_{i-1} + a_i\alpha_i + c_{i+1}\alpha_{i+1}$，令 $w = \sum_{i=0}^{D} u_i\alpha_i$，则 $w_i = u_{d(i,x)}$。

$$\begin{aligned} Aw &= \sum_{i=0}^{D} u_i(b_{i-1}\alpha_{i-1} + a_i\alpha_i + c_{i+1}\alpha_{i+1}) \\ &= \sum_{i=0}^{D} \alpha_i(c_i u_{i-1} + a_i u_i + b_i u_{i+1}) \\ &= \sum_{i=0}^{D} \alpha_i \theta u_i = \theta w \end{aligned}$$

□

由于 $d = D$，且 $L$ 有 $D+1$ 个不同特征值，因此 $A$ 的 $D+1$ 个不同特征值就是 $L$ 的 $D+1$ 个不同特征值。甚至可以给出每个特征值的重数。

**引理 3.5** (Biggs' 公式)**.** $A$ 的特征值 $\theta$ 重数 $m(\theta) = \frac{n}{\sum_{i=0}^{D} k_i u_i^2}$

回看 $E_i = \prod_{j=0, j\neq i}^{d} \frac{A-\theta_j I}{\theta_i-\theta_j}$，实为 $A$ 的正交投影矩阵。我们证明：$E_i^2 = E_i$，$AE_i = \theta_i E_i$

令 $f_i(x) = \prod_{j=0, j\neq i}^{d} \frac{x-\theta_j}{\theta_i-\theta_j}$，则 $f_i(\theta_j) = \delta_{i,j} = [i = j]$，且 $E_i = f_i(A)$，[1] 令 $g_i(x) = f_i^2(x) - f_i(x)$ 在所有 $\theta_j$ 处为 0，因此 $g_i(A) = 0$，$E_i^2 = E_i$。

考虑 $AE_i = \theta E_i$，重定义 $g_i(x) = xf_i(x), h_i(x) = \theta_i f_i(x)$，则 $g_i(x) - h_i(x) = 0$ 在所有 $\theta_j$ 处成立。因此 $g_i(A) - h_i(A) = 0$。$AE_i = \theta_i E_i$。

注意到 $E_i = f_i(A) = P^{-1} f_i(\Lambda) P$，又 $f_i(\theta_j) = \delta_{i,j} = [i = j]$，且相似变换不改变矩阵的迹。于是 $tr(E_i) = tr(f_i(\Lambda)) = m_i$ 为 $\theta_i$ 的重数。

由于 $E_i \in \mathbb{A}$，设 $E_i = \sum_{i=0}^{D} \nu_i A_i$，由 $AE_i = \theta E_i$ 和 $AA_i = b_{i-1}A_{i-1} + a_i A_i + c_{i+1}A_{i+1}$，且 $A_i$ 线性无关得 $c_i\nu_{i-1} + a_i\nu_i + b_i\nu_{i+1} = \theta\nu_{i+1}$，于是 $\nu$ 是 $\theta_i$ 的特征向量，满足 $\nu_i = \nu_0 u_i$。

由 $E_i^2 = E_i$，得 $\sum_{i,j} \nu_i\nu_j A_i A_j = \sum_i \nu_i A_i$，两边取迹，得 $tr(\sum_{i,j} \nu_i\nu_j A_i A_j) = n\nu_0$，当 $j \neq i$ 时 $tr(A_i A_j) = 0$，因此 $n\nu_0 = tr(\sum_i \nu_i^2 A_i^2) = \sum_i k_i \nu_i^2$，$\frac{1}{\nu_0} = \sum_i k_i u_i^2$，于是 $tr(E) = n\nu_0 = \frac{n}{\sum_i k_i u_i^2}$ □

**例题 3.1.** 使用五元环 $C_5$ 验证结论。

$C_5$ 直径为 2，$b_0 = 2, b_1 = 1$，$c_1 = 1, c_2 = 1$，$k_0 = 1, k_1 = k_2 = 2$，交叉数组 $\{2, 1, 1, 1\}$，交叉矩阵：

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

特征多项式为 $\lambda^3 - \lambda^2 - 3\lambda + 2 = (\lambda - 2)(\lambda^2 + \lambda - 1)$。$\theta_1 = 2$，利用 $c_i u_{i-1} + a_i u_i + b_i u_{i+1} = \theta u_i$，$u_0 = 1, u_1 = 1, u_2 = 1$，重数 $\frac{n}{\sum k_i u_i^2} = 1$。$\theta_2, \theta_3$ 满足 $\theta^2 + \theta - 1 = 0$，于是 $u_0 = 1, u_1 = \theta/2, u_2 = \theta^2/2 - 1 = (-\theta - 1)/2$，重数 $\frac{n}{\sum k_i u_i^2} = \frac{5}{1+2(\theta^2+\theta^2+2\theta+1)/4} = \frac{5}{1+2(1-\theta+1-\theta+2\theta+1)/4} = 2$。

邻接矩阵：

[1] 对于矩阵的多项式，需要把常数项 $c_0$ 换成 $c_0 I$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

特征多项式为 $\lambda^5 - 5\lambda^3 + 5\lambda - 2 = (\lambda - 2)(\lambda^2 + \lambda - 1)^2$，与重数相符。 □

## 3.4 邻接矩阵的特征多项式

有了上述引理，我们可以优化距离正则图邻接矩阵的特征多项式求解复杂度。

**定理 3.1.** 在模大数 $q(> n)$ 意义下，距离正则图邻接矩阵的特征多项式可在 $\tilde{O}(D^{1.5}\log q + D\log^2 q + n\log q)$ 时间复杂度内求解。其中 $\tilde{O}$ 表示忽略 $\log D$，$\log n$ 和 $\log\log q$ 因子，认为 $\mathbb{F}_q$ 下加减乘除基本运算是 $\tilde{O}(\log q)$ 的。

算法分为如下几步：

- 使用分治 NTT 在 $\tilde{O}(D\log q)$ 内求解交叉矩阵的特征多项式。
- 使用 Kaltofen–Shoup 过程在 $\tilde{O}(D^{1.5}\log q + D\log^2 q)$ 将特征多项式分解为不可约多项式的乘积。
- 对每个 $r$ 次不可约多项式，求解其在邻接矩阵上的重数。这一部分可以做到总复杂度 $\tilde{O}(D\log q)$。
- 计算每个不可约多项式幂次的乘积，先对每一项取对数，结合分治 NTT 可以做到 $\tilde{O}(D\log q)$ 变为两个 $D+1$ 次多项式的商，再 $\tilde{O}(n\log n\log q)$ 计算。

对于第一步，可以将递推式 $F_i = (\lambda - a_{i-1})F_{i-1} - c_{i-1}b_{i-2}F_{i-2}$ 写成矩阵形式，然后分治 NTT 加速。

对于第二步，参考 [3]，由于并不是本文重点，在此略过。

对于第三步，需要注意不可约多项式的所有根重数相等。

注意 $u_i$ 的递推关系，满足 $c_iu_{i-1} + a_iu_i + b_iu_{i+1} = \theta u_i$，由此可以推出 $u_{i+1} = f_{i+1}(\theta)$，考虑使用矩阵乘法维护 $(u_i^2, u_i, \sum k_iu_i^2)$，可以使用分治 NTT 加速，做到 $\tilde{O}(D\log q)$ 内将 $\sum_{i=0}^{D} k_iu_i^2$ 写为 $f(\theta)$，一个次数不超过 $2D$ 的多项式。

接下来代入 $\theta$ 是多项式的根。设多项式为 $x^k + a_{k-1}x^{k-1} + \cdots + a_0$，令 $g(x) = x^k - \sum_{i=0}^{k-1} a_ix^i$。则只需计算 $h(x) = f(x) \bmod g(x)$，然后 $f(\theta) = h(\theta)$ 即为答案。由于 $x^k + a_{k-1}x^{k-1} + \cdots + a_0$ 是不可约多项式，即是每个 $\theta$ 的最小多项式。因此 $h(x)$ 只在常数项有值，所有根的重数相等，均为 $[x^0]h(x)$。（反之违背 $\sum k_iu_i^2$ 是有理数）

问题变为:给一个 $2D$ 次的多项式 $f(x)$,$q$ 次询问 $g_i(x)$,求 $f(x) \bmod g_i(x)$,满足 $\sum \deg g_i(x) = D$，可离线。

仿照多点求值的想法，我们维护一棵关于 $g$ 的线段树，从下往上对每个节点 $l, r$ 计算 $h_{l,r}(x) = \prod_{i=l}^{r} g_i(x)$，这个就是分治 NTT 的过程。然后从上往下，计算 $f(x) \bmod h_{l,r}(x)$，再递归到两侧，递归到叶子时即为答案。由于多项式取模可以在 $\tilde{O}(len \log q)$ 时间复杂度内完成。因此这部分复杂度与分治 NTT 相同。总时间复杂度 $\tilde{O}(D \log q)$。

对于最后一步，要求计算 $\prod_{i=1}^{c} f_i(x)^{m_i}$，先取对数后变为 $\sum_{i=1}^{c} m_i \ln f_i(x)$，

再求导变为 $\sum_{i=1}^{c} \frac{m_i f_i'(x)}{f_i(x)}$，对其进行分治 NTT，维护区间的和为 $A(x)/B(x)$，则时间复杂度为 $\tilde{O}(D \log q)$。然后 $\tilde{O}(n \log q)$ 进行多项式求逆和 exp 操作。 □

上述算法的复杂度瓶颈在于多项式的因式分解，可能的优化方向是，我们不需要彻底分解，只需每个因式的所有根重数相同，即 $f(\theta)$ 总是常数。反之若不是，则其所有因子都是 $(f(x) \bmod g(x)) + c$ 的因子，可能有更好的分解方法。

## 3.5 应用

通过 3.3 给出的特征值和重数公式，我们可以在不高于特征多项式求解的复杂度内解决一些问题。由于距离正则图的直径往往远小于点数，因此这些问题在距离正则图上可以得到很大优化。

### 3.5.1 生成树计数

定义拉普拉斯矩阵 $\Delta = D - A$，由矩阵树定理，$G$ 的生成树个数为任选 $1 \le i \le n$，去掉第 $i$ 行和第 $i$ 列后的行列式。设 $\Delta$ 的特征多项式为 $f(\lambda)$，不难发现 $[\lambda^1]f(\lambda)$ 即为生成树个数的 $n$ 倍。实际上，$\Delta$ 的每一行之和为零向量，因而存在特征值 0。设 $\lambda_0 = 0$，则 $[\lambda^1]f(\lambda) = \prod_{i=1}^{n-1}(-\lambda_i)$

注意到 $\Delta$ 的特征值 $\lambda_i$ 与 $A$ 的特征值 $\theta_i$ 之间满足 $\lambda_i = b_0 - \theta_i$，仍然进行 3.4 的前三步，然后忽略分解出的多项式 $(x - k)$，对于剩下的多项式 $f_i(x)$，先变为 $g_i(x) = f_i(k - x)$，然后求 $\prod_i [x^0] g_i(x)^{m_i}$，由于只需提取零次项系数，因此不需要进行 3.4 $\tilde{O}(n \log q)$ 的多项式求逆和 exp 操作。时间复杂度瓶颈在 3.4 的第二步。

### 3.5.2 随机游走

定义图上的随机游走为从一个点 $u$ 出发，设当前在点 $x$，以 $P_{x,v}$ 的概率走到 $v$。例如 $P_{x,v} = [\exists(x,v)]\frac{1}{k}$，为每次随机走到周围的一个点。对于这种随机游走，从起点到终点期望走过的步数之和只与两点的距离有关。设 $f_i$ 为两点距离为 $i$ 还要经过的期望步数。不难发现 $f_i = 1 + \frac{1}{k}(c_i f_{i-1} + a_i f_i + b_i f_{i+1})$。由 $f_0 = 0$，可以从小往大递推，将每个 $f_i$ 写为 $k_i f_1 + b_i$，最后由 $f_D$ 与 $f_{D-1}$ 的关系式算出 $f_1$，进而算出所有 $f_i$，时间复杂度 $O(D)$。

对于更复杂的情况，例如计算 $t$ 次移动后距离的概率分布。可以转化为求解 $f \times (\frac{L}{k})^t$，使用特征多项式优化可以做到 $\tilde{O}((D^2 + D\log t)\log q)$。

## 4 总结

本文从正则图的性质出发，介绍了二分正则图匹配相比一般二分图匹配的复杂度优化。引入了距离正则图的一些性质。距离正则图还有诸多有趣的性质，在 [2] 中有介绍。笔者希望通过本文，起到抛砖引玉的作用，鼓励大家对正则图进行研究。

## 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢广东实验中学陈茂彬教练的关心和指导。
感谢家人、朋友对我的支持与鼓励。
感谢赵晟昊、张君游同学等同学与我交流、讨论。

## 参考文献

[1] Richard Cole*, Kirstin Ost, Stefan Schirra. ”Edge-Coloring Bipartite Multigraphs in $O(E \log D)$ Time.” Combinatorica, Volume 21, Issue 1 (2001), pages 5–12

[2] Edwin R. van Dam, Jack H. Koolen, Hajime Tanaka. ”Distance-regular graphs.” Electronic Journal of Combinatorics (2016)

[3] Kiran S. Kedlaya, Christopher Umans. ”Fast polynomial factorization and modular composition.” 2011

# 浅谈一类基于斜二进制的序列与树上数据结构

上海市上海中学 金怀恩

## 摘 要

本文将介绍一种可以在 $O(\log n)$ 的时间内支持单点修改区间查询，并能够在 $O(1)$ 时间内完成简单末尾追加的序列数据结构，与它的树上扩展。

## 1 记号与约定

本文研究对象为序列/有根树，且序列长度/结点数量为 $n$。在序列上，我们认为 $n$ 个位置的下标依次为 1 至 $n$，且 1 之前有位置 0。在有根树上，我们认为 $n$ 个节点的标号依次为 1 至 $n$，且根节点的“父节点”是节点 0（从而 0 是树上所有节点的祖先）。对于有根树，还有以下定义存在：

**定义 1.1** (深度)**.** 定义**深度**表示结点到根结点路径上的结点数量，下文中我们一般记作 $\mathrm{d}(x)$ 表示结点 $x$ 的深度。特别地，我们定义节点 0 的深度为 0。

**定义 1.2** (父结点)**.** 定义 $\mathrm{fa}(x)$ 表示结点 $x$ 的父结点。特别地，我们定义节点 0 的父节点同样是节点 0（即使这会违背在 1 到 $n$ 上成立的作为树的性质）。

## 2 斜二进制

斜二进制是一种进制。它从低到高第 $i \geq 1$ 位位权为 $2^i - 1$。用斜二进制表示一个数时，需要满足至多一位为 2，且低于此位者均为 0；其余位不超过 1。

**定义 2.1** (最低有效位)**.** 定义最低有效位表示一个斜二进制数从低到高最低满足值非 0 的位。斜二进制只有最低有效位上可能为 2。

**定义 2.2** (skew_lowbit)**.** 定义 $\mathrm{skew\_lowbit}(x)$ 表示斜二进制数 $x$ 最低有效位的位权。特别地，定义 $\mathrm{skew\_lowbit}(0) = 0$。

**定义 2.3** (lb)**.** 定义 $\mathrm{lb}(x)$ 表示 $x - \mathrm{skew\_lowbit}(x)$。其含义为 $x$ 在最低有效位上减 1 后的值。

**定理 2.1.** 对于正整数 $x$ 且 $\mathrm{lb}(x) \neq 0$，$\mathrm{skew\_lowbit}(x) \leq \mathrm{skew\_lowbit}(\mathrm{lb}(x))$，取等当且仅当 $x$ 的最低有效位上是 2。

证明. 根据定义，$x$ 的最低有效位上的值只能为 1 或 2。

当 $x$ 的最低有效位上值为 1 时，$\mathrm{lb}(x)$ 在这一位（以及之后的位）上均为 0，于是 $\mathrm{skew\_lowbit}(\mathrm{lb}(x))$ 大于 $\mathrm{skew\_lowbit}(x)$ 或等于 0（若 $\mathrm{lb}(x) = 0$）。从而，$\mathrm{skew\_lowbit}(x) \neq \mathrm{skew\_lowbit}(\mathrm{lb}(x))$。

当 $x$ 的最低有效位上值为 2 时，$\mathrm{lb}(x)$ 在这一位（以及之后的位）上为 1 且之后的位均为 0，于是 $\mathrm{skew\_lowbit}(x) = \mathrm{skew\_lowbit}(\mathrm{lb}(x))$。 □

**推论 2.1.** $\mathrm{skew\_lowbit}(x) = \mathrm{skew\_lowbit}(\mathrm{lb}(x))$ 当且仅当 $x = 0$ 或 $x$ 的最低有效位为 2。

**定理 2.2.**

$$\mathrm{skew\_lowbit}(x+1) = \begin{cases} 2 \times \mathrm{skew\_lowbit}(x) + 1 & if\ \mathrm{skew\_lowbit}(x) = \mathrm{skew\_lowbit}(\mathrm{lb}(x)) \\ 1 & otherwise \end{cases}$$

证明. $\mathrm{skew\_lowbit}(x) = \mathrm{skew\_lowbit}(\mathrm{lb}(x))$ 当且仅当 $x$ 的最低有效位上的值为 2 或 $x$ 为 0。此时对 $x$ 加上 1 触发进位，令原最低有效位的上一位加上 1，而这一位归 0，于是新的最低有效位变为上一位。否则，对 $x$ 加上 1 会导致最低位增加 1 并不触发进位，新的最低有效位变为最低位。 □

**定理 2.3.**

$$\mathrm{lb}(x+1) = \begin{cases} \mathrm{lb}(\mathrm{lb}(x)) & if\ x - \mathrm{lb}(x) = \mathrm{lb}(x) - \mathrm{lb}(\mathrm{lb}(x)) \\ x & otherwise \end{cases}$$

证明. $x - \mathrm{lb}(x) = \mathrm{lb}(x) - \mathrm{lb}(\mathrm{lb}(x))$ 等价于 $\mathrm{skew\_lowbit}(x) = \mathrm{skew\_lowbit}(\mathrm{lb}(x))$，此时

$$\begin{aligned} \mathrm{lb}(x+1) &= (x+1) - \mathrm{skew\_lowbit}(x+1) \\ &= x - 2 \times \mathrm{skew\_lowbit}(x) \\ &= \mathrm{lb}(x) - \mathrm{skew\_lowbit}(\mathrm{lb}(x)) \\ &= \mathrm{lb}(\mathrm{lb}(x)) \end{aligned}$$

否则 $\mathrm{lb}(x+1) = (x+1) - \mathrm{skew\_lowbit}(x+1) = x$。 □

# 3 序列结构

## 3.1 预处理和更新

对于位置 $x$，考虑维护 $(\mathrm{lb}(x), x]$ 的区间。

**定理 3.1.** 所有这样的区间两两包含或不交。

证明. 若 $\mathrm{lb}(x) < y < x$，则 $y$ 的最低有效位一定低于 $x$ 的最低有效位（因为在此后一定有非 0 位置）。从而，$\mathrm{lb}(y)$ 一直到 $x$ 的最低有效位（含）之前都与 $\mathrm{lb}(x)$ 一致，从而 $\mathrm{lb}(x) < \mathrm{lb}(y)$，即 $(\mathrm{lb}(y), y] \subset (\mathrm{lb}(x), x]$，得证。 □

从而可以将所有 $(\mathrm{lb}(x), x]$ 的区间按树形结构组织。我们认为节点 $x$ 管辖 $(\mathrm{lb}(x), x]$。

**定理 3.2.** 对于节点 $x$ 与其祖先 $y$，$y$ 的管辖区间长度大于 $x$ 管辖区间长度的两倍。

证明. 节点 $x$ 和 $y$ 的管辖区间长度分别为 $\mathrm{skew\_lowbit}(x)$ 和 $\mathrm{skew\_lowbit}(y)$。由于 $y$ 是 $x$ 的祖先，有 $\mathrm{skew\_lowbit}(y) > \mathrm{skew\_lowbit}(x)$；而 skew_lowbit 的取值一定是斜二进制位权，相邻项满足 $\times 2 + 1$ 的递推关系，从而 $\mathrm{skew\_lowbit}(y) \geq 2 \times \mathrm{skew\_lowbit}(x) + 1$。 □

**推论 3.1.** 树的高度不超过 $O(\log n)$。

证明. 从叶子节点往上走，每走一步都会让管辖区间长度至少翻倍，而每个节点的管辖区间长度都不超过 $n$，因此至多翻 $O(\log n)$ 倍。 □

**定理 3.3.** 对于非叶节点 $x + 1$，其恰有两个儿子，分别是 $x$ 和 $\mathrm{lb}(x)$。

证明. 由于 $\mathrm{lb}(x + 1) = \mathrm{lb}(\mathrm{lb}(x))$，可知 $x$ 和 $\mathrm{lb}(x)$ 都是 $x + 1$ 的后代。此外，$(\mathrm{lb}(\mathrm{lb}(x)), \mathrm{lb}(x)] \cup (\mathrm{lb}(x), x] = (\mathrm{lb}(x + 1), x]$，与 $(\mathrm{lb}(x + 1), x + 1]$ 仅相差 $x + 1$ 的单点，而 $x$ 与 $\mathrm{lb}(x)$ 父亲的管辖区间大小一定不小于 $\mathrm{skew\_lowbit}(x + 1)$，从而 $x + 1$ 有且仅有 $x$ 和 $\mathrm{lb}(x)$ 两个儿子。 □

**推论 3.2.** $1, 2, \cdots, n$ 是结构的一个拓扑序，儿子永远出现在父亲前。

考虑在序列 $a(1), a(2), \cdots, a(n)$ 上维护信息，满足节点 $x$ 上维护的信息为其管辖区间内所有元素的合并（即 $\circ_{i=\mathrm{lb}(x)+1}^{x} a(i)$），记为 tr(x)。此外，记 $\mathrm{tf}(x)$ 为节点 $x$ 的父亲（不存在即记为 0）。

**定理 3.4.**

$$\mathrm{tr}(x+1) = \begin{cases} \mathrm{tr}(\mathrm{lb}(x)) \circ \mathrm{tr}(x) \circ a(x+1) & if\ x - \mathrm{lb}(x) = \mathrm{lb}(x) - \mathrm{lb}(\mathrm{lb}(x)) \wedge x \neq 0 \\ a(x) & otherwise \end{cases}$$

证明. 若 $x + 1$ 为非叶节点，则

$$(\mathrm{lb}(x + 1), x] = (\mathrm{lb}(\mathrm{lb}(x)), \mathrm{lb}(x)] \cup (\mathrm{lb}(x), x] \cup \{x + 1\}$$

整理即得上述关系式。 □

**定理 3.5.** 存在对于给定序列使用 $O(n)$ 预处理时间建立结构（包括 lb, tr, tf 三个数组）的方法，且能支持在 $O(1)$ 的时间复杂度内向序列末尾追加元素并维护结构。

证明. 由于 $1, 2, \cdots, n$ 是结构的拓扑序，依次进行递推即可。由于已知 $x+1$ 唯二的儿子（若有）是 $x$ 和 $\mathrm{lb}(x)$，可以在这一过程中维护 tf。追加元素时，其一定维护拓扑序的最后，只需要建立这一单点，不会影响即有的 lb 和 tr，只会影响至多两个位置的 tf（这些位置原本的 tf 应当为 0）。 □

**定理 3.6.** 当改变某一特定 $a(x)$ 的值时，存在使用 $O(\log n)$ 时间维护结构（tr 数组）的方法。

证明. 由于结构的高度是 $O(\log n)$ 的，且记录了每个节点的父亲，只需要从节点 $x$ 开始遍历所有祖先，并按照从低到高的顺序重新运行递推即可。 □

## 3.2 查询

**定理 3.7.** 给定区间 $(l, r]$，存在算法可以在 $O(\log n)$ 的时间复杂度内通过信息合并计算得到 $\circ_{i=l+1}^{r} a(i)$。

证明. 考虑如下算法。

**Algorithm 1:** Query in $O(\log n)$ time

**Input:** Query bound $l$, $r$

**Output:** $s_{l+1} \circ s_{l+2} \circ \cdots \circ s_r$

```
Function query(l, r)
    res ← e
    while r > l do
        if lb(r) < l then
            res ← a(r) ∘ res
            r ← r − 1
        else
            res ← tr(r) ∘ res
            r ← lb(r)
    end
    return res
```

**定义 3.1** (秩高)**.** 定义**秩高** $h(x) = \log_2(\mathrm{skew\_lowbit}(x) + 1)$，则 $h(x) = O(\log n)$。特别地，我们定义 $h(0) = \lfloor \log_2(n) \rfloor + 1$。

**定理 3.8.** $h(\mathrm{lb}(x)) \geq h(x)$

证明. 由于 $\mathrm{skew\_lowbit}(\mathrm{lb}(x)) \geq \mathrm{skew\_lowbit}(x)$ 或 $\mathrm{lb}(x) = 0$，这一结论是自然的。 □

**定理 3.9.** 对于正整数 $x$，$h(\mathrm{lb}(\mathrm{lb}(x))) \geq h(x) + 1$。

证明. $h(\mathrm{lb}(\mathrm{lb}(x))) \geq h(\mathrm{lb}(x)) \geq h(x)$。由于 $x$ 和 $\mathrm{lb}(x)$ 不可能同时最低有效位为 2，且 $h(0) \geq h(x)+1$，这一结论是自然的。 □

**推论 3.3.** 所示算法在第一次未执行 else 部分前，复杂度是 $O(\log n)$ 的。

证明. 每执行两次 $r \leftarrow \mathrm{lb}(r)$，都会使 $h(r)$ 至少加 1；而 $h(r)$ 有上界 $O(\log n)$，从而得以证明。 □

以下考虑 $r$ 开始执行第一个分支后的部分。

**定理 3.10.** 若 $r$ 某一次执行第一个分支时 $h(r) = x$，那么此后都会满足 $h(r) < x$。

证明. $r$ 前最后一个秩高不低于 $x$ 的位置是 $\mathrm{lb}(r)$，由于在 $r$ 进入了第一个分支，$r$ 不可能以 $\mathrm{lb}(r)$ 的值再次进入循环体。 □

**定理 3.11.** $r$ 第一次执行第一个分支后，不再可能连续两次执行第二个分支。

证明. $r$ 执行第一个分支会使得秩高恰好减 1（退位）。而至多两次执行第二个分支会将 $r$ 的最低有效位归 0，使得秩高加 1，导致 $h(r)$ 的秩高与之前执行第一个分支时一致，矛盾 □

**推论 3.4.** 所示算法在第一次未执行 else 部分后，复杂度是 $O(\log n)$ 的。

**定理 3.12.** 所示算法的复杂度是 $O(\log n)$ 的。

□

**定理 3.13.** 给定 $r$ 和函数 ok，满足存在 $l$ 满足 $ok(i) = [i \leq l]$，那么存在算法可以在 $O(\log n)$ 的时间复杂度内找到 $l$。

证明. 考虑如下算法。

**Algorithm 2:** Query in $O(\log n)$ time

**Input:** Query bound $r$,Function ok

**Output:** Maximum $l$ such that $\mathrm{ok}(l) = 1$

```
Function query(r)
    while ok(r) = 0 do
        if ok(lb(r)) = 0 then
            r ← lb(r)
        else
            r ← r − 1
    end
    return r
```

复杂度证明是类似的。 □

### 3.3 小结

在信息学竞赛中，如上所示算法可以在 $O(n)$ 时间内完成结构建立，$O(1)$ 时间内完成末尾追加，$O(\log n)$ 时间内完成单点修改和区间查询。这一算法相较（递归）线段树代码较短，但空间占用更大（需要记录额外的 lb 和 tr 数组）。因此，这一算法上在序列上主要作为一种可以考虑的备选。

## 4 树上结构

考查如下结构：对于每个点，维护一条由其到某个祖先（不含）的链；设当前点深度为 $i$，则该祖先的深度为 $\mathrm{lb}(i)$。设当前点为 $x$，则记这个祖先为 $\mathrm{lb}(x)$（原 lb 的概念不再被使用）。

**定理 4.1.** 若 $\mathrm{d}(x) = \mathrm{d}(y)$，则 $\mathrm{d}(\mathrm{lb}(x)) = \mathrm{d}(\mathrm{lb}(y))$

这是由定义带来的自然结果，$\mathrm{d}(\mathrm{lb}(x))$ 由 $\mathrm{d}(x)$ 唯一确定。

**定理 4.2.** 所有这些链构成一个有向无环图的结构。对于节点 $x$，记 $p = \mathrm{fa}(x), q = \mathrm{lb}(p), r = \mathrm{lb}(q)$，则节点 $x$ 直接连向节点 $p$ 和节点 $q$。因此，原树的拓扑序（父亲在先）同样也是结构的拓扑序。

**定理 4.3.** 结构上最长有向路径的长度是 $O(\log n)$ 的。

**定理 4.4.** 对于节点 $x$，记 $p = \mathrm{fa}(x), q = \mathrm{lb}(p), r = \mathrm{lb}(q)$，则。

$$\mathrm{lb}(x) = \begin{cases} q & if\ \mathrm{d}(p) - \mathrm{d}(q) = \mathrm{d}(q) - \mathrm{d}(r) \\ p & otherwise \end{cases}$$

对于每个节点 $x$，我们同样记 $x$ 上的元素为 $a(x)$，那么额外维护 $\mathrm{tr}(x)$ 表示 $a(x) \circ a(\mathrm{fa}(x)) \circ \cdots$，一直到 $a(\mathrm{lb}(x))$（不含）。

**定理 4.5.** 对于节点 $x$，记 $p = \mathrm{fa}(x), q = \mathrm{lb}(p), r = \mathrm{lb}(q)$，则。

$$\mathrm{tr}(x) = \begin{cases} a(x) \circ \mathrm{tr}(p) \circ \mathrm{tr}(q) & if\ \mathrm{d}(p) - \mathrm{d}(q) = \mathrm{d}(q) - \mathrm{d}(r) \wedge p \neq 0 \\ a(x) & otherwise \end{cases}$$

**定理 4.6.** 结构（包含 lb 数组和 tr 数组）可以在 $O(n)$ 的时间内被建立；与此同时，给定建立完毕的结构，可以在 $O(1)$ 的时间内向任意一个节点添加一个儿子而维护结构。

**定理 4.7.** 给定一条祖先到后代的链 $x \leftarrow y$，存在算法可以在 $O(\log n)$ 的时间复杂度内通过信息合并计算得到 $a(x) \circ a(\mathrm{fa}(x)) \circ \cdots$，一直到 $a(y)$（不含）。

**定理 4.8.** 给定 $x$ 和函数 ok，满足存在 $x$ 的祖先 $y$ 满足 $ok(i) = [d(i) \le d(y)]$（要求 $i$ 是 $x$ 的祖先），那么存在算法可以在 $O(\log n)$ 的时间复杂度内找到 $y$。

以上诸条，仅考虑深度，与序列上的情况是完全对应的。

**定理 4.9.** 给定节点 $u$ 和 $v$，则存在算法可以在 $O(\log n)$ 的时间复杂度内利用上述结构求出 $u$ 和 $v$ 的最近公共祖先。

证明. 考虑如下算法。

**Algorithm 3:** Finding LCA in $O(\log n)$ time

**Input:** Tree nodes $u$ and $v$

**Output:** LCA of $u$ and $v$

```
Function lca(u, v)
    if d(u) < d(v) then
        Swap u, v
    while d(u) > d(v) do
        if d(lb(u)) > d(v) then
            u ← lb(u)
        else
            u ← fa(u)
    end
    while u ≠ v do
        if lb(u) ≠ lb(v) then
            u ← lb(u)
            v ← lb(v)
        else
            u ← fa(u)
            v ← fa(v)
    end
    return u
```

它的正确性是自然的，而复杂度根据以上定理也得到保证。 □

## 4.1 小结

在信息学竞赛中，如上所示算法可以在 $O(n)$ 时间内完成结构建立，$O(1)$ 时间内完成叶子追加，$O(\log n)$ 时间内完成路径刻画。这一算法相较线段树套树剖具有显著代码长度优势，相较传统倍增算法则具有显著预处理时空优势，是处理相关问题的优秀选择。

# 5 应用

**例题 5.1** (城池攻占[1])**.** 小铭铭最近获得了一副新的桌游，游戏中需要用 $m$ 个骑士攻占 $n$ 个城池。

这 $n$ 个城池用 1 到 $n$ 的整数表示。除 1 号城池外，城池 $i$ 会受到另一座城池 $f_i$ 的管辖，其中 $f_i < i$。也就是说，所有城池构成了一棵有根树。

这 $m$ 个骑士用 1 到 $m$ 的整数表示，其中第 $i$ 个骑士的初始战斗力为 $s_i$，第一个攻击的城池为 $c_i$。

每个城池有一个防御值 $h_i$，如果一个骑士的战斗力大于等于城池的生命值，那么骑士就可以占领这座城池；否则占领失败，骑士将在这座城池牺牲。占领一个城池以后，骑士的战斗力将发生变化，然后继续攻击管辖这座城池的城池，直到占领 1 号城池，或牺牲为止。

除 1 号城池外，每个城池 $i$ 会给出一个战斗力变化参数 $(a_i, v_i)$。若 $a_i = 0$，攻占城池 $i$ 以后骑士战斗力会增加 $v_i$；若 $a_i = 1$，攻占城池 $i$ 以后，战斗力会乘以 $v_i$。

注意每个骑士是单独计算的。也就是说一个骑士攻击一座城池，不管结果如何，均不会影响其他骑士攻击这座城池的结果。

现在的问题是，对于每个城池，输出有多少个骑士在这里牺牲；对于每个骑士，输出他攻占的城池数量。数据范围：$1 \le n, m \le 3 \times 10^5$，$-10^{18} \le h_i, v_i, s_i \le 10^{18}$，$1 \le f_i < i, 1 \le c_i \le n, a_i \in \{0, 1\}$，保证 $a_i = 1$ 时，$v_i > 0$，保证任何时候骑士战斗力值的绝对值不超过 $10^{18}$。

## 5.1 基本思路

解法. 由于攻占城池对骑士战力的影响可以作为一次函数快速复合，我们在每个节点和结构维护的链上维护，该段城市对骑士血量的影响，以及合法的骑士血量范围。每次从给定节点一路向上倍增即可。对单个骑士可以以 $O(\log n)$ 的复杂度在线查询。相较于传统的可并堆解，本解法支持在线；相较于 $k$ 进制倍增解（本题空间限制不支持传统倍增通过），本解法更加自然，理论复杂度也更优；相较于树剖线段树解，本解法代码量显著短。 ♠

**例题 5.2** (小 U 的树[2])**.** 给定 $n$ 个点，点有权，初始时两两均未连边。

你需要支持 $m$ 次两种操作之一：

*1.* 在两个不联通的点间连上一条边；

*2.* 求两个给定点构成路径上所有点权的最大子段和（不能为空）。

强制在线。数据随机生成。

数据范围：$1 \le n, m \le 3 \times 10^6$。

[1]来源：JLOI2015https://www.luogu.com.cn/problem/P3261

[2]来源：原创https://www.luogu.com.cn/problem/P14302

### 5.2 基本思路

解法. 由于结构可以支持 $O(1)$ 时间动态加叶子，我们使用启发式合并维护联通块，即可在均摊 $O(\log n)$ 的时间内完成 1 操作，并在严格 $O(\log n)$ 的时间内完成 2 操作。虽然查询常数较大，但由于这一操作的 log 在深度上而随机生成的树深度较小，综合常数可以接受。♠

## 6 总结

本文介绍了一种基于斜二进制的序列/树上数据结构，并通过两道例题展现了其（在树上）的应用。本文所述算法和结构由笔者独自原创。经考据，Codeforces 上曾更早出现相似内容，但并没有得到良好的推广。

希望本文可以起到抛砖引玉的作用，让读者了解到斜二进制结构在序列/静态树上的应用。期待有更多相似的数据结构在信息学竞赛中出现并得到推广。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练任舍予的指导。

感谢上海市上海中学毛黎莉、钱涛等老师的关心与指导。

感谢家人、朋友对我的支持与鼓励。

# 浅谈信息学竞赛中的一些数形结合方法

长沙市长郡中学 李秉霈

## 摘 要

数形结合作为一种广泛应用的数学思想，旨在利用图形的直观性来启发思路并简化推导。本文探讨了数形结合方法在冒泡分析、格路计数以及嵌入优化等问题的具体应用与优势，为部分传统问题提供了更贴合直觉的新视角，同时研究了其他类别问题通过数形结合思想可启发的直觉。

## 1 前言

在算法竞赛中，很多算法思想和问题的性质挖掘需要灵感与启迪。因此在不同视角下看待问题，难度亦有分别。如果能够找到合适的视角研究问题，会起到事半功倍的效果。

其中，几何直觉在严谨的数学表达之外，可以对结构的理解带来直接的启发。在数学中许多领域都有采用数形结合思想的例子，如使用拓扑学中的 Borsuk-Ulam 定理解决项链分割问题 [1]、把含根号不等式问题转化为平面上动点距离问题来解决等。

受此启发，笔者发现如果对部分算法竞赛问题使用数形结合的方法、从几何的视角看待，往往会有助于启发直觉，有时甚至能借用几何中凸性一类的工具直接解决问题。

笔者把作用场合类似的方法归为一类，归纳出了冒泡分析、格路计数、嵌入优化三类，以及其他一些较为特殊、未能归类的方法。

## 2 冒泡分析问题

冒泡排序是一种非常经典的排序算法，最早由 Edward Harry Friend 在 1956 年提出 [2]。其工作流程为：不断扫描整个序列并交换相邻逆序元素，直到整个序列变得有序。

在很多题目中，都有类似的交换相邻逆序元素操作，部分题目直接出现了冒泡排序过程中序列形态的查询。笔者将这些问题概括为冒泡分析问题。

### 2.1 右上格路径

为了介绍冒泡分析问题中通用的数形结合方法，我们首先需要引入右上格路径的概念。

**定义 1** (01 序列对应的右上格路径)**.** 设 01 序列 $a = (a_1, a_2, \cdots, a_n) \in \{0,1\}^n$。定义一条平面折线路径 $P$，其起点为 $(0,0)$，并依次对每个 $i = 1, 2, \cdots, n$，按如下规则移动：$(x, y) \mapsto (x + 1 - a_i, y + a_i)$。最终得到的 $P$ 即称为 $a$ 对应的右上格路径。

通俗地说，01 序列对应的右上格路径就是从原点开始，根据“0 右 1 上”规则走出的路径。下图是一个简单的例子：

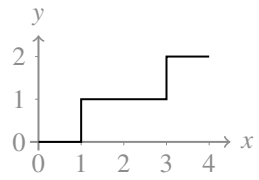



图 1: 序列 $(0,1,0,0,1,0)$ 对应的右上格路径

由上述定义，易知所有的 01 序列和所有的右上格路径之间构成双射。这为从右上格路径视角研究 01 序列问题提供了可能。接下来笔者将先阐述右上格路径是如何直接应用在 01 序列的冒泡分析问题中的，再讨论如何将其扩展至序列值域更一般的情况。

## 2.2 右上格路径的直接应用

为了研究右上格路径的应用，我们首先需要说明冒泡排序的基本操作对右上格路径的影响。在 01 序列中，交换相邻逆序元素即为把相邻的 $(1,0)$ 交换为 $(0,1)$。下图以 $(0,1,0,0,1,0)$ 交换为 $(0,0,1,0,1,0)$ 为例，展示了交换前后，对应右上格路径的局部形变。

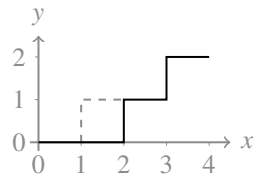



图 2: 序列 $(0,1,0,0,1,0)$ 交换为 $(0,0,1,0,1,0)$，右上格路径的局部形变

容易发现，$(1,0)$ 所对应的上-右局部被交换为了 $(0,1)$ 对应的右-上。形象地说，就如同把一个角按进去了。这表明在右上格路径视角下，冒泡分析问题有较好的几何直观意义。下面的例题展示了这样的直观如何对解题思路产生启发。

**例题 1** (Chorus[1])**.** 给定 $n, k$ 和一个包含 $n$ 个 A 与 $n$ 个 B 的字符串 $S$。求最少需要交换相邻元素多少次，能够使得 $S$ 可以划分为 $k$ 个子序列，每个子序列 A 与 B 数量相等且所有 A 在所有 B 前面。数据范围：$k \le n \le 10^6$。

[1]**题目来源**：Japanese Olympiad in Informatics Spring Training 2023. Day 3. (https://qoj.ac/problem/6338)。

首先考虑研究如何判定一个字符串 $S$ 是否是合法的。

注意到当 $k < n$ 时，如果其能划分为 $k$ 个满足要求的子序列，那么也一定能划分为 $k+1$ 个。因此判定也就是要最小化划分出的子序列数量。通过调整法容易得到如下的贪心：不断把字符串开头连续的 A 和前相同数量个 B 划分为一个子序列。如果任意时刻都没有出现字符串开头为 B 的情况，而且最终划分出的子序列数量 $\le k$，那么 $S$ 就是合法的。

得到的判定还是过于复杂，考虑从右上格路径的视角看待这个贪心划分的过程。在这里我们把 A 视作 0，B 视作 1，也就是“A 右 B 上”。第一步能取的 A 的数量等于字符串开头连续的 A，也就是右上格路径初始连续向右的步数。接着取相同数量的 B，也就是从刚才走到的位置不断向上走直到碰到直线 $y = x$。

因此，原先的划分过程在右上格路径上等价于：从 $(0,0)$ 开始，在不越过路径的情况下尽量向右，然后再向上直到碰到 $y = x$。下图是一个示例：

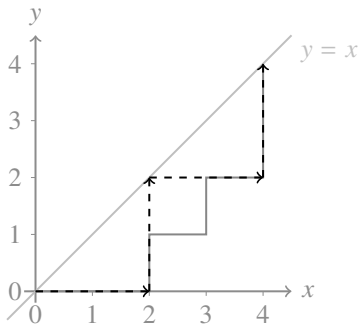



图 3: 序列 AABABABB 的判定过程，实线为右上格路径，虚线为判定轨迹

定义 $k$-标准路径为重复 $k$ 次“向右若干单位，向上相同长度”得到的右上格路径，即如同图中虚线这样的右上格路径。可知，$S$ 是合法的当且仅当可以作出一条 $k$-标准路径，使得 $S$ 对应的右上格路径完全位于它的右下方。

因此，假如已经固定了作出的 $k$-标准路径，根据之前的分析，交换相邻逆序元素会让 $S$ 对应的右上格路径有一个角被按进去，故此时的最少交换次数即为 $S$ 初始对应的右上格路径越过该标准路径的总面积。

令 $S$ 初始对应的右上格路径在 $x = i - 1$ 处的最高点为 $y_i$。设该 $k$-标准路径的所有右上拐点横坐标为 $(x_1, x_2, \cdots, x_k)$，特殊地，令 $x_0 = 0$。那么此时越过的总面积即为

$$\sum_{i=1}^{k} \sum_{j=x_{i-1}+1}^{x_i} \max(y_j - x_{i-1}, 0)$$

可以使用动态规划来求解所有 $k$-标准路径的最少次数。设 $f_{i,j}$ 表示在已经决策了 $x_{1\sim i}$，其中 $x_i = j$ 的情况下，可能的最小总贡献是多少。令 $w(\ell, r) = \sum_{i=\ell+1}^{r} \max(y_i - \ell, 0)$，转移即为 $f_{i,j} + w(j,p) \to f_{i+1,p}$。

观察可得 $w(\ell, r)$ 满足四边形不等式，因此答案关于 $k$ 具有凸性。可以通过二分切线斜率（在算法竞赛中通常称为 WQS 二分或 Alien's Trick）的方法转化为没有 $k$ 的限制的情况。这部分由于与本文关系不大，在此略去。具体证明与算法介绍可见参考文献 [3]、[4]。

现在 $w(\ell, r) = m + \sum_{i=\ell+1}^{r} \max(y_i - \ell, 0)$，其中 $m$ 为二分的切线斜率。由于式子中 max 的存在，该 DP 仍然不是很方便优化。

考虑变换一下形式，提前预处理出 $z_i = \max(i, \max j \text{ s.t. } y_j \le i)$，那么有 $w(\ell, r) = m + \sum_{i=z_\ell+1}^{r} (y_i - \ell)$。预处理 $y_i$ 的前缀和 $s_i$ 后即

$$w(\ell, r) = m + s_r - s_{z_\ell} - \ell \cdot (r - z_\ell)$$

因此可以使用经典的斜率优化技巧解决。单次求解 DP 时间复杂度 $O(n)$，总时间复杂度 $O(n \log n)$。

## 2.3 更广泛的情况

上面已经讨论了一些使用右上格路径方法解决 01 序列上的冒泡分析问题的例子。而对于值域不局限于 $\{0, 1\}$ 的冒泡分析问题，常用的方法是，枚举值域分界线 $v$，把原先的一般序列 $a$ 转为 01 序列 $b_i = [a_i \ge v]$ 来分析，并尝试通过每个 $v$ 的信息还原原始答案。

常见的还原手段有两种：在统计问题中，通常会使用 $x = \sum_{v \ge 1} [x \ge v]$ 类似的等式来还原；在求解排序网络结束时间的问题中，通常使用 0-1 原理 [5] 的一个推论：一个序列被某排序网络排序的结束时间，等于其在所有阈值二值化后所得 01 序列结束时间的最大值。

以下为求解排序网络结束时间的一个例子。

**例题 2** (Bubble Sort 2[2])**.** 维护序列 $a = (a_1, a_2, \cdots, a_n)$。进行 $q$ 次单点修改，每次修改之后查询：假如对序列运行冒泡排序算法，在序列排好序之前会扫描序列多少轮。

数据范围：$n, q \le 5 \times 10^5$。

考虑如何求序列被冒泡排序的轮数。根据上面的结论，对于每个 $v$ 求出 $a'_i = [a_i \ge v]$ 的答案，这些答案的最大值即为原问题答案。

在 01 序列中，冒泡一轮相当于：对于每个 1，把它移动到它后面紧邻的 0 连续段后。从右上格路径的视角也就是说，把路径中 $y > 0$ 的部分整体向下平移了一单位，并在末尾补充了一条向上一单位长度的线段。如下图：

从而可得，01 序列被冒泡排序的轮数，即为其初始对应的右上格路径中，横向线段的 $y$ 轴高度最大值（若无横向线段，则轮数为 0）。

枚举该线段对应的原序列下标 $i$，那么其高度即为 $a'_{[1,i)}$ 中 1 的个数。要选取一个 $v$ 使得 $a'_i = 0$ 且线段高度尽可能大，易知应取 $v = a_i + 1$。因此答案为

$$\max_{i=1}^{n} \sum_{j=1}^{i-1} [a_j > a_i]$$

[2]**题目来源**：JOI Open Contest 2018 (https://qoj.ac/problem/2643)。

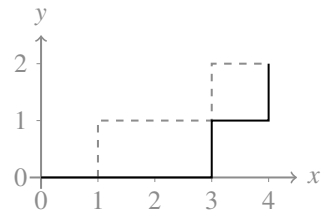



图 4: 序列 $(0,1,0,0,1,0)$ 冒泡一轮变为 $(0,0,0,1,0,1)$

注意到横向线段的高度最大值一定在最右一条取到，即对应的位置为序列中的最后一个 0。因此答案也一定在 $a_i$ 为后缀严格最小值时取到。注意到此时 $\sum_{j=1}^{i-1}[a_j > a_i] = \sum_{j=1}^{n}[a_j > a_i] - (n-i)$，而对于其他 $a_i$，左式一定大于右式，因此答案也可以改写为

$$\max_{i=1}^{n}\left(\sum_{j=1}^{n}[a_j > a_i] - (n-i)\right)$$

可以离散化后直接使用权值线段树维护答案。时间复杂度为 $O((n+q)\log(n+q))$。

# 3 格路计数问题

格路计数问题一般指在平面上施加一些限制后，求两点间格路径数量的问题。在求解格路计数问题的过程中，在两类格路径集合之间建立某种对应关系通常是一类较为有力的观察。而使用数形结合的方法，我们将可能通过对格路径简单直观的几何变换来启发思路，并找到对应关系。

## 3.1 建立双射

一个直接的应用是：通过可逆的几何变换，在需要统计的格路径集合和另一个集合间建立双射，从而改变计数的对象。

**例题 3** (Japanese Knowledge[3])**.** 给定一个单调不减数组 $(a_1, a_2, \cdots, a_n)$，对每个 $0 \le k \le n$，求满足如下条件的单调不减数组 $(x_1, x_2, \cdots, x_n)$ 的数量，对 998244353 取模：

- 对任意 $1 \le i \le n$，都有 $x_i \le a_i$。
- 满足 $x_i = a_i$ 的位置 $i$ 的数量恰好为 $k$。

数据范围：$1 \le n, a_i \le 2.5 \times 10^5$。

[3]**题目来源**：XXI Open Cup named after E.V. Pankratiev, Grand Prix of Tokyo (https://qoj.ac/problem/3091)。

考虑把 $x$ 视为一条有 $n$ 条横向单位线段的右上格路径，其中第 $i$ 条线段的高度为 $x_i$，并对 $a$ 也做同样的处理。那么问题转化为：给定一条右上格路径，对每个 $0 \le k \le n$，求有多少条右上格路径满足：整体位于给定路径的下方，且有恰好 $k$ 条横向线段与给定路径重合。

从一些特殊的例子开始研究：$k = 0$ 的情形是经典的“阶梯型格路计数”，可以使用分治 NTT 解决：取出给定格路径的中点，把阶梯分为左下、右上两个小阶梯和一个矩形。递归求解两个小阶梯，并使用 NTT 加速计算矩形部分。具体细节由于与本文关系不大，不在此处赘述，可见参考文献 [6]。

当 $k = 1$ 时，考虑对 $k = 1$ 对应的路径做如下变换：

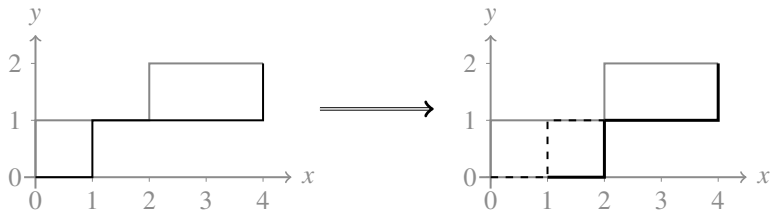



图 5: $a = (1, 1, 2, 2), x = (0, 1, 1, 1)$ 时的平移示例

即，把路径在重合的横向线段处以前的部分向右平移一单位。容易发现该变换是可逆的：找到最左的接触给定路径拐点的位置，并平移回去即可。因此，所有 $k = 1$ 的格路径与所有从 $(0, 1)$ 出发的 $k = 0$ 格路径之间是一一对应的。

容易发现该变换也可以推广至 $k > 1$ 的情况：从左至右依次找到重合处并平移即可。因此对于任意 $k$，所有该类格路径与所有从 $(0, k)$ 出发的 $k = 0$ 格路径之间形成双射。

故我们只需要统计从各 $(0, k)$ 出发的“阶梯型格路计数”，同样可以使用上面提到的分治 NTT 方法解决。时间复杂度 $O((n + V) \log^2(n + V))$。

## 3.2 构造反射

在格路计数问题中，常见的一种限制形式是格路不允许跨越某种边界。然而存在一类被称为反射的方法，其以特定权值计算其他与计数对象存在对应关系的格路径类，通过正负抵消，构造出天然不可跨越的边界，从而解决该限制。

与上一节类似的，我们仍然可以通过具有对称性等特殊性质的几何变换，构造出与其他格路径类的正负抵消。

**例题 4** (Chess Rush 简化版[4])**.** 有一个 $R \times C$ 棋盘。$Q$ 次查询，每次给定 $A$ 与 $B$，求一个国际象棋中的国王棋子从 $(1, A)$ 出发到达 $(R, B)$ 有多少条不同的最短路径。答案对 $10^9 + 7$ 取模。

[4]**题目来源**：Central European Olympiad in Informatics 2020 Day 2，有改编。

数据范围：$C, Q \le 10^3$，$C \le R \le 10^9$。

由于 $C \le R$，最短路径长度显然为 $R-1$。因此问题转化为：从 $(1, A)$ 出发，到达 $(R, B)$，每步可以 $(x, y) \mapsto (x+1, y \pm 1)$ 或 $(x, y) \mapsto (x+1, y)$，不能超出棋盘边界，求合法路径数。

考虑动态规划，设 $f_{i,j}$ 表示走到 $(i, j)$ 的方案数，那么转移形如 $f_{i,j} \to f_{i+1,j-1}, f_{i+1,j}, f_{i+1,j+1}$，但需要注意的是需要每次手动把 $f_{i,0}$ 与 $f_{i,C+1}$ 重设为 0。

如果没有重设为 0 这一步，直接计算 $[x^{B-A}](1+x+\frac{1}{x})^{R-1}$ 即可。因此难点在于 0 与 $C+1$ 处的边界不好处理。考虑构造反射来解决，对棋盘做如下的变换：

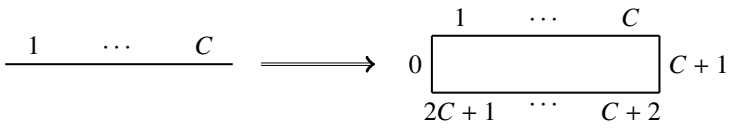



图 6: 对棋盘（第二维坐标）的变换

此时，如果在与 $A$ 对称的位置 $2C+2-A$ 放置一个权值为 $-1$ 的起点，那么归纳可得任意 $f_{i,0}$ 与 $f_{i,C+1}$ 都一定为 0，不需要手动重设。

因此只需要预处理 $(1+x+\frac{1}{x})^{R-1} \bmod (x^{2C+2}-1)$，即可每次 $O(1)$ 回答。使用快速幂与暴力卷积，时间复杂度 $O(C^2 \log R + Q)$。使用多项式算法优化则可以做到 $O(C \log C \log R + Q)$。

# 4 嵌入优化问题

组合优化问题指的是在某类候选解集合中寻找最优解的问题。而在组合优化问题中有一类特殊的问题，其候选解集合中的元素都可以抽象为二维向量，此时可以考虑一种数形结合的方法：把候选解集合当作平面中的一些点，通过几何直观尝试解决。这种方法被称为二维嵌入，而笔者把这类问题概括为嵌入优化问题。

## 4.1 排除非最优解

二维嵌入后，我们往往能够通过一些平面上的与权值函数相符的性质，快速排除非最优解元素，起到优化的目的。下面是两种较为经典的方法。

### 4.1.1 直接求出凸壳

如果可以说明权值函数 $F(x, y)$ 的最优解一定会取在候选解集合中 $(x, y)$ 的凸壳上，那么我们就仅需要考虑凸壳上的元素。事实上很多常见的 $F(x, y)$ 都满足该条件，比如 $F(x, y) = x \cdot y$。因此在一些 $|x|$ 与 $|y|$ 都不会很大的问题中，我们可以考虑直接求出候选解的凸壳。

**例题 5** (timeismoney[5])**.** 给定一张 $n$ 个点 $m$ 条边的无向连通图，每条边有两个权值 $(a_i, b_i)$。求该图的一棵生成树，最小化 $(\sum a_i) \times (\sum b_i)$。

数据范围：$n \le 200$，$m \le 10^4$，$0 \le a_i, b_i \le 255$。

把每种生成树方案视为平面上的一个点 $(\sum a_i, \sum b_i)$，易知只需要保留左下凸壳的方案。

我们容易直接使用最小生成树算法求出最左点与最下点（分别对应 $\sum a_i$ 与 $\sum b_i$ 最小的方案），它们即为左下凸壳的两端点，分别设为 $A$ 和 $B$。此时使用直线 $AB$ 切该凸壳，设切点为 $C$，那么如果 $C$ 与 $A$ 或 $B$ 重合，则说明该凸壳恰为线段 $AB$；否则可以递归至 $(A, C), (C, B)$ 两个子问题。

我们有如下结论：

**引理 1.** 设有点集 $\{(a_i, b_i)\}_{i=1}^n$，其中 $a_i, b_i \in \mathbb{Z}$，且 $\max_i\{|a_i|, |b_i|\} \le v$。令 $\mathcal{H}$ 为其下凸壳，则该凸壳上的顶点数量 $K$ 满足：

$$K = O\left(v^{2/3}\right)$$

该引理的详细证明可见 [7]。

由引理1，如果可以 $O(C)$ 求出某直线切该左下凸壳的切点，那么整个问题就可以在 $O(C(nv)^{2/3})$ 的时间复杂度内解决。

设直线斜率为 $k$，那么求其切该凸壳的切点，也就是要最小化截距 $y - kx = \sum b_i - k\sum a_i = \sum(b_i - ka_i)$。这相当于是把边权设为 $b_i - ka_i$ 后求解最小生成树问题，可以 $O(m \log m)$ 解决。因此原问题可以做到 $O((nv)^{2/3} m \log m)$。

事实上，上面直接求出凸壳的算法十分具有扩展性。在大部分情况下，求某直线切凸壳的切点都是比原问题要简单的，此时如果方案嵌入后对应的点均为整点，且范围不大，则使用该算法将大幅简化问题。

### 4.1.2 二维偏序调整

在将最优化问题的局部二维嵌入后，我们常常能够得到形如二维偏序的最优性调整。而这种形如二维偏序的调整的性质实际上是很好的：此时最优解的候选集合，也就是这个调整下的极优解，在平面上会有额外的性质，由此我们可以得到对该集合的一个直观刻画。

**例题 6** (Price Combo[6])**.** 有 $n$ 个物品需要购买，第 $i$ 个物品在商店 $A$ 的价格为 $a_i$，在商店 $B$ 的价格为 $b_i$。每次可以在任意一个商店购买两个物品，花费是这两个物品的价格最大值；或原价购买一个物品。求买到每个物品的最小总花费。数据范围：$n \le 2 \times 10^5$。

考虑把每个物品当作平面上的点 $(a_i, b_i)$。那么有如下的调整法：如果 $a_i \ge a_j \wedge b_i \le b_j$，且 $i$ 是在 A 买的、$j$ 是在 B 买的，那么可以调整成 $i$ 在 B 买、$j$ 在 A 买，一定不劣。

[5]**题目来源**：Balkan Olympiad in Informatics 2011 Day 2 (https://qoj.ac/problem/290)。

[6]**题目来源**：The 3rd Universal Cup. Stage 7: Warsaw (https://qoj.ac/problem/9222)。

那么也就是说：对于每一个在 A 购买的点，其左上方的所有点也都是在 A 购买的。因此对于任意一个方案，所有在 A 购买的点会形成一条形如右上格路径的边界，要求只有这个边界下方的点才能在 B 购买。

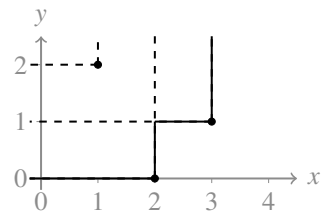



图 7: 形成的边界示例

也就是说，我们只需要统计满足如下条件的方案：

- 存在一条从原点出发的右上格路径，其左上全为 A，右下全为 B。

可以考虑动态规划：设 $f_{i,j,k}$ 表示当前路径从最右上开始走到了 $(a_i, b_i)$，在商店 A 和 B 分别的购买数量奇偶性为 $j, k$，最小总花费是多少。写出该动态规划的转移式后，可以使用数据结构方法加速转移。这部分由于与本文关系不大，在此不再赘述。

## 4.2 观察并利用整体性质

在另外的一些情况中，候选解集合本身就具有一些可以加速计算或便利维护的性质（比如凸性），而通过嵌入我们将更容易发现这些性质。注意这一部分并不要求一定是二维嵌入，事实上更高维的嵌入也是可行的，但需要对公式代表的几何意义更加熟悉，或借助图形计算器等手段使其直观。

**例题 7** (Grand Finale: Circles[7])**.** 给定平面上的 $n$ 个圆，求一个半径最大的圆被这 $n$ 个圆的交覆盖。保证这些圆的交非空。

数据范围：$n \le 10^5$，圆心坐标为 $[-10^6, 10^6]$ 内的整数，半径为 $[1, 2\times 10^6]$ 内的整数，对答案要求的精度误差为 $10^{-7}$。

假设给定的圆分别为 $(x-a_i)^2 + (y-b_i)^2 = c_i^2$。考虑如何判定圆心在 $(x, y)$，半径为 $r$ 的圆是否合法。那么其需要满足 $0 \le r \le c_i$ 且 $(a_i - x)^2 + (b_i - y)^2 \le (c_i - r)^2$。

使用图形计算器等方法容易观察到，这等价于要求点 $(x, y, r)$ 在某个圆锥内。那么最终合法的 $(x, y, r)$ 构成的集合就是一些圆锥的交，也就是一个三维凸集。现在我们要求解的即为该凸集的最高点。

[7]**题目来源**：Codeforces Round 930 (Div. 1) (https://codeforces.com/problemset/problem/1936/F)。

由三维凸集的性质可知，设固定 $x$ 的情况下，最大可以取到的 $r$ 为 $f(x)$，那么 $f$ 是凸函数。考虑对 $f$ 三分，那么只需要求 $O(\log \frac{V}{\varepsilon})$ 次 $f$ 的点值。

而求 $f$ 的点值是容易的：固定 $x$ 后，所有合法的 $(y,r)$ 构成二维凸集，因此最大可以取到的 $r$ 可以再通过一次三分求出。因此只需要求 $O(\log^2 \frac{V}{\varepsilon})$ 次固定 $(x,y)$ 情况下最大的 $r$。这可以直接用上面判定部分的式子计算，单次求解时间复杂度 $O(n)$，总时间复杂度 $O(n \log^2 \frac{V}{\varepsilon})$。

# 5 其他技巧

## 5.1 观察输出

如果把题目中研究的对象转为了平面上的几何元素，那么除了直接感受或推导，也可以生成一些随机数据，并输出这些数据下的所需研究的几何元素，来观察可能存在的性质。

**例题 8** (Candy Piles[8])**.** 初始桌面上有 $n$ 堆石子，每堆的石子个数分别为 $(a_1, a_2, \cdots, a_n)$。两名玩家轮流执步，每步取走石子个数最多的一堆，或在每堆取走一颗石子。

取走最后一颗石子的玩家判负。请判断谁必胜。

数据范围：$n \le 10^5$，$1 \le a_i \le 10^9$。

我们将 $a_i$ 降序排序，画出一个由方格组成的图形，第 $i$ 列有 $a_i$ 个方格，且所有列下对齐（如下所示）。

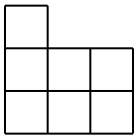


图 8: $a = (3,2,2)$ 对应的图形

在该图形的视角下，容易发现两种决策分别为去掉最下行和去掉最左列。从而，每个时刻的局面可以被当前图形的左下角在原图形中的位置唯一描述。

因此原问题等价于这样的博弈问题：从原点开始，在该图形上，两人轮流选择向右一单位或向上一单位，移动到边界上的人判负。考虑输出每个点出发的必胜态来观察性质：

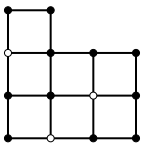


图 9: $a = (3,2,2)$ 对应的每个点必胜态，实心/空心圆圈分别表示先/后手必胜

[8]**题目来源**：AtCoder Grand Contest 002 (https://atcoder.jp/contests/agc002/tasks/agc002_e)。

经过对各种 $a$ 的观察，可以得到如下结论：如果 $(x+1,y+1)$ 不是边界上的点，那么 $(x,y)$ 的必胜态和 $(x+1,y+1)$ 一致。实际上该结论可简单归纳证明，可见原题解 [8]。

从而，求出最大的 $(i,i)$ 满足其不在边界上，那么只需要求出 $(i,i)$ 向右与向上分别离边界的距离，即可判断其是先手必胜还是后手必胜。时间复杂度 $O(n\log n)$。

## 5.2 Ferrers 图

**定义 2** (Ferrers 图)**.** 给定一个整数 $n$ 的分拆 $\lambda=(\lambda_1,\lambda_2,\ldots,\lambda_k)$，其中 $\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_k>0$ 且 $\sum\lambda_i=n$。*Ferrers* 图就是由 $k$ 行方格组成的图形，第 $i$ 行有 $\lambda_i$ 个方格，且所有行左对齐。

Ferrers 图常用于对整数分拆问题的数形结合研究。

**例题 9** (逆序对[9])**.** 给定 $n,k$，求有多少个 $n$ 阶排列逆序对数为 $k$。答案对 $10^9+7$ 取模。

数据范围：$n,k\leq 10^5$。

首先，排列的每个位置 $i$ 与其前面的 $i-1$ 个位置对总逆序对数的贡献在 $[0,i)$ 之间，且各位置之间贡献互相独立，因此问题也即：求有多少种 $x_1+x_2+\cdots+x_n=k$ 的方案，其中 $0\leq x_i<i$。对 $x_i<i$ 的条件容斥，那么答案为

$$\sum_{S\subseteq[n]}(-1)^{|S|}\cdot\binom{k-\sum_{i\in S}i+n-1}{n-1}$$

因此，只需要对每个 $[0,k]$ 的数 $x$，求出

$$\sum_{(\sum_{i\in S}i)=x}(-1)^{|S|}$$

即可。直接使用 01 背包，复杂度无法接受。考虑把方案用 Ferrers 图表示：

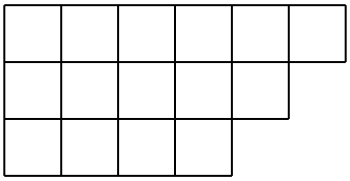


图 10: 选择 $\{4,5,6\}$ 对应的 Ferrers 图

传统的 01 背包方法在这个图形上的意义即为按行加入。但在该问题中，由于选的数互不相同，因此行数是 $O(\sqrt{k})$ 的。故可以考虑按列加入：设 $f_{i,j}$ 表示 $i$ 行面积为 $j$ 的 Ferrers 图的权值之和。每次在左侧加入一列，或在左侧加入一列并在下面添加一行。注意需要扣除第一行长度 $>n$ 的情况。转移即为 $f_{i,j}=f_{i,j-i}-f_{i-1,j-i}+f_{i-1,j-n-1}$。

根据之前的分析，第一维大小是 $O(\sqrt{k})$ 的。因此总时间复杂度 $O(k\sqrt{k})$。

[9]**题目来源**：2017 山东一轮集训 Day7 (https://loj.ac/p/6077)。

## 6 总结

本文首先引入了右上格路径这一平面结构，并介绍了冒泡分析问题中基于此结构的数形结合技巧；接下来介绍了在格路计数问题中，通过几何变换构造双射、反射来完成计数的例子；然后介绍了嵌入优化问题中，排除非最优解的通用方法，以及观察问题解集整体性质的例子；最后介绍了观察输出、Ferrers 图这样未能归类的方法。

本文介绍的诸多数形结合方法相比于这些问题的传统方法，有不需要记忆繁复结论、推导起来更加直观简易的优势，其中的很多方法也能应用到其他问题中，具有较好的拓展性。笔者希望这些方法能够在读者尝试解决类似问题时提供思路或启发灵感。

数形结合的方法浩如烟海，限于笔者水平和本文篇幅，难以详尽介绍。希望本文可以起到抛砖引玉的作用，也希望未来能够出现更多关于数形结合的题目、方法或理论总结。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练任舍予的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，谢秋锋老师的教导和同学们的帮助。

感谢时庆钰、刘承奥与我讨论本文内容，并提供写作上的指导。

感谢程楷轩、周雨衡、孙铁铮、郝心悦同学为本文审稿并提出诸多宝贵建议。

## 参考文献

[1] Noga Alon and Douglas B West. The borsuk-ulam theorem and bisection of necklaces. Proceedings of the American Mathematical Society, 98(4):623–628, 1986.

[2] Edward H Friend. Sorting on electronic computer systems. Journal of the ACM (JACM), 3(3):134–168, 1956.

[3] Wolfgang W Bein, Lawrence L Larmore, and James K Park. The d-edge shortest-path problem for a monge graph. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1992.

[4] 王钦石. 浅析一类二分方法. IOI2012 中国国家集训队第二次作业（自选部分）, 2012.

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms. MIT press, 2022.

[6] 吴畅. 再谈格路计数. 2024 年国际信息学奥林匹克竞赛中国国家集训队论文集, 2024.

[7] George E Andrews. A lower bound for the volume of strictly convex bodies with many boundary lattice points. Transactions of the American Mathematical Society, 106(2):270–279, 1963.

[8] AtCoder. Atcoder grand contest 002 editorial. https://img.atcoder.jp/data/agc/002/editorial.pdf, 2016. Accessed: 2026-01-08.

# 浅谈近似算法及其应用

乐山高新区嘉祥外国语学校 黎莫轩

## 摘 要

在现代的算法领域，近似算法占据了重要的地位。如今，传统算法、随机化算法等在信息学竞赛中已经有了许多应用。但受限于信息学竞赛考察的形式，近似算法并没有在其中得到普及。本文介绍了一系列经典的采用了贪心、随机化、线性规划等各种方法的近似算法，并介绍了部分算法优化的困难性。

## 1 近似算法初探

### 1.1 引入

在信息学竞赛的学习中，我们早已知晓存在一系列的问题，在目前被认为是难以在多项式复杂度内解决的。例如：旅行商问题、3-SAT 问题、背包问题等。

然而，在现实与理论研究之中，这些问题又有广泛的应用。因此，我们不得不放宽要求，在可接受的时间内求解一个并不完全精确的近似解。这就是我们要讨论的问题。

**定义 1.1** (近似比)**.** 令最优解为 $OPT$，且某个近似算法给出的解为 $SOL$。
若在最大化问题中，满足 $\alpha OPT \le SOL \le OPT$，则称算法的近似比为 $\alpha$。
若在最小化问题中，满足 $OPT \le SOL \le \alpha OPT$，则称算法的近似比为 $\alpha$。
在上述情况中，我们称这个算法是 $\alpha$-近似的。

接下来，我们考虑一些具体的问题。[14]

### 1.2 负载均衡问题

**问题 1.1** (负载均衡问题)**.** 给定非负数列 $\{t_1, t_2, \cdots, t_n\}$，求一组把 $S = \{1, 2, \cdots, n\}$ 划分成 $S = T_1 \cup T_2 \cup \cdots T_m$ 的方案，满足对任意 $1 \le i \le m$，$\sum\limits_{j \in T_i} t_j$ 的最大值最小。

我们知道，上述问题的判定版本（即，给定 $K$，判定问题的答案是否 $\le K$）是 NPC 的，换句话说，如果我们承认 P ≠ NP，那么这个问题不存在多项式复杂度的求精确解算法。接下来，让我们考虑，我们能在多项式的时间内让近似比到达多少。

为了方便起见，我们下面称 $\sum_{j\in T_i} t_j$ 为“集合 $T_i$ 的总和”。

### 1.2.1 算法 1

一个显然的想法是贪心。我们不妨按照任意顺序枚举所有 $t_i$，每次维护所有 $m$ 个集合的当前总和，把 $t_i$ 加入总和最小的集合。

那么，这个算法的近似比是多少呢？

**定理 1.1.** 这是一个 2-近似算法。

证明. 设最优解为 $M = \mathrm{OPT}$。假设在某一刻，某个集合的总和超过了 $2M$，同时这次操作加入的物品为 $w$，则必须有 $w \le M$，即加入这个物品前该集合的总和已经超过了 $M$。然而，我们所选取的是总和最小的集合，这代表所有集合的总和都超过了 $M$，与 $M$ 为最优解矛盾。 □

我们可以说明 $\alpha = 2$ 对这个算法是紧的，也就是说，不存在 $\alpha' < 2$ 是该算法的近似比。如下构造数据：我们有 $m(m-1)$ 个 1 和 1 个 $m$，显然这组数据的最优解为 $m$。然而，上述的贪心算法会将最优解算成 $2m-1$。取足够大的 $m$，则 $\frac{2m-1}{m} \to 2$。

### 1.2.2 算法 2

由于上述算法在“先放很多个小物品，再放一个大物品”这组数据上表现较差，一个想法是：我们把所有 $t_i$ 从大到小排序再运行原先的算法。这是否给出了更优的近似比呢？

**定理 1.2.** 这是一个 $\frac{3}{2}$-近似算法。

证明. 设最优解为 $M = \mathrm{OPT}$。假设在某一刻，某个集合的总和超过了 $\frac{3M}{2}$，假设这次操作加入的物品为 $w$。若 $w \le \frac{M}{2}$，则可以推出操作前所有集合大小都至少为 $M$，这是矛盾的。因此 $\frac{M}{2} < w \le M$。

由于我们把所有 $t$ 从大到小排序，先前加入的所有 $t$ 均 $> \frac{M}{2}$。显然加入当前的 $w$ 时所有集合非空，因此，存在至少 $(m+1)$ 个 $> \frac{M}{2}$ 的元素，从而由鸽笼原理得到不存在使得所有集合大小 $\le M$ 的方案，矛盾。 □

### 1.2.3 算法 3

如果尝试构造，会发现我们难以把上述算法近似比卡到 $\frac{3}{2}$ 的界。这提示我们，这个界可能并不是紧的。

接下来，我们证明更优的近似比。

**定理 1.3.** 这是一个 $\frac{4}{3}$-近似算法。

证明. 设最优解为 $M = \mathrm{OPT}$。类似前面算法的证明，我们只需要考虑 $t_i > \frac{M}{3}$ 的元素，剩下的元素必定不会使得所求解超出 $\frac{4}{3}M$。

这样的元素，在每个集合里最多有两个，因此总共至多有 $2m$ 个这样的元素。这实际上直接变成了两两匹配使得总和最小的问题。对于这样的问题，我们有经典的结论：最大匹配最小、次大匹配次小，以此类推即可。可以发现，我们的算法刚好正确地给出了这样的解，因此它对 $> \frac{M}{3}$ 的元素给出的解必不可能超过 $M$，从而最终是一个 $\frac{4}{3}$-近似。 □

## 1.3 集合覆盖问题

**问题 1.2** (集合覆盖问题)**.** 令 $U = \{1, 2, \cdots, n\}$，给出若干 $S_i, C_i$，其中 $S_i$ 为 $U$ 的子集，$C_i$ 为非负数。选出其中的一些，使得选出的 $S_i$ 的并集为 $U$，且 $C_i$ 之和最小。

### 1.3.1 算法

我们仍然给出一个贪心算法。令 $X \subseteq U$ 为当前未被覆盖的元素。每次在 $X \cap S_i \neq \varnothing$ 的集合中，选择 $\frac{C_i}{|X \cap S_i|}$ 最小的集合，然后把覆盖的元素从 $X$ 里删去。不断重复这个过程直到 $X = \varnothing$ 为止。这时我们求得了一组方案。

### 1.3.2 分析

设 $H_n = \sum_{i=1}^{n} \frac{1}{i}$。

**定理 1.4.** 这是一个 $H_n$-近似算法。

证明. 令 $M = \mathrm{OPT}$。

考虑某个时刻的情况，我们证明存在一个 $S_i$ 使得 $\frac{C_i}{|S_i \cap X|} \leq \frac{M}{|X|}$。

这是因为：考虑最优解中选择的和 $X$ 有交的所有集合，显然它们的并集包含 $X$，从而与 $X$ 的交集包含的元素个数之和至少为 $|X|$。假设它们都满足 $\frac{C_i}{|S_i \cap X|} > \frac{M}{|X|}$，则它们的价值之和大于 $\frac{M}{|X|} \sum |S_i \cap X| \geq M$。这就导出了矛盾，因此一定能找到 $\frac{C_i}{|S_i \cap X|} \leq \frac{M}{|X|}$ 的集合。

记第 $i$ 次选择了 $S_{t_i}$，在这时 $x_i = |S_{t_i} \cap X|$，$n_i = |X|$。则最终求得的答案不超过 $M \sum_i \frac{x_i}{n_i}$，显然 $x_i = 1$ 时取最值 $MH_n$，从而这个算法是一个 $H_n$-近似算法。 □

## 1.4 K 中心问题

在描述这个问题前，我们先给出度量空间的定义。

**定义 1.2** (度量空间)**.** 对于集合 $M$，定义函数 $d : M \times M \to \mathbb{R}$。满足对任意 $x, y, z \in M$：

- $d(x, y) = 0 \Leftrightarrow x = y$

- $d(x,y) = d(y,x)$
- $d(x,z) \le d(x,y) + d(y,z)$。

则称 $d$ 为 $M$ 上的度量，称 $(M,d)$ 为度量空间。

形象化地说，可以把度量空间看作满足三角形不等式的无向完全图，$d(x,y)$ 代表 $x,y$ 之间的边权。

**问题 1.3** (K 中心问题)**.** 在一个度量空间 $(V,d)$ 内，选择一个大小为 $k$ 的集合 $S \subseteq V$，最小化 $\max\limits_{u\in V}\min\limits_{v\in S} d(u,v)$。

### 1.4.1 算法

首先，我们发现可能的答案只有 $O(|V|^2)$ 种，因此枚举或二分答案 $r$ 后，转化为判定性问题。下面，我们称一个点集 $S \subseteq V$ 的 $r$-邻域为 $\{v \in V \mid \min\limits_{u\in S} d(u,v) \le r\}$。

设 $S = V$，$C = \varnothing$。每次从 $S$ 任选一个点 $u$ 加入 $C$，然后从 $S$ 内删去所有 $d(u,v) \le 2r$ 的点 $v$。若最终 $|C| \le k$ 则认为成功。对于所有成功的方案，选取最小的 $2r$ 作为最终的答案。

### 1.4.2 分析

**定理 1.5.** 这是一个 2-近似算法。

证明. 当 $r = \mathrm{OPT}$ 时，假设最优解的点集为 $C'$，并初始化集合 $W = \varnothing$。运行上述算法时，我们假设每当向 $C$ 加入一个元素 $u$ 时，都从 $C'$ 中删去距离它最近的元素 $v$ 加入集合 $W$。我们归纳说明：$C$ 内元素的 $2r$-邻域一定包含 $W$ 内元素的 $r$-邻域，且 $d(u,v) \le r$。从而说明至多执行 $k$ 次这个过程后，所有点都会被从 $S$ 删除。

首先，假设 $d(u,v) > r$，则 $C'$ 内当前所有元素距离 $u$ 都 $> r$。且由于 $v$ 当前未被删去，一定有 $C$ 内元素的 $2r$-邻域不含 $u$，于是 $W$ 内元素的 $r$-邻域也不含 $u$。这说明初始的 $C'$ 内就没有距离 $u$ 小于等于 $r$ 的点，与 $C'$ 最优矛盾。

因此，$d(u,v) \le r$，从而由三角不等式，若 $d(v,w) \le r$，一定有 $d(u,w) \le d(u,v)+d(v,w) \le 2r$。这就说明了 $C$ 内元素的 $2r$-邻域一定包含 $W$ 内元素的 $r$-邻域。

而 $S$ 内 $k$ 个点的 $r$-邻域已经包含了整个 $V$，从而 $C$ 内至多有 $k$ 个元素。因此，$r = \mathrm{OPT}$ 时，一定能构造出不超过 $k$ 个点的方案。从而这是一个 2-近似。 □

## 1.5 旅行商问题（TSP）

### 1.5.1 一般图上的 TSP

**问题 1.4** (TSP)**.** 给出完全图 $(V,E)$，记 $u,v$ 之间的边权为 $c(u,v)$。

求解 $p_0, p_1, \cdots, p_{|V|-1}$，其中 $\forall i \neq j$，$p_i \neq p_j$，且 $p_i \in V$。

最小化：

$$\sum_{i=0}^{|V|-1} c(p_i, p_{(i+1) \bmod |V|})$$

**定理 1.6.** 对于上述问题，只要 $P \neq NP$，就不存在任何多项式时间内的 $2^{poly(n)}$-近似算法。

证明. 对于一张任意的无向无权图 $G$。构造一张新的完全图 $G' = (V, E')$。满足：

$$c(e) = \begin{cases} 1 & e \in E \\ 2^{\text{poly}(n)} n & e \notin E \end{cases}$$

显然新的 $G'$ 的规模仍然是 $n$ 的多项式，而任何不经过 $E$ 内的边的方案的总代价为 $n$，任何经过 $E$ 内的边的代价大于 $n2^{\text{poly}(n)}$，这意味着，如果存在 $2^{\text{poly}(n)}$-近似，则我们可以判定 $G$ 是否存在哈密顿回路。然而哈密顿回路是 NP-Hard 的！这就导出了矛盾，所以原问题没有多项式时间的 $2^{\text{poly}(n)}$-近似。 □

### 1.5.2 度量 TSP

虽然一般图上的 TSP 问题过于困难，但某些特殊条件下，我们仍然能够得到一些 TSP 问题的近似算法：

**问题 1.5** (度量 TSP)**.** 在一个度量空间 $(V, d)$ 内，求解 $p_0, p_1, \cdots, p_{|V|-1}$，其中 $\forall i \neq j, p_i \neq p_j$，且 $p_i \in V$。

最小化：

$$\sum_{i=0}^{|V|-1} d(p_i, p_{(i+1) \bmod |V|})$$

为了方便起见，我们直接把度量空间 $(V, d)$ 描述成无向完全图 $(V, E)$，$E$ 的边权满足三角不等式。

### 1.5.3 算法 1

求解原图的任意一棵最小生成树（MST），然后输出它的任意一个深搜序。

**定理 1.7.** 这是一个 2-近似算法。

证明. 显然，给出的方案不劣于沿 MST 的树边模拟一遍 DFS 过程所经过的边权和。即 MST 边权和的 2 倍。

而 TSP 方案删去任意一条边就可以得到一棵生成树，从而 MST 权值小于等于 TSP 的答案。

因此，这个算法是 2-近似的。 □

值得一提的是，这里的 2 这一近似比是紧的，考虑如下构造：

- $V = \{0, 1, \cdots, 2n\}$。
- 0 和任何点的距离为 1。
- 对于 $0 \le i < 2n$，$i$ 和 $i+1$ 的距离为 1。
- 其余点对的距离为 2。

运行上述算法，可能求得的 MST 为以 0 为中心的菊花图（即，0 的度数为 $|V|-1$），从而一个可能的深搜序为 $0, 1, 3, 5, \cdots, 2n-1, 2, 4, 6, \cdots, 2n$。

此时，它的代价为 $4n$，而最优解则是 $0, 1, 2, \cdots, 2n$，代价为 $2n+1$。取足够大的 $n$，就有 $\frac{4n}{2n+1} \to 2$。

### 1.5.4 算法 2

实际上，上述算法相当于把 MST 加倍后求解了一条欧拉回路。然而，如果只是为了求解欧拉回路，我们并不总需要把 MST 加倍，因此我们得到如下算法：

先求解原图的任意 MST，再找到所有度数为奇数的点，显然这样的点有偶数个。那么，我们求这样奇点的最小权完美匹配，再将它们按照匹配相连，就可以使得所有点的度数为偶数，从而得到欧拉回路。对欧拉回路去除重复经过的点即可得到哈密顿回路。

其中，最小权匹配可以使用带权带花树算法在多项式时间内解决，从而整个算法的时间复杂度是多项式的，这被称为 Christofides 算法。

**定理 1.8.** 这是一个 $\frac{3}{2}$-近似算法。

证明. 考虑所有的奇点，取出最终回路 $C$ 的这些奇点组成的子序列，则可以得到一个连接这些奇点的环，且环长 $\le$ OPT。

我们把环划分成两组不交的完美匹配，则由鸽笼原理，至少有一组 $\le \frac{\text{OPT}}{2}$。

而 MST 的权值 $\le$ OPT。故二者总和 $\le \frac{3\text{OPT}}{2}$。从而这一算法是 $\frac{3}{2}$-近似的。 □

$\frac{3}{2}$ 这一近似比也是紧的。考虑构造如下完全图：

- $V = \{0, 1, \cdots, 2n\}$。

- 对于 $0 \le u \le v \le 2n$，$d(u,v) = \left\lceil \frac{v-u}{2} \right\rceil$。

显然最优解为 $2n+1$。取 MST 为 $0, 1, \cdots, 2n$ 这条链，则奇点只有 $0, 2n$，它们的距离为 $n$。从而 MST 和匹配的总权值为 $3n$。取足够大的 $n$，有 $\frac{3n}{2n+1} \to \frac{3}{2}$。

## 1.6 斯坦纳树问题

**问题 1.6** (斯坦纳树问题)**.** 给定带权完全图 $(V, E)$ 和点集 $R \subseteq V$，寻找原图的树子图，使得它包含 $R$ 中的所有点，且边权和最小。

显然，如果边权不满足三角不等式，则我们可以用 $(u, v)$ 的最短路替代连接 $(u, v)$ 的边权，从而转化为边权满足三角不等式的问题。因此，不妨直接假定它满足三角不等式。

### 1.6.1 算法

考虑 $R$ 的导出子图，直接求解它的 MST 作为答案。

### 1.6.2 分析

**定理 1.9.** 这是一个 2-近似算法。

证明. 假设最优解为 $M$，则与我们对 TSP 问题的证明类似地，存在恰好包含 $R$ 内所有点的，边权和不超过 $2M$ 的回路。这说明 $R$ 内所有点的 MST 边权和不超过 $2M$。从而我们的算法是一个 2-近似算法。 □

近似比 2 是紧的。考虑构造如下完全图：

- $V = \{0, 1, \cdots, n\}$。
- $R = V - \{0\}$。
- 0 和任何点的距离为 1。
- 其余任何点对的距离为 2。

显然最优解为以 0 为中心的菊花图，边权和为 $n$。然而上述算法求解出的答案为 $2(n-1)$。取 $n$ 足够大，就有 $\frac{2(n-1)}{n} \to 2$。

## 1.7 最大 3-SAT 问题

首先，我们给出一些基础的定义。

**定义 1.3** (文字)**.** 文字是一个布尔变量，或其否定。我们把变量 $x_i$ 的否定记作 $\neg x_i$。

**定义 1.4** (子句)**.** 子句是若干文字通过逻辑或连接得到的逻辑表达式。其中，逻辑或的符号为 $\vee$。例如，$x_1 \vee \neg x_2 \vee x_3$ 是一个包含 3 个文字的子句。

若逻辑表达式为真，即组成子句的文字至少有一个为真，那么我们称这个子句被满足。

**定义 1.5** (合取范式)**.** 合取范式（*CNF*）是多个子句通过逻辑与连接得到的逻辑表达式。其中，逻辑与的符号为 $\wedge$。例如，$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_4 \vee \neg x_5)$ 是一个合取范式。

对于标准的 3-SAT 问题，我们需要对于一个所有子句都恰好包含 3 个文字的 CNF，求解它的所有子句能否同时被满足。而我们现在要处理的则是它的最大化版本：

**问题 1.7** (最大 3-SAT 问题)**.** 给出一个所有子句都恰好包含 3 个文字的 *CNF*，寻找一组对所有变量的赋值方案，使得被满足的子句数量最大。

### 1.7.1 算法

考虑对所有变量等概率独立随机赋值。对于一个子句，它被满足的几率为 $1 - \left(\frac{1}{2}\right)^3 = \frac{7}{8}$。因此，我们期望满足 $\frac{7}{8}$ 的子句。这也说明：存在至少一组满足 $\geq \frac{7}{8}$ 子句的方案。

现在的问题是如何找到这组方案。考虑依次确定每个变量的值。当我们枚举到一个变量时，分别检验它为 $0, 1$ 时，剩下变量随机取值满足子句数量的期望。显然，两种方案的至少一种仍然期望满足 $\geq \frac{7}{8}$ 的子句。重复这一过程，我们得到了一个 $\frac{7}{8}$-近似的多项式复杂度算法。

## 1.8 背包问题

**问题 1.8** (背包问题)**.** 给定 $n$ 组数对 $(v_i, w_i)$，其中 $1 \leq i \leq n$，并给出阈值 $W$。

选择一个 $\{1, 2, \cdots, n\}$ 的子集 $S$，使得 $\sum\limits_{i \in S} w_i \leq W$，且最大化 $\sum\limits_{i \in S} v_i$。

显然 $w_i > W$ 的数对并没有作用，因此为了下文方便起见，我们假定不存在这样的数对。

### 1.8.1 算法 1

把所有数对按照 $\frac{v_i}{w_i}$ 从大到小排序，依次贪心决策，如果能放入集合 $S$ 就放入，否则跳过。

不幸的是，这个算法的效果非常差。考虑如下构造：有两个数对 $(2, 1), (W, W)$。贪心算法将选取第一个数对，然而 $W \geq 2$ 时，最优解是选取第二个数对。两种方案的答案比值为 $\frac{2}{W}$，也就是说，我们的算法没有常数近似比，在 $W$ 足够大时它的近似比将趋近于 0。

### 1.8.2 算法 2

我们仍然采取算法 1 的贪心策略，不同的是，我们加入这样一个特判：如果 $v_i$ 最大的数对的 $v_i$ 大于我们用算法 1 计算出的答案，就把它作为真正的答案。

**定理 1.10.** 这是一个 $\frac{1}{2}$-近似算法。

证明. 考虑原问题的弱化版：每个物品可以选取非整数个（称为分数背包）。形式化地，对每个物品设置变量 $0 \le x_i \le 1$，要求 $\sum w_i x_i \le W$，且 $\sum v_i x_i$ 最大。显然这个问题的答案大于等于原问题答案，且这个问题是容易解决的：按照性价比排序，能选就选，即可得到最优解。

我们的贪心算法，和分数背包的算法相比，只相差了至多一个物品的价值。因此：贪心算法的答案与价值最大的物品的价值之和，一定大于等于分数背包的最优解，也就大于等于原问题的最优解 OPT。由鸽笼原理，二者至少有一个 $\ge \frac{1}{2}\mathrm{OPT}$。 □

### 1.8.3 算法 3

我们已经知道，对于背包问题，存在关于 $v_i$ 值域与 $n$ 的多项式复杂度动态规划算法。

设一变量 $\varepsilon$，令 $v'_i = \left\lceil \frac{v_i}{b} \right\rceil$，其中 $b = \frac{\varepsilon}{n} \max_i v_i$。

此时，$v'_i$ 的值域被限制在了 $\mathrm{poly}\left(\frac{1}{\varepsilon}, n\right)$ 内。对 $(v'_i, w_i)$ 进行动态规划，即可得到一个 $\mathrm{poly}\left(\frac{1}{\varepsilon}, n\right)$ 复杂度的算法。

**定理 1.11.** 这个算法是 $(1-\varepsilon)$-近似的。

证明. 记 $S$ 为我们的算法选择的集合，$S'$ 为最优解选择的集合。记 $\overline{v_i} = bv'_i$。

$$\sum_{i \in S'} v_i \le \sum_{i \in S'} \overline{v_i} \le \sum_{i \in S} \overline{v_i} \le \sum_{i \in S} (v_i + b) \le nb + \sum_{i \in S} v_i$$

代入 $b$ 的值并放缩 $\max_i v_i \le \sum_{i \in S'} v_i$，并移项，即可得到：

$$(1-\varepsilon) \sum_{i \in S'} v_i \le \sum_{i \in S} v_i$$

从而这是一个 $(1-\varepsilon)$-近似。 □

这个算法说明了，对于任意常数近似比，我们可以在多项式复杂度内求出该近似比下的背包解。

## 2 线性规划

### 2.1 引入

线性规划在信息学竞赛中是一个常被提及的知识点。它的形式如下：

**问题 2.1** (线性规划，LP)**.** 给出 $n \times m$ 矩阵 $A$，$n$ 行的列向量 $b$，$m$ 行的列向量 $c$。要求求解 $m$ 行的列向量 $x$，满足：$Ax \leq b$，且 $c^T x$ 最大。

显然，通过对一些元素取相反数，这里提到的最大可以换成最小，小于等于号也可以换成大于等于号。

1979 年，线性规划的弱多项式复杂度算法被苏联科学家 Leonid Khachiyan 提出。[11] 其中，弱多项式意味着它是一个关于输入的总位数的多项式复杂度算法。线性规划在近似算法之中的应用十分广泛，下面，我们介绍一些相关的问题。

## 2.2 顶点覆盖问题

**问题 2.2** (顶点覆盖问题)**.** 给定一张无向图 $(V, E)$，点有点权 $C_i$，选择一个集合 $S \subseteq V$，使得对于所有边 $(u, v) \in E$，都有 $u \in S$ 或 $v \in S$。最小化 $\sum_{u \in S} C_u$。

### 2.2.1 算法

考虑求解如下线性规划问题：

$$\min C^t x$$

$$s.t.\ x_i + x_j \geq 1 \text{ for edge } (i, j) \in E$$

$$0 \leq x_i \leq 1 \text{ for all } i \in V$$

对于求解出的解，我们把点 $u$ 加入 $S$ 当且仅当 $x_u \geq \frac{1}{2}$。

### 2.2.2 分析

首先，我们需要说明这个算法构造的方案是正确的。因为对于边 $(i, j)$，有 $x_i + x_j \geq 1$，从而 $x_i, x_j$ 至少有一个 $\geq \frac{1}{2}$，从而被加入 $S$ 中。因此，我们的确覆盖了每条边。

**定理 2.1.** 这个算法是 2-近似的。

证明. 显然，原问题的最优解大于等于 LP 问题的最优解。

同时，由于我们只会把 $\geq \frac{1}{2}$ 的权值改成 1，我们构造出的解不超过 LP 问题解的两倍。从而这是一个 2-近似算法。 □

## 2.3 集合覆盖问题

**问题 2.3** (集合覆盖问题)**.** 令 $U = \{1, 2, \cdots, n\}$，给出若干 $S_i, C_i$，其中 $S_i$ 为 $U$ 的子集，$C_i$ 为非负数。选出其中的一些，使得选出的 $S_i$ 的并集为 $U$，且 $C_i$ 之和最小。

先前我们已经给出了一个 $H_n$-近似的算法。这一次，我们尝试用 LP 的观点看待这个问题。

## 2.4 算法

考虑求解如下线性规划问题：

$$\begin{aligned} &\min C^t x \\ s.t. &\sum_{i:j\in S_i} x_i \geq 1, \forall j \in U \\ &0 \leq x_i \leq 1 \end{aligned}$$

进行随机舍入：假设求解出 $x_i$，则以 $x_i$ 的几率选择 $S_i$，以 $(1-x_i)$ 的几率不选择。

显然这并不总是给出合法的解，因此我们重复多轮这一操作，并把选择的集合取并，直到合法为止。

## 2.5 分析

首先，一轮花费的期望代价为 $\sum C_i x_i \leq \mathrm{OPT}$。因此我们需要考虑期望进行的轮数。

对于每个元素 $j$，我们知道 $\sum_{i:j\in S_i} x_i \geq 1$，而它一轮后未被覆盖的几率为 $\prod_{i:j\in S_i}(1-x_i)$，这在 $x_i$ 相等时取得最大值 $\left(1-\frac{1}{k}\right)^k \leq \frac{1}{e}$。

因此，重复这一过程 $3\ln n$ 轮，元素 $j$ 被覆盖的几率至少为 $1-\left(\frac{1}{e}\right)^{3\ln n} = 1-\frac{1}{n^3}$。由 Union Bound，所有元素被覆盖的几率至少为 $1-\frac{1}{n^2}$。从而说明操作的轮数是 $O(\log n)$ 的。

因此，上述算法的近似比是 $O(\log n)$ 的。

# 3 半正定规划

## 3.1 引入

可以看出，线性规划已经是一个比较强的工具。但是它的“线性”这一点，仍然给我们的建模带来了一些阻碍。由此，我们将其扩展到了半正定规划的范畴内。在此之前，我们先介绍一些基础概念。

**定义 3.1** ($\mathbb{R}$ 上的内积)**.** 对于 $n$ 阶实矩阵 $A, B$，定义内积 $\langle A, B\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}B_{i,j}$。

对于 $n$ 维实向量 $x, y$，定义内积 $\langle x, y\rangle = \sum_{i=1}^{n} x_i y_i$。

**定义 3.2** (半正定矩阵)**.** 对于 $n$ 阶实对称矩阵 $A$，若对所有 $n$ 维实非零列向量 $x$ 都有 $x^T A x \geq 0$，则称 $A$ 为半正定的。

可以说明：

- 对于 $n$ 阶实对称矩阵 $A$，$A$ 的所有特征值非负。
- 对于 $n$ 阶实对称矩阵 $A$，存在实矩阵 $B$ 使得 $A = B^T B$。
- 对于 $n$ 阶实对称矩阵 $A$，存在 $v_1, v_2, \cdots, v_n \in \mathbb{R}^m$，使得 $A_{i,j} = \langle v_i, v_j \rangle$。

上述条件的任何一个满足都当且仅当 $A$ 是半正定矩阵。

此外，给定一个半正定矩阵，存在多项式复杂度的算法求出 $v_1, v_2, \cdots, v_n \in \mathbb{R}^m$，使得 $A_{i,j} = \langle v_i, v_j \rangle$。

**问题 3.1** (半正定规划，SDP)**.**

$$\min \langle C, X \rangle$$
$$s.t. \ \langle A_i, X \rangle \geq b_i$$
$$X \succeq 0$$

其中 $X \succeq 0$ 代表 $X$ 为半正定矩阵。

可以发现，LP 等强于 SDP 的所有矩阵退化为对角阵的特殊情况。
半正定规划对任何精度 $\varepsilon$ 存在关于 $\log\left(\frac{1}{\varepsilon}\right)$ 的多项式复杂度算法。

## 3.2 最大割

**问题 3.2** (最大割)**.** 给定带权无向图 $(V, E)$，边权为 $c(u, v)$。把点集划分成 $S$ 和 $V - S$，使得 $\sum_{u \in S, v \notin S} c(u, v)$ 最大。

使用与最大 3-SAT 相同的方法，可以得到一个 $\frac{1}{2}$-近似的算法，不过我们希望做得更好。

### 3.2.1 算法

[6] 形式化地描述最大割问题：

$$\max \frac{1}{2} \sum_{(i,j) \in E} c(i, j)(1 - y_i y_j)$$
$$s.t. \ y_i \in \{-1, 1\}, \forall i \in V$$

不妨把 $y_i$ 的范围扩大到 $\mathbb{R}^n$ 中的向量，满足 $||y_i|| = 1$，则设 $X_{i,j} = \langle y_i, y_j \rangle$，这就是半正定矩阵的定义！问题变为：

$$\max \frac{1}{2} \sum_{(i,j)\in E} c(i,j)(1 - X_{i,j})$$

$$s.t.\ X_{i,i} = 1, \forall i \in V$$

$$X \succeq 0$$

这是一个标准的半正定规划问题，求解它，可以得到所有的 $y_i$。接下来，我们要把它们舍入到 $\{-1, 1\}$。

考虑随机选取一个单位向量 $r \in \mathbb{R}^n$，把所有 $\langle y_i, r\rangle \geq 0$ 的 $i$ 划分入 $S$。

### 3.2.2 分析

接下来，我们说明这个算法的期望近似比。

对于任意边，考虑计算它的两端点被分入不同类的几率，即 $y_i, y_j$ 和 $r$ 的内积符号不同的几率。设 $W = \text{span}(\{y_i, y_j\})$，则 $\mathbb{R}^n = W + W^\perp$，令 $r = u + v$，其中 $u \in W, v \in W^\perp$，则可以把 $r$ 换成 $u$。显然，符号只和 $u$ 的方向有关，而 $u$ 是球面上均匀分布的 $r$ 在二维平面上的投影，故 $u$ 的方向是均匀分布的。因此，只需解决二维的情况，而这是简单的：这个几率应当为为 $\frac{\arccos\langle y_i, y_j\rangle}{\pi}$。

记 $\theta_{i,j} = \arccos\langle y_i, y_j\rangle$，则上面的几率表示为 $\frac{\theta_{i,j}}{\pi}$。而这条边在 SDP 内的贡献为 $\frac{1}{2}(1 - \cos\theta_{i,j})$。

对一条边，计算二者比值为 $\frac{2\theta_{i,j}}{\pi(1-\cos\theta_{i,j})}$，可以证明，对于 $0 \leq \theta_{i,j} \leq \pi$ 其最小值约为 0.878。从而我们得到了一个期望 0.878-近似的算法。

因此，对任意常数 $\varepsilon$，我们都可以在多项式的时间内以极高的概率得到一个 $(0.878-\varepsilon)$-近似的解。

# 4 近似的困难性

## 4.1 引入

我们已经给出了一些问题的近似算法，那么下一个问题就是，我们该如何评估它们是否足够好？

在先前的信息学竞赛学习中，我们已经学习了将问题归约到已知的“困难问题”（例如，矩阵乘法，NPC 问题等）从而证明其困难性的方法，实际上，在前文中，我们已经用类似的方法说明了 TSP 问题在近似上的困难性，那么，我们还可以证明哪些问题的近似比下界呢？

下面，我们给出一些问题的近似困难性的研究。

## 4.2 K 中心问题

**定理 4.1** (K 中心问题的困难性)**.** 对任意 $\varepsilon > 0$，$K$ 中心问题的 $(2-\varepsilon)$-近似是 *NP-Hard* 的。

证明. 我们将 K 中心问题的 $(2-\varepsilon)$-近似归约到无向图的支配集问题。支配集问题是说：给定一张图和正整数 $K$，判断是否能选择不超过 $K$ 个点，使得所有点距离最近被选择点的距离 $\leq 1$。这个问题是 NP-Hard 的。

考虑如下定义 $d(u,v)$：$d(u,u) = 0$，对于原图中有的边 $(u,v)$，$d(u,v) = 1$，其它情况 $d(u,v) = 2$。不难验证它满足三角不等式。此时，若存在大小为 $K$ 的支配集，则 K 中心问题的答案为 1，否则答案为 2。若存在多项式复杂度 $(2-\varepsilon)$-近似，则可以区分这两种情况，从而 P = NP。 □

而我们刚刚已经得到了这个问题的 2-近似！这说明了我们得到的算法是足够优秀的。

## 4.3 装箱问题

**问题 4.1** (装箱问题)**.** 给出 $n$ 个 $(0,1]$ 内的值 $a_1, \cdots, a_n$。求一组把 $S = \{1, 2, \cdots, n\}$ 划分成 $S = T_1 \cup T_2 \cup \cdots T_m$ 的方案，满足对任意 $1 \leq i \leq m$，$\sum\limits_{j \in T_i} a_j \leq 1$，且 $m$ 最小。

**定理 4.2** (装箱问题的困难性)**.** 对任意 $\varepsilon > 0$，装箱问题的 $\left(\frac{3}{2} - \varepsilon\right)$-近似是 *NP-Hard* 的。

证明. 考虑归约到如下问题：对数列 $\{c_1, \cdots, c_n\}$，判断能否将其划分为两个和相等的子集。这个问题是 NP-Hard 的。

设 $a_i = \frac{2c_i}{\sum\limits_{i=1}^{n} c_i}$，若上述问题有解，则装箱问题的解为 2，否则其解至少为 3。因此，若存在 $\left(\frac{3}{2} - \varepsilon\right)$-近似，就解决了这一 NP-Hard 问题。从而我们得到了一个归约。 □

有趣的是，尽管要求严格的 $(\frac{3}{2} - \varepsilon)$-近似是困难的，但是如果我们放宽要求的话，可以证明：直接将所有物品从大到小排序，并贪心放入当前最小的箱子，放不下了再开新箱子，这一算法满足 $\text{SOL} \leq \frac{11}{9}\text{OPT} + \frac{6}{9}$。[4]

## 4.4 其它问题

实际上，在证明近似的困难性这一领域，人们已经发明了许多工具，例如 PCP 定理，和目前尚无定论的 UGC（Unique Games Conjecture）等。受限于笔者的水平，无法在这里作详细的介绍。但至少笔者可以在这里列举前文提到的问题目前被证明的上下界，供读者参考（假设 P ≠ NP）：

- 负载均衡问题：对任意常数 $\varepsilon > 0$，存在多项式复杂度的 $(1+\varepsilon)$-近似算法。[7]

- 集合覆盖问题：不存在 $(1-\varepsilon)\ln n$-近似。值得一提的是我们给出的 $H_n$-近似算法已经有 $H_n \le \ln n + 1$，这说明这个算法很接近最优。[5]
- 度量 TSP：近似比下界为 $\frac{123}{122}$[10]，存在约 $(1.5 - 10^{-36})$-近似。[9]
- 斯坦纳树：近似比下界为 $\frac{96}{95}$[2]，存在约 1.39-近似。[1]
- 最大 3-SAT：近似比的上下界均为 $\frac{7}{8}$。[8]
- 顶点覆盖：近似比下界约为 1.3606[3]，特别地，如果承认 UGC，那么近似比下界为 2。[13] 且存在近似比为 2 的算法。
- 最大割：近似比下界为 $\frac{16}{17}$[8]，特别地，如果承认 UGC，那么近似比下界是上文提到算法所得到的 0.878。[12]

## 5 总结

本文介绍了一系列经典的问题及其近似算法，介绍了贪心、随机化、层级化、线性规划、半正定规划等方法在其中的应用，并说明了一些问题的近似困难性。笔者希望本文起到抛砖引玉的作用，让读者获得对近似算法的初步认识，从而进一步地探索近似算法这一在信息学竞赛中几乎尚未被发掘的领域。

## 6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢家人、朋友、教练对我的关心、鼓励和支持。

感谢黄镜元同学在本文写作过程中提供的帮助。

感谢陈佳雨同学、黄镜元同学、胡墨同学、赖泓锐同学、孙铁铮同学、王德懋同学为本文验稿并提出建议。

## 参考文献

[1] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. J. ACM, 60(1), 2013.

[2] Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. Theoretical Computer Science, 406(3), 2008.

[3] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. Annals of Mathematics, 162(1), 2005.

[4] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $\mathrm{ffd}(i) \leq 11/9\mathrm{opt}(i) + 6/9$. In Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. Springer Berlin Heidelberg, 2007.

[5] Uriel Feige. A threshold of ln n for approximating set cover. J. ACM, 45(4), 1998.

[6] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM, 42(6), 1995.

[7] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. In 26th Annual Symposium on Foundations of Computer Science (sfcs 1985), 1985.

[8] Johan Håstad. Some optimal inapproximability results. J. ACM, 48(4), 2001.

[9] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, New York, NY, USA, 2021. Association for Computing Machinery.

[10] Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for tsp. 81(8), 2015.

[11] Leonid Khachiyan. A polynomial algorithm for linear programming. Doklady Akademii Nauk SSSR, 224(5), 1979.

[12] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In 45th Annual IEEE Symposium on Foundations of Computer Science, 2004.

[13] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. Journal of Computer and System Sciences, 74(3), 2008.

[14] Jon Kleinberg and Éva Tardos. Algorithm Design. Addison-Wesley, 2005.

# 不同组合对象的哈希方法以及应用

福建师范大学附属中学 林祺昊

## 1 摘要

哈希思想在信息学奥林匹克竞赛以及应用中是一类重要的方法，其基本问题为通过压缩存储的信息量加速对组合对象的计算。其简洁直观的特点使得其在大量问题中得到了应用。本文将在哈希思想的基础上介绍不同组合对象的哈希方法及应用，并建立多项式、有序序列、无序序列、树哈希、图哈希之间的联系，最后介绍部分确定性的哈希算法，并提出二叉合并理论。

## 2 哈希思想

哈希思想常常被应用于判断两个组合对象是否相等，其为通过将一个复杂组合对象的信息通过若干个简单量（下称为哈希值）刻画。一个理想的哈希函数应该使相同的对象可以被映射到相同的哈希值上，将不同的对象映射到不同的哈希值上（反之称为哈希冲突或哈希碰撞），从而根据判断两个组合对象的哈希值是否相等来判断这两个组合对象是否相同。

然而，由于信息论的限制，无法设计出一个可以减小组合对象信息量的抗碰撞的确定哈希函数。因此只能够通过不确定算法，使得两个不同的组合对象具有相同的哈希值的概率在可接受范围内（在信息学竞赛以及应用中，往往只需做到 $1-\frac{1}{2^{32}}$ 的正确率或者更高）。

并且，在一部分问题中，并不止需要判断两个组合对象是否相同，这时需要一些别的技巧。不同的哈希方法往往具有不同的性质（如易于维护、可合并、正确率等），在实际应用中往往需要根据需要选取适合的方式使用。

值得注意的是，如果允许通过根据先前的输入动态调整哈希函数，则有时可以得到确定性的哈希算法。

下文中，笔者将先对于每种不同的组合对象（Laurent 多项式、集合、模意义下的 Laurent 多项式、有序列表、循环序列、树、图）给出常见的哈希方法并指出它们的异同点，并在文章最后分析部分确定性的哈希算法，最后给出一些其它哈希应用。

## 3 符号与约定

在下文中，记 $|F|$ 表示一个群或域中元素的个数。

Laurent 多项式指的是一个形如 $\sum f_i x^i$ 的和式，其中 $f$ 只有有限项不为 0，因此在只考虑这些项时，此处的指标求和范围有上下界。

笔者将使用符号 $\oplus_p$ 表示 $p$ 进制下的不进位加法，若无下标则默认 $p = 2$。使用符号 $\otimes$ 表示 2 进制下的 Nimber 积。

同时，使用符号 $S + T$ 表示可重集 $S, T$ 之间的加法，使用符号 $-$ 表示可重集 $S, T$ 之间的减法，此时应有 $T \subseteq S$，使用符号 $S \oplus T$ 表示集合 $S, T$ 之间的对称差。

使用 $F(x), G(x)$ 等符号表示一个关于 $x$ 的函数，用 $f_i, g_i$ 等符号表示对应大写字母代表的函数的各项系数，即 $f_i = [x^i]F(x), g_i = [x^i]G(x)$。

在本文中，视哈希表每次调用的时间复杂度为 $O(1)$。

# 4 Laurent 多项式哈希

最为基础的哈希是对于一个 Laurent 多项式进行哈希，其与普通多项式最大的区别在于 $x$ 的指数可以为负数，事实上这与普通多项式的区别不大，即其将多项式的项数扩展到了负数次幂。

## 4.1 幂次赋权哈希方法

任取一奇质数 $p$ 以及其对应的域 $\mathbb{F}_p$，对于域上各项指数中 $x$ 的指数均在 $[-n, n]$ 的不同的 Laurent 多项式 $F(x) = \sum_{i=-n}^{n} f_i x^i$ 和 $G(x) = \sum_{i=-n}^{n} g_i x^i$，有以下结论：

**定理 4.1.** 随机选取一个不超过 $p$ 的非负整数 $x_0$，则 $F(x_0) \equiv G(x_0) \pmod p$ 的概率不超过 $\frac{2n}{p}$。

证明. 取多项式 $H(x) = (F(x) - G(x))x^n$，则 $H(x)$ 的次数不超过 $2n$ 且非零。

由于非零多项式在有限域上的零点个数不超过其次数，因此 $H(x) \equiv 0 \pmod p$ 的解至多只有 $2n$ 个，因此 $F(x_0) \equiv G(x_0) \pmod p$ 的概率不超过 $\frac{2n}{p}$。 □

使用 Union Bound 立刻可以得到以下推论：

**推论 4.1.** 对于任意 $k$ 个两两不同的各项指数绝对值不超过 $n$ 的 *Laurent* 多项式，随机选取 $x_0$，则它们的哈希值两两不同的概率至少为 $1 - \frac{nk^2}{p}$。

事实上，可以将上述定理进行推广：

**定理 4.2** (Schwartz–Zippel 引理)**.** 对于有限域 $F$ 上的 $n$ 元多项式 $P(x_1, x_2, \cdots, x_n)$，若所有 $x_i$ 均在域 $F$ 中独立均匀随机选取，则 $\Pr[P(x_1, x_2, \cdots, x_n) = 0] \le \frac{d}{|F|}$。

该哈希方式具有良好的可合并性，即给定 $c_1, c_2$，通过 Laurent 多项式 $F(x), G(x)$ 的哈希值可以在常数复杂度内得到多项式 $c_1F(x) + c_2G(x)$ 的哈希值。更进一步的，该哈希方式可以对于给定的整数 $k$ 高效得到 $F(x)x^k$ 以及 $F(x)G(x)$ 的哈希值。

在实际应用中，常常采用质数 $p$ 的任一原根作为 $x_0$ 代入进行检验。

这种哈希方式的缺点是对于 Laurent 多项式的次数有限制，并且当 $n$ 较大时但 $F$ 非零项稀疏时的计算需要时间复杂度为 $O(\log p)$ 的快速幂，下述的另一种哈希方式可以解决该问题。

## 4.2 随机数赋权哈希方法

任取一独立均匀随机且可在常数复杂度内计算的函数 $H : \mathbb{Z} \to \mathbb{Z}_p$，并将给定 Laurent 多项式 $F(x) = \sum f_i x^i$ 的哈希值定义为 $\sum f_i H(i) \bmod p$，此时有如下定理：

**定理 4.3.** 对于不同的 *Laurent* 多项式 $F(x)$ 和 $G(x)$，用上述方式计算的哈希值相等的概率相等的概率为 $\frac{1}{p}$。

该定理的证明只需要用到对于任意不为 0 的 $x$，所有 $kx(0 \le k < p)$ 构成一个模 $p$ 的剩余系下的完系即可。

同样的，使用 Union Bound 立刻可以得到以下推论：

**推论 4.2.** 对于任意 $k$ 个两两不同的 *Laurent* 多项式，它们的上述哈希值两两不同的概率至少为 $1 - \frac{k^2}{2p}$。

可以看出，这种哈希方式仍然保留了可合并性的优点，并且 $p$ 的选取可以无视 Laurent 多项式的最高次数，且计算单个 Laurent 多项式的哈希值时不需要快速幂的复杂度，但其丢失了可高效计算 $F(x)x^k$ 以及 $F(x)G(x)$ 的哈希值的优点。

事实上，在实际应用中，无论次数 $n$ 的上界为多少，在随机选取大质数 $p$ 的情况下，以上两种哈希方式基本上都能有可观的正确率。

# 5 可重集合哈希

可重集合是信息学竞赛中一类非常常见的组合对象，可重集合之间有加减运算，并且可以与标量进行乘法，它也是最为简单进行哈希的对象之一，因为其与 Laurent 多项式的哈希几乎没有区别。

常常使用一个哑元 $x$ 作为它与 Laurent 多项式的哈希之间的桥梁，此处的 $x$ 无实际意义，仅为便于推导代数式子，最后可以机械地化为不带 $x$ 的哈希形式。

## 5.1 与 Laurent 多项式哈希之间的联系

定义 $S$ 为一个整数的可重集合，显然函数 $F(S)=\sum\limits_{v\in S}x^v$ 为函数的一个哈希函数（当 $v$ 在 $S$ 中出现多次时 $x^v$ 也计算多次），其将集合 $S$ 映射到一个关于 $x$ 的 Laurent 多项式上。

因此，可重集合哈希可被转化为对 Laurent 多项式进行哈希，此时对应的最高次数为 $\max\limits_{v\in S}|v|$，因此它也可以使用上文的两种哈希方法进行处理。

这样的哈希方式即可支持可重集合加减法，以及将一个可重集合中每一个元素复制 $k$ 遍。若使用幂次赋权哈希方式，则还可以支持将集合内的每一个元素都加上同一个整数。

**例题 5.1** (可重集[1])**.** 给定长度为 $n$ 的数列 $a$，需要支持 $q$ 次操作，每次操作为以下两种之一：

- 给定 $x,y$，将 $a_x$ 修改为 $y$；
- 给定 $l_1,r_1,l_2,r_2$，保证 $r_1-l_1=r_2-l_2\geq 0$，判断是否存在整数 $d$ 使得可重集合 $\{a_{l_1}+d,a_{l_1+1}+d,\cdots a_{r_1}+d\}$ 与 $\{a_{l_2},a_{l_2},\cdots a_{r_2}\}$ 相等。

保证 $n,q\leq 10^6,0\leq a_i,y\leq 10^6$。

可以使用线段树支持单点修改以及查询区间最小值，记区间 $[l_1,r_1]$ 和区间 $[l_2,r_2]$ 的最小值分别为 $c_1$ 和 $c_2$。那么，若存在一个这样的 $d$ 满足两个可重集合相等，则必然有 $d=c_2-c_1$。

因此，只需要对于 $d=c_2-c_1$ 的情况判断是否有 $\{a_{l_1}+d,a_{l_1+1}+d,\cdots a_{r_1}+d\}$ 与 $\{a_{l_2},a_{l_2},\cdots a_{r_2}\}$ 相等即可。

记 $F_1(x)=\sum\limits_{i=l_1}^{r_1}x^{a_i},F_2(x)=\sum\limits_{i=l_2}^{r_2}x^{a_i}$，则上述条件等价于 $F_1(x)x^{c_2-c_1}=F_2(x)$。由于需要进行多项式平移，因此只能使用上文所述的幂次赋权哈希方式。该算法的时间复杂度为 $O(n\log n)$（视 $n,q,\max\{a_i\},\max\{y\}$ 同阶）。

**例题 5.2** (Same Sum[2])**.** 给定长度为 $n$ 的数列 $a$，需要支持 $q$ 次操作，每次操作为以下两种之一：

- 给定 $L,R,v$，将 $a_L,a_{L+1},\cdots,a_R$ 加上 $v$；
- 给定 $L,R$，保证 $R-L+1$ 为偶数，判断是否能将 $a_L,a_{L+1},\cdots,a_R$ 两两分组，使得在 $\frac{R-L+1}{2}$ 组中每组的两个数字的和均相同。

保证 $n,q\leq 2\times 10^5,0\leq a_i,v\leq 2\times 10^5,L\leq R$。

[1]来源：P6688, 洛谷

[2]来源：Problem G, CCPC 2024 Zhengzhou Site

容易使用线段树维护区间和 $S$，记 $c=\frac{2S}{R-L+1}$，则若 $a_L,a_{L+1},\cdots,a_R$ 可被划分为若干个二元组且划分方案满足题意，则每组的和必然为 $c$。

因此 $c$ 必然为整数，且题目所叙条件等价于可重集合 $\{a_L,a_{L+1},\cdots,a_R\}$ 与可重集合 $\{c-a_L,c-a_{L+1},\cdots,c-a_R\}$ 相等。

故考虑记 $F_+(x)=\sum_{i=L}^{R}x^{a_i},F_-(x)=\sum_{i=L}^{R}x^{-a_i}$，则只需检查是否有 $F_+(x)=F_-(x)x^c$。使用上文所述的幂次赋权哈希方式维护即可。总时间复杂度为 $O(n\log n)$（视 $n,q,\max\{a_i\},\max\{v\}$ 同阶）。

## 5.2 处理部分与不可重集合有关的哈希问题

**例题 5.3** (星战[3])**.** 给定一张 $n$ 个点 $m$ 条边的有向图 $G$，初始时所有边均为激活状态，需要进行 $q$ 次操作，每次操作为以下四种之一：

- 给定有向边 $(u,v)$，保证其为激活状态，将其状态修改为未激活；
- 给定节点 $u$，将所有连向 $u$ 的点状态修改为未激活；
- 给定有向边 $(u,v)$，保证其为未激活状态，将其状态修改为激活；
- 给定节点 $u$，将所有连向 $u$ 的点状态修改为激活。

每次修改后，你需要判断此时被激活的边是否构成一个内向基环树森林。

保证 $n,m,q\le 5\times 10^5$。

边构成一个内向基环树森林等价于每一个顶点的出度为 1，于是可以自然地想到维护每条边的起点构成的可重集 $S$，则第一种操作与第三种操作对于集合 $S$ 的影响即为加入或删除一个元素。

但是在进行第二类和第四类操作时，不能够直接加入或删去其所有入边对应的入点构成的集合 $A_i$，这是因为可能存在已经被激活 / 未激活过的边，此时会使得这条边对 $S$ 的贡献错误。

因此，再维护 $B_i$ 表示第 $i$ 个点的所有被激活的入边对应的入点构成的集合，则未被激活的入边对应的入点构成的集合为 $A_i-B_i$。这样就可以处理第二类操作以及第四类操作了。

为了动态维护 $B_i$，只需在操作一或操作三后加入或删除对应元素，在操作二后将 $B_i$ 清空，在操作四后将 $B_i$ 赋值为 $A_i$ 即可。

使用上述随机数赋权哈希方法进行维护即可，时间复杂度为 $O(n+m+q)$。

从上例中可以看出，这种哈希方法有时也可以解决维护集合时的“去重”问题。

[3] 来源：CSP-S 2022 第三题

## 5.3 处理另一种等价关系下的哈希

在算法竞赛中，还有一种常见问题为给定两个有序列表，判断是否存在一个双射，使得将第一个有序列表经过映射后与第二个有序列表相等。

这虽然是有关有序列表的问题，但是可以将其转为关于（可重）集合的哈希问题。具体而言，记组合对象 $x$ 出现的位置集合为 $S_x$，则存在一个这样的双射当且仅当在两个序列中，所有 $S_x$ 构成的集合相等。

但是由于维护的可重集合中的对象为集合，因此无法对最里部元素快速进行整体计算（例如整体加上一个整数等），故无法完成在一个序列中截取一段的操作。解决这个问题的方法将在下文中进行探讨（可参看“有序列表的哈希”一节）。

**例题 5.4** (集合[4])**.** 给定两个长度为 $n$ 的有序列表 $A, B$，其中 $A, B$ 的元素均为一个大小为 3 的不可重集合，记集合的三个元素为 $A_{i,0}, A_{i,1}, A_{i,2}$ 或 $B_{i,0}, B_{i,1}, B_{i,2}$，保证元素的大小均不超过 $m$。

现在给定 $q$ 次询问，每次询问给定 $[l, r]$，你需要回答是否存在一个整数到整数的一一映射 $f$，使得对于任意 $l \le i \le r$ 均有 $\{f(A_{i,0}), f(A_{i,1}), f(A_{i,2})\} = \{B_{i,0}, B_{i,1}, B_{i,2}\}$。

保证 $n, m, q \le 5 \times 10^5$。

这即为上文中提到的模型，因此只需判断区间 $[l, r]$ 上对应的 $A$ 的哈希值和 $B$ 的哈希值是否相同。

但是由于该算法的局限性，无法在序列中截取一段完成计算，但是它允许在 $O(1)$ 的时间复杂度内插入或删除一个元素。

当 $l$ 一定时，显然存在 $p_l$ 使得 $[l, r]$ 符合题意当且仅当 $r \le p_l$，因此可以使用双指针的方式分别维护两个有序列表 $A, B$ 对应子段的哈希值。即初始时将左端点设为 1，在左端点增加的时候在满足 $[l, r]$ 合法的前提下尽可能地向右调整右端点 $r$，并将调整后的 $r$ 设置为 $p_l$。

在插入或删除一个 $A_i$ 或 $B_i$ 的过程中，相当于对位置集合的集合进行至多 3 次修改。记 $F(S) = \sum_{T \in S} x^{\sum_{v \in T} x^v}$，则只需在过程中动态维护 $S$ 的哈希值即可，例如 $\sum_{T \in S} H_1\left(\sum_{v \in T} H_2(v)\right)$，其中 $H_1, H_2$ 为两个均匀随机且可在常数时间复杂度内计算的函数。容易发现在进行一次对 $S$ 的修改时，可以 $O(1)$ 计算出新的哈希值，因此该算法的时间复杂度为 $O(n + q)$。

# 6 随机数赋权法解决模质数下的 Laurent 多项式哈希

根据上述讨论，对于 Laurent 多项式的哈希方法和集合的哈希方法是可以互相转化的。因此在下文中，笔者仅仅讨论代数上更为直观的关于 Laurent 多项式的哈希方法，得出对应的集合哈希方法是不难的。

[4]来源：NOI 2024 第一天第一题

有时，在 Laurent 多项式的计算中，由于一些原因需要将每一项的系数对某个质数 $p(p \geq 2)$ 取模，例如在做集合对称差运算操作时就对应了 $p = 2$ 的情况。

## 6.1 扩展先前的哈希方法以及该理论的一般化

由于 $p$ 为质数，因此可以任取一个特征为 $p$ 的有限域，两个 Laurent 多项式 $F(x), G(x)$ 的和即为对应哈希值在域上的和，而 $kF(x)(k \in \mathbb{Z})$ 也可以通过域中的运算实现，并且计算 $F(x)x^k$ 以及 $F(x)G(x)$ 也可以转化为域中两个元素的一次乘法。

在信息学竞赛中，常常需要对于给定的序列求出一个区间构成的（可重）集合的哈希值。一种常见的处理方式为预处理出序列各元素哈希值的前缀和 $S_i$，并在查询子区间 $[l, r]$ 时使用 $S_r - S_{l-1}$ 计算出对应的哈希值，而其中的加减运算均在域中进行。

上述讨论可以说明 Laurent 多项式的哈希可以通过寻找一个对应的域来解决。特别的，若无计算 $F(x)x^k$ 以及 $F(x)G(x)$ 的需求，则只需找到一个交换群即可。例如，在 $p$ 为质数时，取交换群 $\mathbb{Z}_p^k(k \in \mathbb{Z}^*)$。此时群中元素为所有 $[0, p^k)$ 间的整数，对应的二元运算为 $k$ 位 $p$ 进制二进制加法 $\oplus_p$。

同时，为了保证正确率，仍需要一独立均匀随机且可在常数复杂度内计算的函数 $H : \mathbb{Z} \to \left(\mathbb{Z}_p\right)^k$。这时，将一个 Laurent 多项式 $F(x)$ 的哈希值定义为 $\sum f_i H(i)$，此处求和为群中的 $\oplus_p$ 运算。

类似先前提到的随机数赋权哈希方法，可以得到同样的结论：

**定理 6.1.** 对于模 $p$ 下不同的 *Laurent* 多项式 $F(x)$ 和 $G(x)$，用上述方式计算的哈希值相等的概率相等的概率为 $\frac{1}{p^k}$。

证明可以考虑 $p$ 进制下的 $k$ 位是互相独立的，而每一个对应位哈希值相等的概率均为 $\frac{1}{p}$，因此两个多项式哈希冲突的概率为 $\frac{1}{p^k}$。也即可以将该过程看作是先前提到的哈希方式进行了 $k$ 次检验。

该定理也可以使用 Union Bound 进行推广，形式类似，在此略去不表。

该哈希方式往往用于集合的对称差运算中（即 $p = 2$ 的情形），以下是一些应用。

**例题 6.1** (Odd Mineral Resource[5])**.** 给定一棵由 $n$ 个点组成的树，第 $i$ 个点的点权为 $a_i$，保证所有 $a_i$ 均为不超过 $n$ 的正整数。

给定 $q$ 次询问 $(u_i, v_i, l_i, r_i)$，你需要输出任意一个 $c_i$ 满足 $l_i \leq c_i \leq r_i$ 且 $c_i$ 在节点 $u_i$ 到节点 $v_i$ 经过的路径对应的点权中出现了奇数次，或者输出不存在这样的 $c_i$。

保证 $n, q \leq 3 \times 10^5$。

将第 $i$ 个节点上的集合 $S$ 设为 $\{a_i\}$，并记 $u_i, v_i$ 之间的路径经过的点的集合为 $P$，则询问相当于找到任意一个 $l_i \leq c_i \leq r_i$ 使得 $c_i$ 在 $\bigoplus\limits_{v \in P} S_v$ 中。

[5] 来源：Problem D, Codeforces Round 700 (Div. 1)

用上文所述方法将每一种点权映射到一个 64 位无符号整数上并用其按位异或代替集合对称差，则两个不同的集合具有相同的哈希值的概率为 $\frac{1}{2^{64}}$。

但是，提取一个集合的在 $[l_i, r_i]$ 内的值并不能用上文所述的方法进行维护，因此需要寻找其它的维护方案。

使用前缀和的思想，记第 $i$ 个点到根（不妨设为 1 号节点）的路径上的所有 $S$ 的对称差为 $T_i$，则 $u$ 到 $v$ 的路径上的所有 $S$ 的对称差为 $T_u \oplus T_v \oplus S_{\mathrm{LCA}(u,v)}$，其中 $\mathrm{LCA}(u,v)$ 为 $u,v$ 在树上的最近公共祖先。

先将 $S_{\mathrm{LCA}(u,v)}$ 造成的影响特别处理，发现此时只需要处理 $T_u$ 和 $T_v$ 是否相等。因此可以使用可持久化线段树维护每个 $T_i$，其中线段树下标为 $a$ 的值域。在每个节点维护在仅考虑它对应的值域区间时对应的哈希值，由于只需要找到 $T_u$ 和 $T_v$ 中任一 $[l_i, r_i]$ 内的元素使得其恰好在一个集合中出现，因此可以在线段树上二分。

具体而言，找到 $[l_i, r_i]$ 在线段树上对应的 $O(\log n)$ 个区间，并找到任意一个在 $T_u, T_v$ 对应的可持久化线段树上对应哈希值不同的区间，并将其作为初始节点进行递归。此时若其为叶节点则输出，否则因为左节点和右节点对应的区间中至少有一个区间对应的哈希值是不同的，因此总能进行递归至某侧的子节点作为新的当前节点并递归。

上述算法的总时间复杂度为 $O((n+q)\log n)$。

**例题 6.2** (平方数（弱化版）[6])**.** 给定一个长度为 $n$ 的正整数序列 $a$，求其满足其中所有数字之积为完全平方数的子区间 $[l, r]$ 数量。

保证 $n \le 3 \times 10^5, \max\{a_i\} \le 10^7$。

对于一个正整数 $x = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_m^{\alpha_m}$，若将其的质因子分解形式以集合 $S(x)$ 的形式写出来（若 $\alpha_i$ 的次数为奇数则将 $p_i$ 加入到集合 $S$）中，则 $x$ 为完全平方数当且仅当 $S(x)$ 为空。

进一步的，对于正整数 $x$ 和 $y$，它们的积对应的集合 $S(xy) = S(x) \oplus S(y)$。

在线性筛出每个数字的最小质因子后，计算单个 $a_i$ 对应的 $S(a_i)$ 可以通过不断地将 $a_i$ 除掉最小质因子进行计算，时间复杂度为 $O(\omega(a_i))$，其中 $\omega(a_i)$ 表示 $a_i$ 的质因子数量。

记 $S$ 的前缀异或和为 $T$，则 $S_l \oplus S_{l+1} \oplus \cdots \oplus S_r = \varnothing$ 等价于 $T_{l-1} = T_r$。

将每一个质因子随机映射到一个 64 位无符号整数上并维护用上述方法计算的哈希值，并计算前缀异或和后计算相同数对数量即可，这可以使用哈希表快速计算。综上所述，该问题在 $O(\max\{v_i\} + n\max\{\omega(v_i)\})$ 的时间复杂度内得解。

**例题 6.3** (Hyperregular Bracket Strings[7])**.** 给定数字 $n$ 以及 $k$ 个区间 $[l_i, r_i] \subseteq [1, n]$，求有多少个长为 $n$ 的合法括号序列使得对于任意 $1 \le i \le k$，其下标在 $[l_i, r_i]$ 内的子串也是一个合法括号序列，答案对 998244353 取模。

多组测试。保证单个测试点内测试数据组数 $t \le 10^5$，$\sum n, \sum k \le 3 \times 10^5$。

[6]来源：NOI 2024 陕西省队选拔第二试第一题

[7]来源：Codeforces Round 875 (Div. 1)

若 $n$ 为奇数或某 $r_i - l_i + 1$ 为奇数则答案显然为 0，因此不妨假设 $n$ 为偶数。将合法括号序列看作是 $n$ 个点之间的 $\frac{n}{2}$ 对匹配，使得每一个点恰好出现一次且不存在两个匹配相交但不包含。

事实上，在上述前提下，$[l_i, r_i]$ 内的子串也是合法括号序列当且仅当若将 $l_i - 0.5$ 与 $r_i + 0.5$ 之间连边，则不存在一条匹配边经过 $l_i - 0.5$ 与 $r_i + 0.5$ 之间的边。

因此，这 $k$ 条边将 $n$ 个点划分为了若干个等价类。为了维护该等价类，维护 $n$ 个集合 $S_1, S_2, \cdots, S_n$，对于每个区间 $[l_i, r_i]$，在 $S_{l_i}, S_{l_i+1}, \cdots S_{r_i}$ 之间插入 $i$，则上文的等价类划分与基于 $S$ 的等价类划分得到的结果一致。

使用差分思想，在 $S_{l_i}$ 与 $S_{r_i+1}$ 中加入 $i$，并在 $k$ 个区间处理结束后对 $S$ 求前缀对称差即可。使用上述方式维护，该部分可以做到 $O(n + k)$ 的复杂度（取 64 位无符号正整数进行哈希，则正确率符合要求）。

得到这些等价类后，由于每个等价类之间的连边方案互不干扰，因此记卡特兰数 $C_{2n} = \frac{1}{n+1}\binom{2n}{n}$ 为 $2n$ 个点之间的匹配方案，则答案即为每个等价类大小 $s$ 对应的 $C_{\frac{s}{2}}$ 的乘积。综上所述，本题在 $O(n + k)$ 的总时间复杂度内得解。

## 6.2 处理部分图连通性问题

由于割空间和环空间在 $\mathbb{F}_2$ 的线性空间上的优美性质，因此该哈希方式在图论中也具有广泛应用。下文的讨论基础均基于简单无向连通图 $G$，记图 $G$ 的点集 $V$ 的大小为 $n$，边集 $E$ 的大小为 $m$。下文中的向量与矩阵以及矩阵与矩阵之间的乘法均在域 $\mathbb{F}_2$ 下进行。

**定义 6.1** (图的关联矩阵)**.** 对于一张图 $G$，其关联矩阵是一个 $n$ 行 $m$ 列的矩阵 $\boldsymbol{M}$，其中 $\boldsymbol{M}$ 的第 $i$ 行第 $j$ 列的元素为 1 当且仅当第 $j$ 条边的端点之一为节点 $i$。

**定义 6.2** (图的边空间)**.** 对于一张图 $G$，定义其边空间为所有 $\mathbb{F}_2$ 的 $m$ 维向量，其中向量的每一分量对应了一条边。

**定义 6.3** (图的割)**.** 对于一张图 $G$，可以将其点集划分为两部分 $A, B (A \cap B = \varnothing, A \cup B = V)$，则它们之间的所有边构成了图 $G$ 的一组割。因此，$\boldsymbol{x}$ 为图的割集当且仅当存在向量 $\boldsymbol{y}$ 使得 $\boldsymbol{x} = \boldsymbol{M}^\top \boldsymbol{y}$。

**引理 6.1** (割和点集的对应)**.** 在连通图 $G$ 中，任取一点 $v \in V$ 并强制在上述划分中将 $v$ 划分至集合 $A$ 中，则一组割可以唯一对应一组划分 $A, B$，这也说明了割的数量为 $2^{|V|-1}$。

同时，对于上文的 $\boldsymbol{x} = \boldsymbol{M}^\top \boldsymbol{y}$ 分解，若要求 $\boldsymbol{y}$ 对应 $v$ 的分量上的值为 0，则 $\boldsymbol{x}$ 对应的 $\boldsymbol{y}$ 唯一。

**定理 6.2** (割空间)**.** 连通图的割集构成了一个线性空间，其秩为 $|V| - 1$。记两个割 $\boldsymbol{x_1}, \boldsymbol{x_2}$ 的对称差为 $\boldsymbol{x}$，则其对应的 $\boldsymbol{y}$ 也为 $\boldsymbol{x_1}, \boldsymbol{x_2}$ 对应的 $\boldsymbol{y_1}, \boldsymbol{y_2}$ 的对称差。

该线性空间的一组基可以通过任取 $|V|-1$ 个对应集合不含 $v$ 的线性无关的 $\boldsymbol{y}$ 取到。例如，所有对应集合大小为 1 且不等于 $\{v\}$ 的向量 $\boldsymbol{y}$ 构成一组基；以 $v$ 为根任取一棵 *dfs* 树，所有除根节点外的节点为根的子树对应的节点集合构成一组基。

**定义 6.4** (图的环)**.** 对于一张图 $G$，称其一个边子集为其的环当且仅当在图的每个节点都与其中的偶数条边相连。

即，$\boldsymbol{x}$ 为图的环集当且仅当 $\boldsymbol{Mx}=\boldsymbol{0}$。

**定理 6.3** (环空间)**.** 连通图的环集构成了一个线性空间，其秩为 $|E|-|V|+1$。为了得到它的一组基，任意取图的一棵生成树 $T$，则每条非生成树上的边与其对应的唯一一条简单树上路径构成了图的一个简单环，而这些简单环构成的环空间的一组基。

由于 $\boldsymbol{x}$ 为图的割集当且仅当存在向量 $\boldsymbol{y}$ 使得 $\boldsymbol{x}=\boldsymbol{M}^{\top}\boldsymbol{y}$，而 $\boldsymbol{x}$ 为图的环集当且仅当 $\boldsymbol{Mx}=\boldsymbol{0}$，运用线性代数的知识可以得到：

**定理 6.4.** 图的割空间和环空间互为正交补。

进一步，可以得到：

**定理 6.5.** 图的割空间和环空间的直和为边空间当且仅当图的生成树个数为奇数。

**定理 6.6.** 对于一个边集合 $T$，图在删去 $T$ 后不连通当且仅当存在 $\varnothing\neq S\subseteq T$ 且边集 $S$ 为图 $G$ 的割。

由于这些定理的证明与本文关系不大，故略去。

**例题 6.4** (边双连通分量)**.** 给定一张 $n$ 个点 $m$ 条边的连通简单无向图 $G$，求出其所有极大边双连通分量。

该问题等价于找到图的所有割边。

任取图的一棵 dfs 树，则仅有其中的树边可能成为图 $G$ 的割边。其将树划分为了子树内和子树外两个部分，则易知删去这条边后图连通当且仅当不存在一条返祖边经过这条边。

类似前缀和与差分的思想，对每一个节点维护集合 $S$ 将这条返祖的两个端点的 $S$ 与仅包含这条边的集合做对称差，并自下而上算出每个节点子树内所有集合的对称差。此时，若树边较深的节点为 $v$，则删去该树边后图非连通当且仅当 $S_v=\varnothing$。

使用上文的哈希方式进行维护即可，总时间复杂度为 $O(n+m)$。

同时，本问题也存在时间复杂度为 $O(n+m)$ 的确定性算法（常见算法为 Tarjan 算法）。

由于图的割空间和环空间互为正交补，因此可以使用线性代数的知识得到以下结论：

**定理 6.7.** 任取图的一棵生成树 $T$，并对每一条边维护集合 $S$，对于每一条非树边，将两端点其在 $T$ 上的唯一简单路径上的每条边的集合 $S$ 中插入这条边，同时将这条非树边也插入到自己的集合 $S$ 中（即在环上的每一条边上插入这条树边），则一个边集为图的割集当且仅当它们对应的集合的对称差为空集。

证明考虑每次将一个元素插入到一个简单环上而所有这样的简单环构成了一组环空间的基，而割空间和环空间互为正交补。

容易使用该定理直接得到以下两个经典问题的解法：

**例题 6.5** (切割[8])**.** 给定一张 $n$ 个点 $m$ 条边的连通简单无向图 $G$，进行 $k$ 次相互独立的询问，每次询问给定图中的 $c_i$ 条边，请判断若删去这些边，操作后的图是否连通。

保证 $n, m, k \le 10^6, c_i \le 4$。

任取生成树 $T$，并使用上文所述方法得出每条边对应的集合 $S$，则只需判断是否存在一个给定的 $c_i$ 条边的非空子集使得其对应的 $S$ 的对称差为空集。

使用随机赋权哈希即可做到 $O(n+m+k2^{\max\{c_i\}})$ 的时间复杂度，若取 $w = 64$ 位的无符号整数则由 Union Bound 可以得到单次询问错误率不超过 $2^{c_i-w}$。

**例题 6.6** (边三连通分量)**.** 给定一张 $n$ 个点 $m$ 条边的连通简单无向图，求出其所有极大边三连通分量。

可以先使用 Tarjan 算法将图分解为若干个边双连通分量，再对每一个边双连通子图分别考虑它们的所有割集。

由于只需考虑大小为 2 的割集，因此只需要找到所有哈希值相同的两个集合对即可，而这是易于处理的。

由于删去两条边后得到的两个连通块可以用至多两个子树的对称差来表示，使用一些分类讨论即可处理每一个哈希值对应的边等价类对图的边三连通分量造成的划分影响，总时间复杂度为 $O(n+m)$。

事实上，截止到目前为止，对于任意 $2 \le k \le 5$，学术界都存在确定性的求解 $k-$ 边连通分量的时间复杂度为线性的算法。

**例题 6.7** (Koosaga's Problem[9])**.** 给定连通简单无向图 $G$，其中 $|V| = N, |E| = M$，求出 $S \subseteq E$ 同时满足以下三个条件：

- 图 $(V, E-S)$ 为二分图;
- $|S| \le 2$；
- 不存在 $T \subseteq E$ 使得 $|T| < |S|$ 且 $T$ 符合以上两个条件。

保证 $N, M \le 2.5 \times 10^5$。

在下文中均在边空间下进行讨论，记图 $(V, E+S)$ 表示边集不变，但是将边权设置为 $E$ 和 $S$ 对应分量的异或和（前者总为 1）。

[8] 来源：GDKOI 2024 普及组第一天第四题

[9] 来源：Problem B, Petrozavodsk Summer 2020. Day 6. Korean Contest

**定理 6.8.** 图 $G$ 删掉 $S$ 后的边为二分图的充分必要条件为存在 $S$ 的子集 $T$ 使得 $G' = (V, E + S)$ 为二分图（不存在奇环）。

证明. 由于该操作将删边更改为了修改边权，因此充分性显然。

考虑证明必要性，则只需证明对于任意极小满足条件的 $S$ 均有 $(V, E+S)$ 为二分图即可。

由于删边数量是极少的，因此每条被删除的边在删除前都被包含在一个奇环内。故对于图中的任意一条被删除的边，其两侧都能通过环的另一侧相连。

由于是奇环，因此删边后的距离为偶数，这等价于两个点之间存在一条边权为 0 的边，故再加入一条边权为 0 的边不影响图为二分图这一事实，即 $(V, E + S)$ 为二分图。

□

故可以考虑类似前文中 6.7 中的方法，仍然求出每条边对应的集合 $S$，则此时由于每一个环都要是偶环，故条件应修改为选取的边集对应的集合的对称差与所有集合的对称差相等。

由于只需考虑 $|S| \le 2$ 的方法，因此可以使用上文所述的随机赋权哈希算法后使用哈希表进行维护，总时间复杂度为 $O(N + M)$。

## 6.3 当 $p$ 为 2 以外的质数时的应用

在上文中，笔者详细讨论了 Laurent 多项式哈希在 $p = 2$ 时的应用，下面笔者将会给出一些 $p$ 为其它数时的例子。

**例题 6.8** (子区间计数[10])**.** 给定一个长度为 $n$ 的正整数序列 $a$，给定 $q$ 次询问，每次询问给定区间 $[l, r]$，求其有多少个子区间的乘积为完全立方数。

保证 $n, q \le 2 \times 10^5, \max\{a_i\} \le 10^7$。

使用类似上文平方数（弱化版）一题的做法，将数 $x$ 的素因子存入集合 $S$ 上，并将出现次数对 3 取模。得到每一个 $a_i$ 对应的 $S(a_i)$ 的哈希值并在域上做前缀和后，所求即为区间同色点对数量，可以使用莫队算法维护，总时间复杂度为 $O\left(\max\{v_i\} + n \max\{\omega(v_i)\} + n\sqrt{q}\right)$。

# 7 随机数赋权法解决一般情况下模意义下的 Laurent 多项式哈希

这一部分将推广上文中的 $p$ 为质数的情况，将其扩展为 $p$ 为任意大于等于 2 的整数时的情形。由于字母 $p$ 在数学中常常用于表示质数，因此在下文中笔者将换作使用字母 $w$ 表示。

[10]本题为原创题

## 7.1 当 $w$ 为质数幂次时的 Laurent 多项式哈希

记 $w = p^k$，类似上文，构造随机哈希函数 $H : \mathbb{Z} \to \mathbb{Z}_{w^t}$，并将一个 Laurent 多项式 $F(x)$ 的哈希值定义为 $\sum f_i H(i)$，此处求和为群中的 $\oplus_w$ 运算。类似上文的正确率分析，可以得到以下结论：

**定理 7.1.** 对于模 $w = p^k$ 下不同的 *Laurent* 多项式 $F(x)$ 和 $G(x)$，用上述方式计算的哈希值相等的概率相等的概率不超过 $\frac{1}{p^{k^t}}$。

由于该命题为以上所有与随机赋权有关的正确性命题的加强，因此在这里给出它的证明：

证明. 记非零 Laurent 级数 $D(x) = F(x) - G(x)$，则 $F(x), G(x)$ 哈希碰撞的概率即为 $D(x)$ 哈希值为 0 的概率。

由于群 $(\mathbb{Z}_{w^t}, \oplus_w) \cong \mathbb{Z}_w^t$，因此只需估计单个分量碰撞的概率再利用独立性得到联合概率的上界。

由于 $D(x)$ 非零，因此存在某个 $i$ 满足 $d_i \neq 0$，记 $S = \sum\limits_{j \neq i} d_i H(i)$，由全概率公式，$D(x)$ 的某一分量为 0 的概率即为

$$\sum_{j=0}^{w-1} \Pr\left(S \equiv -j \pmod w\right) \Pr\left(d_i H(i) \equiv j \pmod w\right),$$

由于 $\sum\limits_{j=0}^{w-1} \Pr\left(S \equiv -j \pmod w\right) = 1$，因此其不超过 $\max\limits_{j=0}^{w-1} \{\Pr(d_i H(i) \equiv j \pmod w)\}$。

根据基础数论的知识，当 $d_i, w$ 固定时，上式右侧为 $\frac{\gcd(d_i, w)}{w}$，显然其最小值为 $\frac{1}{p}$，当 $d_i = p$ 时取到。

□

## 7.2 当 $w$ 为任意模数时的 Laurent 多项式哈希

使用中国剩余定理，记 $w$ 的唯一分解形式为 $p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$，则 $x \equiv 0 \pmod w$ 当且仅当对于任意 $1 \leq i \leq m, x \equiv 0 \pmod{p_i^{k_i}}$。

维护 $m$ 个上文所述的哈希值即可，由于一些原因，该算法在信息学竞赛中应用较少。

## 7.3 Nimber 域下模 2 下的幂次哈希法

在本节下文中，为避免可逆性等问题，不妨设 Laurent 多项式仅存在非负次数项，即其退化为普通多项式。

有时，需要支持一个集合上的数字同时加上一个数的操作，或是将多项式相乘，此时就需要将上文所述的群扩展为一个域，并仿照上文所述方法任取一元素 $B$ 并将一个多项式

的哈希值定义为 $\sum_i f_i B^i$，其中加乘操作均在域中完成（但是由于 $f_i$ 是整数，因此在计算 $f_i B^i$ 中 $f_i$ 乘上 $B^i$ 一步时仍然看作是整数乘域中元素）。

由于域的特征必然是质数，因此先考虑最简单的 $p = 2$ 的情形，此时可以取 Nimber 积运算 $\otimes$，对于任意非负整数 $k$，$(\mathbb{Z}_{2^{2^k}}, \oplus, \otimes)$ 构成了一个域。

求解一次 $x \otimes y$ 的复杂度是 $O(2^{2k})$，但是实际应用（$k = 6$）时可以实现至效率近似 $O(1)$。

此时任取一个 Nimber 域中的元素作为 $B$，经过试验可发现该哈希方法具有很高的正确率。

## 7.4 矩阵环下模 2 下的幂次哈希法

由于并不需要域中乘法具有交换律以及逆元存在性，因此可以使用环来代替。具体而言，另一种处理 $p = 2$ 的情况的方式是使用矩阵乘法，取正整数 $k$ 以及一随机 $k \times k$ 的矩阵 $M$，将矩阵群扩展为矩阵环 $(M_k(\mathbb{F}_2), +, \times)$ 并在其下计算哈希函数。

考虑分析该算法的正确率，仍然类似前文考虑某一非零多项式 $G$ 的哈希值为 0 的概率，设随机矩阵 $M$ 的最小多项式为 $Q(x)$，则 $G$ 的哈希值为 0 当且仅当 $Q(x) \mid F(x)$，由于 $Q(x)$ 的次数接近 $n$，因此可以大致感受到 $Q(x) \mid F(x)$ 概率不高，因此取较大的 $k$ 即可保证很高的正确率。

## 7.5 基于其它代数结构的模 2 意义下的幂次哈希法

在研究该问题时，笔者还得到了很多哈希方式，例如将矩阵乘法 $C = A \times B$ 的定义修改为类似二元多项式乘法的形式 $C_{i,j} = \sum_{i_0+i_1 \equiv i \pmod p} \sum_{j_0+j_1 \equiv j \pmod p} A_{i_0,j_0} B_{i_1,j_1}$，则其具有交换律，即可得到交换环的结构。事实上类似这样的哈希方式还可以得到很多。

根据上述哈希方法，并注意到矩阵环 $(M_k(\mathbb{F}_2), +, \times)$ 上的哈希方法实质上是用到了零化多项式的随机性，因此可以将总结出一个方法找到一个较为通用的用于哈希的域，一种方法是找一个既约多项式 $p(x) \in \mathbb{F}_2[x]$（由于多项式乘法的交换律以及结合律，因此其带有天然的域结构），然后将 $\mathbb{F}_2$ 下的多项式环（这也是一个欧几里得整环）上的每一个元素（多项式）对 $p(x)$ 取模得到最终的代数结构。

换而言之，这引出到了域 $\mathbb{F}_2[x]/p(x)$ 的构造，其中 $p(x)$ 为一 $\mathbb{F}_2$ 下的既约多项式，例如取 $p(x) = x^{63} + x + 1$，它的好处是其可以使用一个 64 位无符号整数表示，并且通过指令集以及位运算在常数时间复杂度内计算，一种基于 C++ 的实现方法如下：

```
ull mult(u64 x, u64 y) {
    __m128i a = _mm_set_epi64x(0, x), b = _mm_set_epi64x(0, y);
    __m128i c = _mm_clmulepi64_si128(a, b, 0x00);
    x = c[0];
    y = c[1];
```

```
    return (y << 1) ^ (y << 2) ^ x ^ (9223372036854775811ull & -(x >> 63));
}
```

任取随机 $\mathbb{F}_2[x]/p(x)$ 中的一随机多项式为底数 $B$，取哈希函数为 $\sum_i f_i B^i \bmod p(x)$。在实际测试中，该哈希方式具有极高的正确率。

## 7.6 一般情况下模意义下的幂次哈希法

经过实验，在给定模数 $p$ 为质数时，上述很多算法都能够沿用且具有较高的正确率；进一步的，给定模数 $p$ 为质数的幂次时，上述算法也往往具有良好的表现。

具体而言，关于 Nimber 域能否沿用的问题笔者尚未得到定论，在矩阵环（$(M_k(\mathbb{F}_2),+,\times)$）可直接修改模数（即修改为环 $(M_k(\mathbb{Z}_p),+,\times)$），而域 $\mathbb{F}_2/p(x)$ 可修改为环 $\mathbb{Z}_p/p(x)$，其中 $p(x)$ 为任一既约多项式。

可以证明，对于任意质数幂次 $m$，$(\mathbb{Z}/m\mathbb{Z})[x]$ 中存在任意大次数的既约多项式，在其中选一较随机的多项式作为底数进行哈希即可。

因此，所有 $p$ 为质数幂次的情况已被解决，其余情况只需使用中国剩余定理化归为此类情形即可。

## 7.7 以及带有多个分量的组合对象的哈希方法以及区间询问

作为本节的收尾，在此介绍多个分量的组合对象的哈希方法，例如 Laurent 多项式中两个多项式的和为它们的按位异或。

在上文，探讨了 $\mathbb{Z}_p[x]$ 下的多项式哈希方法，接下来笔者将会讨论群 $\mathbb{Z}_p[x]^k$ 下的多项式哈希方法。

最为简单的方式是对于 $k$ 个分量分别计算哈希函数并做有序哈希值合并。但是，在矩阵环哈希法中，若 $k$ 小于选定的矩阵大小，可以 $f_i$ 对应的这 $k$ 个分量写入一个列向量中（多余位置补 0）并计算 $\sum_i f_i B^i$，这样可以提高整体的运行效率和正确率。

同理，在多项式域下进行哈希的时候，若 $k < \deg p$，也可以改为计算 $\sum_i \left(\sum_{j=0}^{k-1} f_{i,j} x^j\right) B^i$，其中 $f_{i,j}(0 \le j < k)$ 表示 $f_i$ 的第 $j$ 个分量。

最后，在应用中有时会遇到这样的问题：给定一个序列 $a$，求出其一段区间构成的（可重）集合对应的哈希函数，此时只需根据实际需要选择哈希方法，并求出每个 $a_i$ 对应的哈希值 $w_i$ 并对其求出前缀和 $S_i = \sum_{j=1}^{i} w_j$ 即可，并在查询区间 $[l,r]$ 时返回 $S_r - S_{l-1}$ 即可。由于该部分内容较为基础，在此不再赘述。

# 8 有序列表的哈希方式

本节中暂且先考虑其中的有序列表的元素都是整数的情形，并且设有序列列表的下标从 0 开始，因此下文中可简称为数列。

在上文中，读者可能已经注意到了，上文所属的幂次赋权哈希方式与竞赛中常见的字符串哈希方式类似，这是因为多项式和有序列表存在对应关系。

具体而言，由于多项式 $F(x)$ 的项数有限，因此其存在最高次项，可以将其按照次数从 0 开始提取所有项的系数 $f_i$；同理，可以根据一个有序列表反推出这样的一个多项式。

同时，无序列表的哈希也可以通过对元素排序转化为有序列表的哈希，因此可以看出掌握有序列表的哈希方式在信息学竞赛中是非常重要的。

## 8.1 有序列表的哈希方式

对于一个数列 $a$，可以得到一种哈希方式为取哈希值 $\sum_{i=0}^{|a|-1} f_i H(i)$ 进行哈希，其中 $H:\mathbb{Z}\to\mathbb{Z}_p$ 为一独立均匀随机映射（$p$ 为一奇质数）。

同样的，另一种哈希方式为取哈希值 $\sum_{i=0}^{|a|-1} f_{|a|-i-1}B^i$（值得注意的是，为了便于处理区间询问，此处的前缀和顺序与上文所述不同），这两种方法分别为上述随机数赋权法和幂次赋权法的迁移。

上述方法中随机赋权法较幂次赋权法的优势在于其简洁性且正确性度易于证明，而后者的优势为其支持更强大的功能（如给定序列多次区间询问哈希值，序列的拼接）等。

需要注意的是，由于有序列表和无序列表都可以唯一对应到一个（Laurent）多项式上，因此上述所有有关（可重）集合的算法（即对 Laurent 多项式计算哈希函数的方法）均可以迁移至此（包括之前提及的 $\mathbb{Z}_p$ 下的哈希），接下来举几个例子详细说明。

**例题 8.1** (XOR Sum of Arrays[11])**.** 给定长度为 $N$ 的数列 $A$ 并进行 $Q$ 次询问，每次询问给定三个区间 $[a,b],[c,d],[e,f]$，保证 $b-a=d-c$，构造数列 $B_{b-a+1}$（下标从 0 开始计算），其中 $B_i=A_{a+i}\oplus A_{c+i}(0\le i\le b-a)$，问 $B$ 的字典序大小是否严格小于 $A$ 的下标 $[e,f]$ 之间的元素构成的子串。

保证 $N\le 5\times 10^5, 0\le a_i\le 10^{18}, Q\le 5\times 10^4$。

考虑仅仅判断是否对于三个子区间 $R_1,R_2,R_3$，是否有 $\forall i, R_{1,i}\oplus R_{2,i}=\oplus R_{3,i}$。

对于序列 $a$ 构造哈希函数 $F(a)=\sum_{i=0}^{|a|-1} a_{|a|-i-1}B^i$，由于此处的代数结构为 $\mathbb{Z}_2^k$，故可在上文所介绍的 $\mathbb{F}_2[x]/p(x)$ 多项式域中进行计算。

取 $p(x)=x^{63}+x+1$，并每一个 $A_i$ 构造多项式 $w_i=\sum_{j=0}^{62} A_{i,j}x^j$，其中 $A_{i,j}=1$ 代表 $A_i$ 的二进制表达下第 $j$ 位为 1。

[11] 来源：Problem Ex, AtCoder Beginner Contest 274

为了处理区间询问，因此预处理前缀和 $S_i = \sum_{j=0}^{i} w_{i-j}B^j$，则区间 $[l,r]$ 的哈希值为 $S_r - S_{l-1}B^{r-l+1}$。

回到原题，只需计算出异或后第一个不同的位置，使用二分即可做到 $O((n+q\log n)T)$，其中 $T$ 为域中一次乘法计算的时间复杂度。

这种哈希方式的经典方式为字符串哈希，因此给出若干与字符串有关的问题：

**例题 8.2** (最长公共子串)**.** 求给定两个字符串 $S,T$，求出它们的最长公共子串。

保证 $|S|,|T|,|\Sigma| \le 5\times 10^5$。

对于该问题，显然答案具有可二分性——即，若 $S,T$ 具有长度为 $x(x\ge 2)$ 的公共子串，则它们具有长度为 $x-1$ 的公共子串。

考虑判断 $S,T$ 是否具有长度为 $x(x\ge 2)$ 的公共子串，计算出 $S,T$ 所有长度为 $x$ 的子串的哈希值，并使用哈希表维护即可，时间复杂度 $O((|S|+|T|)\log\min(|S|,|T|))$。

**例题 8.3** (最长公共前缀)**.** 求给定字符串 $S$，$q$ 次询问 $S$ 的两个子串的最长公共前缀。

保证 $|S|,q,|\Sigma| \le 5\times 10^5$（下文视 $|S|,q$ 同阶）。

**例题 8.4** (最长子回文串)**.** 求给定字符串 $S$，对每个位置求出以其为回文中心的最长公共子串。

保证 $|S|,|\Sigma| \le 5\times 10^5$。

上述两个经典问题的经典做法分别为后缀数组以及 Manacher 算法，但它们都可以使用该算法在 $O(|S|\log|S|)$ 的时间复杂度内解决。

**例题 8.5** (后缀排序)**.** 求给定字符串 $S$，求其 $|S|$ 个后缀的字典序顺序。

保证 $|S|,|\Sigma| \le 10^5$。

该经典问题的做法为时间复杂度为 $O(|S|\log|S|)$ 的倍增法和时间复杂度为 $O(|S|)$ 的 SA-IS 算法，使用归并排序与二分哈希判断两个后缀的字典序即可得到易于实现的 $O(|S|\log^2|S|)$ 的后缀排序算法。

可以看出，随机赋权法的应用少之又少，在此仅举一例：

**例题 8.6** (字符串编辑[12])**.** 给定 $n$ 个长度为 $m$ 的字符串 $S_i$，进行 $q$ 次修改，每次修改其中一个字符串的一个字符，每次修改后输出每个字符串在 $n$ 个字符串中的出现次数的最大值。

保证 $n,m\le 2000, q,|\Sigma| \le 2\times 10^7$。

[12]来源：本题为原创题

将数列的每一个位置赋上随机权值 $H:\mathbb{Z}\to\mathbb{Z}_{\text{比较大}}$，构造哈希函数 $f(S_i)=\sum_{j=0}^{m-1}S_{i,j}H(j)$，则可以 $O(1)$ 处理单次修改对于哈希值的影响，使用哈希表维护每个哈希值的出现次数。

则问题转化为了有一个数列，每次操作会将其中一个元素加上 1，另一个元素减去 1，需要快速维护最大值。

使用桶维护，其中下标为出现次数，值为该出现次数对应的元素个数，每次修改后最大出现次数的变化量是 $O(1)$ 的。

综上所述，总时间复杂度为 $O(nm+q)$，正确率为 $1-\frac{1}{2^{64}}$。

在不带修改的哈希问题中，几乎没有随机赋权法的应用。例如在上例中，若没有修改，可以使用 Trie 树与哈希表直接维护每个字符串等价类的大小。

## 8.2 再谈另一种等价关系下的哈希

回顾上文中集合哈希一节提到的问题：给定有序序列 $a,b$，判断是否存在某组合类到自身的双射 $f$ 使得 $f(a)=b$。笔者先前已经介绍了“集合套集合”的方法来解决判断是否存在这样的 $f$，在下文中笔者再次对该问题进行探讨。

有结论：对于给定的序列 $a$，记 $p_i$ 为 $a_i$ 之前第一个与 $a_i$ 相同的元素与 $a_i$ 之间的距离，若不存在则为 0，则序列 $a,b$ 在上述意义下相同当且仅当它们对应的 $p$ 相等。

发现，可以对于每一种值使用数据结构维护它的每个出现位置，因此其支持对字符串的修改（仅有至多 3 个 $p_i$ 需要进行修改）。

**例题 8.7** (最长同类子串[13])**.** 对于两个等长字符串 $A,B$，若对于任意 $1\le i,j\le|A|$，$A_i=A_j\Leftrightarrow B_i=B_j$，则称 $A,B$ 是一对同类串。

给定两个字符串 $S,T$，求它们的最长公共同类串。

保证 $|S|,|T|\le 10^5,|\Sigma|=26$。

容易发现最长同类子串的定义等价于上文所述的定义。

仍然使用上文“最长公共子串”问题的方式二分答案，则只需计算出 $S,T$ 中所有长度为 $k$ 的子串的哈希值即可。

先考虑 $S$ 一侧的计算方式，$T$ 一侧是同理的。

维护字符串 $S'$，初始时为 $S$ 的长度为 $k$ 的前缀，每次移动时在末尾假如一个字符并在开头删去一个字符。

使用幂次赋权法，对每种字符开一个队列维护所有出现位置，则每次修改只会进行至多两次队列上的修改，因此只会影响不超过常数个 $p_i$。

需要注意的是，由于操作完后第 $2\sim k$ 个字符的权值会发生变化，因此需要乘除上哈希的底数 $B$，而新字符对哈希值的更改量是好维护的。

综上所述，该问题在 $O((|S|+|T|)\log\min(|S|,|T|))$ 的时间复杂度内解决。

[13]来源：2023 年蓝桥杯国赛 Python A 类

**例题 8.8** (字符串[14])**.** 称两个字符串 $a, b$ 本质相同当且仅当其等长且存在一一映射 $p : \Sigma \to \Sigma$ 使得对于任意 $i$ 都有 $p(a_i) = b_i$。

求给定长度为 $n$ 的字符串 $S$ 的本质不同子串个数。

保证 $|S| \le 10^5, |\Sigma| = 26$。

本质不同子串数量的传统计算方法为后缀排序并求出相邻字典序的后缀之间的最长公共前缀后用 $\frac{n(n+1)}{2}$ 减去它们的和后输出。

本题中即将每个后缀替换为后缀字符串对应的 $p$ 数列，因此也可以对于这些 $p$ 按字典序排序后用 $\frac{n(n+1)}{2}$ 减去相邻元素两两之间的最长公共子串和求出本质不同的 $p$ 的前缀的数量（显然此即为答案）。

先求出整个字符串 $S$ 对应的 $p(S)$，则它的一个后缀对应的 $p$ 可由 $p(S)$ 修改至多 $|\Sigma|$ 个位置（每种字符在该后缀中的第一次出现）的 $p_i$ 得到。

进行 $O(n \log n)$ 的后缀排序 $p(S)$，使得可以 $O(1)$ 比较两个子串的大小关系。

此时使用归并排序对所有后缀进行排序，而在一次比较中，两个后缀的 $p$ 总共只有至多 $2|\Sigma|$ 个位置与原先的 $p(S)$ 有差别。因此，可以将每个后缀对应的至多 $|\Sigma|$ 个位置归并后依次从前到后比较每个特殊位置和每一段（这一段的 $p$ 此时都对应了 $p(S)$ 的一个子串）的字典序关系，找到第一个位置返回即可，这样单次比较的时间复杂度是 $O(|\Sigma|)$ 的。

求解两个后缀对应的 $p$ 的最长公共前缀也是同理的，因此本题在 $O(n \log n|\Sigma|)$ 的时间复杂度内得解。

还要指出，$p_i$ 的定义不仅可以局限在相等关系上，还可以体现在不等关系上，下举一例。

**例题 8.9** (Cartesian Trees[15])**.** 给定一个长度为 $N$ 的排列 $A$，并给定它的 $Q$ 个区间 $[L, R]$，将它们的小根笛卡尔树建出，问其中本质不同的有根树的数量（区分左右儿子顺序）。

保证 $N, Q \le 4 \times 10^5$。

可以证明，记 $p_i$ 为 $A_i$ 之前第一个小于 $A_i$ 的元素与 $A_i$ 之间的距离，若不存在则为 0，则两个序列对应的笛卡尔树相等当且仅当它们的 $p_i$ 相同。具体证明可以考虑笛卡尔树的构造过程，在此略去。

同时，对于一个元素 $a_i \in [L, R]$ 以及对应的 $p_i$，在 $L > p_i$ 时其对哈希函数的贡献应修改为 0。

这启示着使用扫描线从大到小扫描每一个 $L$，同时维护序列 $w$ 表示对哈希函数的真实贡献，初始时将所有 $w_i$ 赋为 0。在考虑的左端点 $L$ 减小时，将所有 $p_i$ 等于当前 $L$ 的位置上的 $w_i$ 修改为 $p_i$，并使用线段树维护 $w_i$ 的区间哈希值即可。

由于区间的左端点在原数列上会改变，因此此处只能使用随机赋权法，总时间复杂度为 $O((N + Q) \log N)$。

[14] 来源：多校联考

[15] 来源：Problem M, Tokyo Tech Programming Contest 2024 (Div. 1)

## 8.3 基于哈希的等价类递归划分算法

在上文中，笔者使用哈希算法将后缀排序这一经典问题做到了 $O(n\log^2 n)$ 的时间复杂度（其中 $n=|S|$ 为串长），事实上，该算法具有改进的空间。

下文中，为方便起见，将在字符串 $|S|$ 的末尾插入 $n-1$ 个字符 $\epsilon$，其字典序小于其它所有字符，并比较其所有长度为 $n$ 的串的子串的字典序。

为了比较若干个长度相同的串的字典序，可以取 $m=\left\lfloor\frac{n}{2}\right\rfloor$ 并依次对两侧进行比较。

取每个字符串的前 $m$ 个字符，使用字符串哈希方式可以将所有字符串依照这 $m$ 位划分为若干个等价类，则等价类之间互不干扰，因此可以对于每个等价类递归后半部分的 $n-m$ 个字符继续计算。

为了拼接不同的等价类，需要得到它们之间的顺序。故考虑对于每一个等价类任选一个代表元，将所有的代表元按照前 $m$ 个字符的字典序进行排序即可。

当递归到 $n=1$ 的情况时直接处理即可。

综上所述，已可以在 $O(n\log n)$ 的时间复杂度内解决后缀排序问题，这与信息学竞赛中常使用的的倍增方法复杂度相同，但常数略大。

该等价类划分算法在信息学竞赛中应用广泛[16]，碍于篇幅在此不多展开。

# 9 循环同构下的哈希方法

两个长度为 $p$ 的有序序列循环同构本质上相当于多项式关于 $x^p-1$ 取模，也即判断是否存在 $k$ 使得 $F(x)x^k\equiv G(x)\pmod{x^p-1}$，下文中仍然假设多项式中的元素均为非负整数。

## 9.1 基于有序序列哈希的方法

一种自然的相同是将每种序列划分到一个等价类中（因为循环同构构成一个等价关系），使得两个序列循环同构当且仅当它们所在等价类相同。

为了实现将一类序列唯一对应到一个等价类这一过程，一个很自然的想法是在每一类中取代表元，例如其最小表示（即所有循环中对应的字典序最小的序列）。

也即，可以通过判断两个序列的最小表示是否相同来判断这两个序列是否循环同构。

更进一步的，可以使用上文所述方法，通过判断两个长度同为 $m$ 的序列的最小表示的 $\sum_{i=0}^{m-1} f_{m-i-1}B^i$ 是否相同来判断两个序列是否循环同构，这样可以减少储存的信息量。

在需要对单个序列进行多次区间计算哈希值时，一种方法是使用经典方法[17]一题中的方法找到最小循环同构对应的循环移位数，并依次算出移位后的字符串的哈希值即可。

具体方法由于与本文关系不大，因此不再展开。

[16] 如 NOI 2025 第一天第三题，数字树

[17] 例如类似 Baby's First Suffix Array Problem (Problem B), The 2020 ICPC Asia Nanjing Regional Contest

## 9.2 基于对多项式取模的方法

考虑以下形式的问题：

**例题 9.1** (循环同构[18]). 初始时有一个长度为 $n$ 的向量 $a_1 = (1, 0, 0, \cdots, 0)$，接下来进行 $q$ 次操作，每次操作为以下三种字母之一：

- 在向量序列 $a$ 末尾插入 $a$ 中两个某两个向量的和；
- 在向量序列 $a$ 末尾插入 $a$ 中某个向量循环右移位一次后的结果；
- 询问向量中的某两个向量是否循环同构。

保证 $3 \le n \le 10^6, q \le 10^5$。

该问题显然不能使用上文所述方法解决，时间复杂度会导致超时。

先考虑一个更为简单的问题：判断一个向量是否为零向量。

由于序列是循环序列，因此可以将其视作多项式（设长度为 $n$ 的序列 $a$ 对应的多项式为 $\sum_{i=0}^{n} a_i x^i$），但是在模 $x^n - 1$ 意义下进行运算。

此时，循环右移位一次就相当于乘上 $x$，而两个序列相加就相当于两个多项式相加，最后只需判断一个多项式是否为 0。

由于计算在模意义下进行，故可以取满足 $x^n - 1 = 0$ 的 $x$，即 $x$ 为全部 $n$ 次单位根，此时 $x^n = 1$，故可以实现取模操作。

一种方法是只代入 $x = \omega_n$ 并判断 $f(\omega_n)$ 是否为 0，但是这样明显是错误的，例如 $f(x) = \sum_{i=0}^{n-1} x^i$ 时会误判为 $f(x) = 0$。

这是因为，满足存在因式 $(x - \omega_n)$ 的整系数多项式 $f$ 不一定是 $x^n - 1$ 的倍数，考虑增加代入的特殊值的数量进行校验。

一种正确的方式是对于 $n$ 的每个约数 $d$，将 $x = \omega_d$ 也代入检验，这样检验的数量个数为 $d(n)$（这表示 $n$ 的约数个数）。

其正确性证明包含与分圆多项式理论（$x^n - 1 = \prod_{d|n} \Phi_d(x)$，其中 $\Phi_d(x)$ 为关于 $x$ 的 $d$ 次分圆多项式）相关的繁杂数学推导，在此不再展开[19]。

由于不能直接存储单位根，所以可以取任意 $p = kn + 1$ 型的质数，并取其某原根 $g$ 的 $k$ 次方作为 $\omega_n$ 进行接下来的数值计算（在模 $p$ 意义下），其正确性类似于快速傅里叶变换与快速数论变换之间的关系。

对于本题，由于需要判断两个向量是否循环同构，因此可以判断是否有循环移位数 $k$ 使得第一个向量乘上 $x^k$ 等于第二个向量，这可以通过计算乘法逆元进行判断，总时间复杂度为 $O(qd(n) \log p)$。

[18] 来源：梦熊 NOI 2025 苍穹计划模拟赛

[19] 可参见 https://zhuanlan.zhihu.com/p/538456549

# 10 树哈希

前文中提到的一切结构都基于线性结构（无序或有序或循环同构有序），并且对于不同的系数对应的代数结构进行了探讨，在下文中笔者将对于树这一常见的结构进行分析。

## 10.1 判断有根树同构的 AHU 算法

类似上文中所提到循环同构哈希的最小化方式，在树哈希中也有类似的结论，对于两棵有根树 $T_1$ 和 $T_2$，有如下定理：

**定义 10.1.** 称点序列 $(v_0, v_1, \cdots, v_k)$ 是一个有根树 $T$（根为 $r$）的欧拉序当且仅当：

- $v_0 = v_k = r$；
- 对于任意 $0 \le i < k$，$(v_i, v_{i+1})$ 之间有边相连；
- 对于任意树边 $(x, y)$，$(x, y)$ 和 $(y, x)$ 都恰好在上一条中作为 $(v_i, v_{i+1})$ 出现一次。

**定义 10.2.** 若 $v_k$ 是有根树 $T$ 的一个欧拉序，称其对应的括号序列为一个长度为 $k$ 的序列 $S_k$，其定义如下：

- 若 $v_k$ 是 $v_{k+1}$ 的父节点，则 $S_k$ 为左括号；
- 若 $v_k$ 是 $v_{k+1}$ 的子节点，则 $S_k$ 为右括号。

**定义 10.3.** 称一棵有根树的最小括号表示为其所有欧拉序对应的括号序列中字典序最小的一个（左括号的字典序小于右括号）。

**定理 10.1.** 有根树 $T_1, T_2$ 同构的充要条件为它们对应的最小括号表示相等。

该定理的证明是显然的。

同时，若记 $S_x$ 表示树 $T$ 以 $x$ 为根节点的子树对应的最小字典序，则可以证明有 $S_x$ 可以由所有 $S_v$（其中 $v$ 为 $x$ 子节点）在外侧嵌套一对括号后排序拼贴得到。

因此，为了计算出每一个节点对应的哈希值（最小括号表示），可以通过计算其所有子节点的哈希值合并得到。由于字符集大小为 2，因此可以将所有字符串插入到 Trie 树上并在 Trie 上 dfs 得出所有串之间的相对大小关系（事实上，此处的排序相当于将儿子的无序序列转化为有序序列）。

对于单棵树 $T$，该算法的时间复杂度为 $O(|V|^2)$。

## 10.2 有根树哈希的哈希算法

在信息学竞赛中，为了加速计算，常常使用和前文“集合套集合”一般的“哈希套哈希”的方式来求出所有子树的哈希值，也即用一个整数来概括一棵子树的信息。

事实上，这样的哈希函数很好构造，其相当于对于子树的哈希值构成的无序列表做哈希，在此，笔者介绍一个较为常见的哈希方式：

仍然自下而上确定哈希值，并将节点 $x$ 的哈希值 $f_x$ 定义为 $f_x = 1 + \sum_{v\in \mathrm{son}(x)} H(f_v)$，其中 $\mathrm{son}(x)$ 表示 $x$ 的所有子节点构成的集合。

可以证明，在 $H$ 为一 $\mathbb{Z}_p \to \mathbb{Z}_p$ 的独立均匀随机函数时，该算法的正确率至少为 $1 - \frac{|V|^2}{2p}$，其中 $p$ 为值域。

这是因为两个节点哈希值冲突等价于其子节点构成的集合哈希值冲突，而此概率根据上文 4.3 中的结论，故恰好为 $\frac{1}{p}$，使用 Union Bound 即可证明该算法的正确率。

该算法在竞赛中时有考察[20]。

但是该哈希函数的一个缺点是难以用数据结构维护，例如需要支持动态加点并询问两个子树是否同构时，例如以下问题：

**例题 10.1** (Automorphism[21])**.** 有一棵有根树，初始时只有一个节点，进行 $m$ 次操作，每次操作形如加入一个叶节点或询问某子树的自同构群大小对 998244353 取模后的结果。

保证 $m \le 3 \times 10^5$。

对于节点 $x$，考虑其所有子节点 $v_1, v_2, \cdots, v_k$，由于同构保持父子关系，因此可按照子节点之间的同构关系将它们划分为若干个等价类，则对其子节点进行置换的方案数为各等价类阶乘之积。

而对于点 $x$ 对应的子树，其答案显然为各节点进行子节点轮换的方案数，即 $\prod_{v\in S_x} f(v)$，其中 $S_x$ 为节点 $x$ 对应的子树中的所有节点。

由于同构的树大小必然相同，因此加入一个节点时，若其某个祖先节点 $u$ 的同构关系改变，则其父节点还拥有一个子树大小和 $u$ 大小相同的子树，故父节点的子树大小大于 $x$ 的子树大小的两倍。

因此，发生变化的同构关系只有 $O(\log m)$ 个，若能够快速对这些节点进行更新。则可以使用线段树维护答案，其中下标为树的 dfs 序。

但是上述哈希不好快速维护，一种方法是取哈希函数 $h_x = H(\mathrm{dep}_x)\left(1 + \prod_{v\in \mathrm{son}_x} h_v\right)$，其中 $H$ 为一随机哈希函数，$\mathrm{dep}_x$ 表示节点 $x$ 的深度，$\mathrm{son}_x$ 表示 $x$ 的子节点构成的集合。

可以证明该哈希方式在本题中可以对相同的子树得到相同的哈希值，并且冲突概率可接受。

[20] 如 NOI 2022 第二天第一题，挑战 NPC II

[21] 来源：Problem A, Petrozavodsk Summer 2018. Day 8: Yuhao Du Contest 5

该哈希函数的优点是其可以使用数据结构维护矩阵维护快速维护，但其不能迁移到其它题目中（因为本题有需要判断的子树的根节点具有相同的深度这一性质）。

综上所述，找到 $O(\log m)$ 个可能需要更新的位置后使用数据结构维护矩阵乘法从而得到哈希值并用哈希表维护每个节点的子节点对应的哈希值构成的集合即可，总时间复杂度为 $O(m \log^2 m)$。

## 10.3 无根树的哈希方法

若两棵无根树同构，则可以证明它们的重心（1 个或 2 个）在对应重标号方案中对应。

因此可以用该特性将无根树转化为有根树，具体而言，若树的重心唯一则令其为根；否则，两个重心一定相邻，在其边中间新建节点并以之为新的根。

需要注意的是，该两种情况应在两种哈希表中进行维护以避免不必要的冲突。

## 10.4 树的确定性哈希方法

先对于有根树的情况进行讨论。

考虑上文中有根树的 AHU 算法的确定性的实质的来源，可以发现是因为其最小括号表示的唯一性和代表性，也即合并方式唯一并且能够代表唯一一个等价类。为了用更为高效的复杂度实现这一过程，找到更小的信息量记录是更为关键的。

做到最优压缩时，应让该每一个范围内的整数都能对应一种树，但是由于至多只有树的大小个本质不同的子树结构，因此没有对于每种树都记录的必要。因此可以转而只将在给定树中作为子树出现的那些树进行编号，并将原树的每一棵子树都用这些编号进行等价类划分。

使用等价类划分思想而不是构造一确定的哈希函数，这一改变是复杂度降低的核心。

类似之前的方法，从下到上确定每个节点所在的等价类（即原文的哈希值）。为了顺接上文的名称以及方便，此处也将其叫做“哈希值”。

# 11 部分哈希问题的确定性算法

本节中的树哈希特指有根树哈希。

## 11.1 树哈希和线性结构哈希之间的联系

在之前所提及的线性结构哈希中，系数所在的域基本上都是整数，考虑扩展它的用处。

由于有时所关心的只有域中元素的等价类划分方式（但是此时无法支持整体加一个数等操作），因此可以将所有相同的组合类看作是一个同一个正整数。

例如，若集合中存储的元素是循环同构意义下的有序数列，则可将内层数列进行映射为一个正整数后，使得外层哈希时所有的元素均为整数。进一步的，这种组合对象之间的嵌套可以进行任意多次，并且这允许将不同种类的元素放入同一个序列中。

读者可能已经注意到了，这个过程与上文“树哈希”一节中最后提出的的方法十分相像。在“NOI 2024 集合”一题中，“集合套集合”的过程实质上就是在对于一棵深度为 2 的树进行树哈希；同时树的确定性哈希算法中也有对子节点进行排序转为有序序列的步骤。

这启示着一种集合嵌套与树结构之间的对应——事实上，一棵树天然就可以用一个集合嵌套来表示：例如一棵 7 个节点组成的完全二叉树就对应了可重集合 $\{\{\varnothing,\varnothing\},\{\varnothing,\varnothing\}\}$，这与树的递归结构是不谋而合的。

所以，可以使用树哈希的确定性算法得到其它哈希的确定性算法，而此时只有树的叶节点带权。

而由于树哈希可以支持加入删除叶节点等操作，因此集合哈希也支持对于叶结点的修改，此时需要更新其到根的路径上每个节点的哈希值。

由于部分层的结构可能是无序序列，有序序列或者循环序列（由于只关心等价性，因此可以只需要考虑元素为正整数的情况，特别的，对于子节点全部为叶节点的节点可以进行特化，由于用处较少不做介绍），因此树的定义需要发生些许改变（例如，一个点的所有子树之间可能是有顺序的），其节点也可以被划分为三种，需要分别处理。

在自底向上进行等价类划分的过程中，对于有序序列可以直接开全局 Trie 树处理，在每次更新一个点所在等价类的时候，将其子节点所在等价类按顺序合并成一个字符集为正整数的字符串。

找到对应节点后，若该字符串尚未分配编号则将其分配一个新的标号。

当为无序序列时，一种方式是对于所有元素排序转化为有序序列处理，注意此时需要另开一棵 Trie 树（对于不同类别的点应该在不同的数据结构上维护等价关系，下同）。

当节点对应为循环序列时，找到它的最小表示法，视作有序序列维护即可。

进一步的，该理论同时还支持插入节点，删除节点，更改父亲，复制节点（内容），复制节点（地址）等操作（但是这些理论只能在无序列表中使用，否则插入操作会导致位置的平移）。

前三种操作都是不难理解的，后两种操作将会在接下来进行讨论。

## 11.2 树的二叉合并理论

对于被修改节点到根的路径上的每一次节点的更新，其时间复杂度都与子节点数量有关，导致其复杂度不够优秀。为了解决该问题，笔者从线段树的二叉合并结构中获得启示，得到了树的二叉合并理论。

由于循环序列的结构较为复杂（笔者尚未得到最小表示法以外的做法）且应用较少，所以在本节的以下部分将忽略这种情形。

当节点 $x$ 为子节点有序的节点的情况，若其度数为 $\leq 2$ 则可以 $O(1)$ 直接合并（即维护 $\{\mathbb{Z}^*,\mathbb{Z}^*\}\to\mathbb{Z}^*$ 的哈希表）。

否则，在建树的时候将子节点 $v_1,v_2,\cdots,v_k(k\geq 3)$ 尽量平均分为两份 $v_{m+1},v_{m+2},\cdots,v_k$（$m=\left\lfloor\frac{k}{2}\right\rfloor$），并建立两个虚点，将该节点连向两个虚点。

此时递归处理，将两个虚点当作新的 $x$ 并用上述方法递归连向其被分配到的的子节点。

这样单次更新一个结点 $x$ 的复杂度即为 $O(\log k)$ 而非 $O(k)$ 了。需要注意的是，需要再在树节点上记录标识表示其是否为虚点，否则将会无法区分 7 个点的菊花和 7 个点的完全二叉树。

当节点 $x$ 的子节点无序的时候，对于每一个节点维护值域线段树，由于前文提到的 Laurent 多项式对应，此时即将无序序列转化为了维护值域线段树所对应的有序序列。在每次更新 $x$ 的某子节点的 $v$ 的哈希值时在 $x$ 对应的值域线段树进行修改，并对于每一个值域线段树上的节点维护对应的哈希值（所处等价类编号）即可。在该算法中，同样需要区分一个点是否是虚点。

若原非确定性算法的时间复杂度为 $T$，则该算法中的值域线段树所维护值域范围不超过 $T$，故只需取 $V=T$。故可以证明在仅更换哈希方式的处理（由随机数赋权法或幂次赋权法修改为该确定性算法）的前提下，该算法其时间复杂度不超过 $O(T\log T)$。

事实上，由该算法反推树哈希，即可得到一支持修改且单次修改复杂度为 $\tilde{O}(\max\limits_{x\in V}\{\mathrm{dep}(x)\})$ 的树哈希算法，其中 $\mathrm{dep}(x)$ 表示节点 $x$ 的深度。

综上所述，在树深度为常数的时候，即可在 $\tilde{O}(1)$ 的情况下解决任意情况的树哈希值更改。

## 11.3 该算法的应用与限制

例如，在上文所述“NOI 2024 集合”一题中，可以使用该算法得到时间复杂度为 $O(n\log n+q)$ 的确定性算法；在“Odd Mineral Resource”一题中可以得到复杂度不变（$O((n+q)\log n)$）的确定性算法（因为原算法基于可持久化线段树这一二叉合并结构）。

考虑上文所述的的复制节点（内容），复制节点（地址）的操作。后者只需让再另一个节点的出边中也加入该节点即可，由于此时一个节点可能会被多个节点指向，因此图构成了一个 DAG（有向无环图）。故在一次修改中需要修改一个子 DAG，时间复杂度分析需要视具体情况而定。

值得注意的是，这并不能解决 DAG 同构问题，原因即为该算法实质是对一个点拆分后做树哈希，而非做 DAG 哈希。

对于复制节点（内容）的操作，可以使用类似可持久化线段树的思想，仅复制根节点（此处为二叉化后的节点以保证复杂度），在修改时谨慎实现，不修改无用节点即可。

注意，与复制地址不同的是，该操作后图的形态仍然是一棵树而非 DAG。因此在二叉化的树的一棵子树上做正常可持久化数据结构所做的操作（即修改时复制节点）即可。

显然，该算法的哈希方式并不能支持将一棵子树内的元素全部加上一个数等操作，因此也可以看出该算法主要支持的是对树上节点信息的修改，此时算法自下向上重新分配每个点所在等价类来起到哈希的作用。由于信息学竞赛中没有将正确性做到完全正确的必要，所以该算法在信息学中应用较少。

# 12 图哈希

图同构是学术界的著名 NP-hard 问题，目前最优结果为 Babai 在 2015 年提出的算法，其时间复杂度为 $\exp(O((\log n)^c))$，为准多项式时间。

## 12.1 图哈希的常见算法

一个经典的错误算法是判断是否任意 $k$，图中长度为 $k$ 的环数量相同，然而这仅仅能判断图是否同谱，即邻接矩阵的特征多项式相同。

另一个常见但是正确没有保障的图哈希方法是 $k$ 维 Weisfeiler-Lehman 算法，其算法过程为初始时令每个点的权值为自身度数并在每一次操作中根据其 $k$ 级邻域进行迭代，其可被 $k$ – CFI 图定向攻击。

还有一种方式为基于迭代等价类划分的 Nauty 算法，由于篇幅限制不再展开。

总而言之，图同构的判定问题是目前学术界悬而未解的一大难题，欢迎有兴趣的同学对其研究攻克。

## 12.2 自动机最小化算法

在图为某些特殊图（例如树）的时候，可能存在确定性多项式复杂度算法。在本节中笔者介绍一种与自动机有关的哈希算法，其中自动机的定义如下：

**定义 12.1.** 一个确定有限自动机（*DFA*）是一个五元组 $(Q, \Sigma, \delta, q_0, F)$，其中：

- **有限状态集合** $Q$。如果将一个 *DFA* 看作一张有向图，那么 *DFA* 中的状态对应于图中的顶点。
- **字符集** $\Sigma$。该自动机只能读取来自字符集 $\Sigma$ 的字符。
- **转移函数** $\delta : Q \times \Sigma \to Q$. 转移函数接受两个参数并返回一个状态，其中第一个参数和返回值均为状态，第二个参数是字符集 $\Sigma$ 中的一个字符。如果将 *DFA* 看作一张有向图，则转移函数对应于顶点之间的有向边，每条边都标记有一个字符。
- **起始状态** $q_0 \in Q$。起始状态是一个特殊的状态。

- **接受状态集合** $F \subseteq Q$。接受状态集合是一组特殊的状态。

对于一个自动机，可以初始时将节点根据是否为接受状态分为两类并迭代划分。在一轮迭代中，将每个节点和自己每种转移所到节点的等价类视作有序序列进行拼接并以此进行哈希，在此处可以使用 Trie 树代替。

当迭代后等价类关系不变时截止，可以证明该算法迭代轮数不超过 $|Q|$ 轮，因此其时间复杂度为 $O(|Q|^2|\Sigma|)$。在信息学的应用中，往往迭代轮数不多，因此效率很高。

该算法可以广泛的应用于各类问题中[22]，碍于篇幅原因在此不再展开。

## 13 一些其它哈希方法

在信息学竞赛中，还有一些较为零散的关于哈希的应用，在此举一两例子以起到抛砖引玉的效果。

在信息学竞赛中，有时目标为判断一个值是否为 0。然而中间结果太大难以存储，此时可以将答案对某大质数取模，判断取模后的值是否为 0。

**例题 13.1** (四点共圆[23])**.** 平面上初始有两个点 $A_1(1,0), A_2(0,1)$，需要进行 $n$ 次操作，每次操作形如加入一个点，其坐标为之前两个点之和或差；或给定四个点判断是否共圆。

保证 $n \le 10^6$。

考虑三维空间下的函数 $z = x^2 + y^2$，将一个点的坐标向上投影到该函数上，则圆上的四个点会变成一个平面，故即判断是否有对应的四点共面，这可以使用矩阵满秩与否进行判断，具体而言，记给定的四个点的编号为 $a, b, c, d$，点 $A_i$ 的横纵坐标分别为 $x_i$ 和 $y_i$，则 $A_a, A_b, A_c, A_d$ 四点共圆当且仅当：

$$\begin{vmatrix} x_a & y_a & x_a^2+y_a^2 & 1 \\ x_b & y_b & x_b^2+y_b^2 & 1 \\ x_c & y_c & x_c^2+y_c^2 & 1 \\ x_d & y_d & x_d^2+y_d^2 & 1 \end{vmatrix} = 0$$

但是每个点的坐标可能很大，难以存储。

注意到行列式的值的绝对值不超过 $16^n$，因此可以任意选择任意 $[10^{17}, 10^{18}]$ 之间的质数 $p$ 进行计算过程中的取模（即将数域从 $\mathbb{Z}$ 改为 $\mathbb{Z}_p$）。

其正确性是由于绝对值若非 0，则其至多是 $[10^{17}, 10^{18}]$ 中 $\frac{\log 16^n}{\log 10^{17}}$ 个质数的倍数，远少于 $[10^{17}, 10^{18}]$ 中质数的个数，因此正确率有保障。

该算法的时间复杂度显然是 $O(n)$ 的。

[22] 如 NOI 2022 第二天第二题，移除石子

[23] 来源：梦熊 NOI 2025 苍穹计划模拟赛

**例题 13.2** (找树[24])**.** 定义 $\otimes_1, \otimes_2, \otimes_3$ 分别为按位与、按位或、按位异或运算。记 $a_i$ 表示 $a$ 的从低位到高位的第 $i$ 个二进制位。定义一个作用在 $w$ 位二进制数上的新运算 $\oplus$，满足对于结果 $a \oplus b$ 的每一位 $(a \oplus b)_i$ 有 $(a \oplus b)_i = a_i \otimes_{o_i} b_i$。不难验证 $\oplus$ 运算满足结合律和交换律。

给出一张 $n$ 个点 $m$ 条边的无向图，每一条边的权值是一个 $w$ 位二进制数（即小于 $2^w$ 的非负整数）。请你找一棵原图的生成树。设你找出的生成树中的边边权分别为 $v_1, \cdots, v_{n-1}$，请你最大化 $v_1 \oplus v_2 \oplus \cdots \oplus v_{n-1}$，你只需要输出这个值或输出图不连通即可。

保证 $n \le 70, m \le 6000, w \le 12$。

考虑将最优化问题转为计数问题，也即对于每种 $i \in [0, 2^w)$，算出图中所有边的 $\oplus$ 的和为 $i$ 的方案数。

由于该操作只是将常见的三种 FWT 操作对位分别计算，因此仍然可以使用 FWT 进行两个序列的卷积。

使用矩阵树定理，则只需求一个对称矩阵的行列式即可。此时可以将矩阵中的每个数单独 FWT 得到一个长度为 $2^w$ 的数列，求完行列式后 IFWT 还原即可。

为了求得矩阵的 FWT 结果，由于此时两个数列的乘积为按位相乘，因此可以对于每一个 $[0, 2^w)$ 中的 $i$ 分别求出矩阵每个位置只考虑第 $i$ 位的元素时整个矩阵对应的行列式，将它们拼接起来即可得到原矩阵的行列式。

为了解决精度问题，可以类似上一例随机选取大质数进行取模，由于对于固定的 $i$，权值为 $i$ 的树的数量不超过 $m^n$，因此可以类似上一题一样证明正确性，该算法的时间复杂度为 $O((n^3 + m)2^w)$。

在文章的最后，笔者想要分享一例自己见到的很有趣的哈希应用，但其与本文关系并不大。

**例题 13.3** (Flower's Land 2[25])**.** 给定一个长度为 $n$ 的字符串的数列 $s$，保证 $s_i \in \{0, 1, 2\}$，你需要维护 $q$ 次操作，每次操作为以下两种之一：

- 给定区间 $[l, r]$，将 $[l, r]$ 中的每个 $s_i$ 替换为 $(s_i + 1) \bmod 3$；
- 给定区间 $[l, r]$，询问若记 $T$ 为将 $s$ 中下标在 $[l, r]$ 中的字符提取出来，能否通过若干次删除 $T$ 中两个位置相邻的一样的字母这一操作将 $T$ 变为空串。

保证 $n, q \le 5 \times 10^5$。

类似线段树结构，若对于字符串 $T$ 记 $f(T)$ 表示 $T$ 能被删成的最短的字符串，显然 $f(T)$ 是良定义的。

而所有字符串可以通过对应的 $f$ 被划分为若干个等价类，所需要判断的即为字符串 $T$ 与空串是否在同一个等价类中。

[24] 来源：THUPC 2019 K 题

[25] 来源：Problem D, CCPC Final 2022

而 $f(S+T)=f(f(S)+f(T))$，一种暴力的想法是用线段树维护 $T$，但是 $T$ 的长度最多为 $n$，不可以高效维护。

一个必要条件是每种字符的出现次数为偶数，但明显是错的。考虑其失败原因，实质上是因为 $f$ 这一函数没有交换律。

但是，$f$ 具有结合律（即 $f(f(S+T)+U)=f(S+f(T+U))$），这启示着使用一个矩阵作为 $f(T)$ 的哈希函数，而字符串的拼接即为矩阵乘法。

考虑对于每一种元素赋随机 $2\times 2$ 可逆矩阵 $M$，并将其在奇数下标出现时的位置的权值赋为 $M$，偶数下标出现时赋为 $M^{-1}$（这是因为一次删除必然删除的是一奇一偶两个位置）。

此时两个相同的相邻的字母对应的矩阵的乘积是单位矩阵，故一个区间可以被删空当且仅当区间内所有元素对应矩阵的乘积为 $I$，使用线段树维护即可，时间复杂度为 $O(n+q\log n)$。

需要注意的是，由于精度限制，可以取大质数 $p$，计算时在域 $\mathbb{F}_p$ 下进行。正确性可以类似前文所述方法证明，在此略去。

## 14 总结

哈希常用于需要划分等价类的问题之上，本文通过建立起 Laurent 多项式、无序序列、有序序列的联系，并对 Laurent 多项式中的系数所在域（整数域和 $\mathbb{Z}_p$）进行了探讨。

后文中，笔者对于循环序列的哈希进行了探讨，并对树哈希、图哈希这类更为复杂的组合对象进行考虑，并指出了树哈希和集合嵌套哈希的等价性，这扩展了集合哈希的域。

事实上，对于本文中大部分算法，都可以扩展到其它域（比如笔者猜想循环同构意义下的哈希的数域也可以是 $\mathbb{F}_p$，尚未验证）。

最后，笔者讨论了保证正确性的哈希方式，并基于此提出了二叉合并理论。在文末，笔者探讨了一些独特的哈希方式的应用。

除了单纯的划分等价类外，上文所述的部分哈希方式还支持全局加 $x$ 等操作，这也为解决很多问题提供了新的思路。

笔者希望能够通过对以上哈希方法的一些探讨，启发大家的一些思考，对哈希有着更深入的理解，也欢迎刚兴趣的同学在此方面继续研究。

## 15 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢本次国家集训队训练负责人任舍予给予的关心和指导。

感谢时庆钰（Qingyu）、钱易（skip2004）提供的训练平台网站。

感谢福建师范大学附属中学信息学竞赛教练周成、王可可的栽培。

感谢学校团队以及历届学长学姐在我学习时对我提供的帮助。

感谢仇嘉明同学、黄奕天同学分别为本文 Nimber 域下哈希部分内容以及循环同构线性结构哈希部分内容提供学术指导。

感谢余杰瑞同学、官文翔同学、任舍予学长为该论文的选题提供灵感。

感谢学长们、COPY 老师、ilem 老师以及网友与身边同学们为我提供的精神鼓舞。

感谢家人对我的关心和支持。

感谢自己多年以来的付出。

## 参考文献

[1] László Babai. Graph isomorphism in quasipolynomial time. 2015.

[2] HaHeHyt (cnblogs). 循环同构哈希.

[3] hhoppitree (cnblogs). 证明：割空间以及环空间的直和为边空间当且仅当图的生成树个数为偶数.

[4] Evangelos Kosinas. Computing the 5-edge-connected components in linear time. 2023.

[5] MatrixGroup. 题解：at_abc274_h [abc274ex] xor sum of arrays.

[6] Ge Nong, Sen Zhang, and Wai Hong Chan. Linear suffix array construction by almost pure induced-sorting. 2009.

[7] OI-Wiki. 字符串哈希.

[8] OI-Wiki. 有限状态自动机.

[9] OI-Wiki. 树哈希.

[10] Wikipedia. Schwartz–zippel lemma.

[11] LesterCircle (Zhihu). 分圆多项式与 kn+1 型素数.

# 浅谈一类数据结构问题中的等价类思想

重庆市巴蜀中学校 刘家炜

## 摘 要

本文介绍了等价类思想在一类数据结构问题中的应用，描述了等价类分治算法，并列举了几类与等价类相关的题目及解题技巧。

## 1 前言

等价类思想是算法竞赛中的一类重要思想。在数据结构问题中，常见的关于等价类的应用是，凭借结构中的等价类减小问题规模。

本文将以普及算法和总结技巧为目的，介绍几类涉及等价类思想的数据结构问题和算法。

如无特殊说明，下文的“树链”均指树上路径，下文的“矩形”均指平面直角坐标系上边与坐标轴平行的矩形。

## 2 等价类分治算法

该算法通常被称为“最优离线范围修改查询算法”。由于这个算法的核心思想是利用操作分治过程的等价类来缩小问题规模，我们往往使用“等价类分治”来称呼该算法。

篇幅所限，本文将略去算法的部分技术细节，着重强调算法的主要思想。

**定义 2.1.** 点与范围：

令全集 $\Omega$ 表示整个空间，$\Omega$ 中的一个元素称作一个点。

一个范围 $R$ 是 $\Omega$ 的一个子集。称 $R$ 包含 $p$ 当且仅当 $p \in R$。

举例来说，对于一个序列，序列中每个元素可以视作一个点，序列上的区间就是一个范围。

**定义 2.2.** 信息与变换的代数性质：

该算法维护的代数结构具有如下性质：

- **信息的可加性**：信息之间存在加法运算 $+$，这里的加法具有交换律和结合律。
- **变换的分配律**：变换 $F$ 可以应用到信息 $u$ 上，表示为运算 $F \cdot u$。变换应用到信息上的过程具有分配律，即对于变换 $F$ 和信息 $u, v$，有 $F \cdot (u + v) = F \cdot u + F \cdot v$。
- **变换的复合性**：变换之间存在复合运算 $\circ$，这里的复合运算具有结合律，即 $(F \circ G) \circ H = F \circ (G \circ H)$。变换施加到信息上也具有结合律，即 $(F \circ G) \cdot u = F \cdot (G \cdot u)$。

直观来说，我们可以简单认为“信息”是向量，而“变换”是矩阵，信息与变换之间的关系和性质，和向量与矩阵类似。

## 2.1 问题形式

给定一个点集 $P \subseteq \Omega$，称为关键点集。其中每个元素具有初始的点信息。

给出一个操作序列，每个操作是一次修改或查询。

每次修改给出一个范围与变换，对范围内所有关键点应用变换。

每次查询给出一个范围，计算这个范围内关键点信息之和。

## 2.2 算法框架

算法的核心是在进行对操作序列的分治。

把 $m$ 个操作编号为 $0, 1, \ldots, m-1$。

考虑一个过程：$F(L, R)$ 表示处理编号 $L, L+1, \ldots, R-1$ 的操作。

对于 $F(L, R)$，取这段区间 $[L, R)$ 的中点 $M$，这个过程可以直接递归调用 $F(L, M), F(M, R)$ 两个子过程。

这个过程在递归到单个操作，即 $L + 1 = R$ 时无法继续递归，此时我们处理操作 $L$。

在上面的描述中，我们描述了一个朴素的暴力算法。接下来我们进行优化。

## 2.3 等价类的概念

记操作 $i$ 的范围是 $R_i$。

在过程 $F(L, R)$ 中，定义两个点 $P, Q$ 属于同一个等价类，当且仅当 $\forall i \in [L, R)$，都分别有 $P \in R_i, Q \in R_i$ 或 $P \notin R_i, Q \notin R_i$。

简单来说，这个过程中的所有范围 $R_i$ 要么同时包含点 $P, Q$，要么同时不包含这两点。

为了优化算法，在分治过程中，我们保证这个等价类数量的最小化。

## 2.4 完整的算法

在从 $F(L,R)$ 转到 $F(L,M)$ 的过程中，等价类之间只会产生合并。

在转移过程中，我们为 $F(L,M)$ 的每个等价类建立虚点，它处理上一层分治 $F(L,R)$ 中的等价类的合并，作为这些等价类的父结点。在合并的同时，也要维护信息的和。

递归的终点是 $L+1=R$，此时等价类只有 $O(1)$ 个，只需对涉及的等价类虚点进行查询或者修改即可。

对于修改，对涉及的等价类虚点的信息和应用变换，同时打一个懒标记。

对于查询，答案就是涉及的等价类虚点的信息和。

在递归 $F(L,M)$ 结束后，要把这些虚点的标记下传，同时删除这些新的虚点。接下来，只需按类似方法处理 $F(M,R)$。

## 2.5 算法的复杂度

定义 $G(x)$ 表示 $x$ 个操作范围最多把点集划分为多少个等价类。显然这里的 $G(x)$ 只与范围的性质相关，与信息与变换的性质是无关的。

我们称“代数结构上的操作次数”是信息合并与标记应用与下放的次数。

该算法在代数结构上的操作次数可以写作 $T(n,m)=O(\min(G(m),n))+2T(n,\frac{m}{2})$，最终的复杂度是 $O(n)+T(n,m)$。

该算法被证明，当 $G(x)=\tilde{\omega}(x)$，即在忽略对数因子后增长速率严格高于线性的情况下，其在代数结构上的操作次数的复杂度是最优的。需要注意，这个最优性不考虑“判定点是否在操作范围内”等过程的复杂度。

## 2.6 算法的应用

通常，在 $G(x)>n$ 时，等价类的合并关系会更难处理。因此，实现时往往设置一个阈值 $B$，满足 $G(B)\le n, G(B+1)>n$，然后把操作序列 $B$ 个分一块，分别处理。由于 $G(x)$ 不降，这个 $B$ 唯一存在。可以发现，这不改变代数结构上的操作次数复杂度。

应用算法时最大的难点是判定等价类之间的合并关系。可以根据范围的性质来设计寻找合并关系的算法，也可以直接利用哈希等方式。使用哈希即判定每个点所涉及的操作集合是否一致，只需要对每个操作随机一个哈希值，然后求出所有结点经过的操作的哈希值之和。这就转化为一个“先若干次范围修改，再一次性查所有单点值”的问题，这个问题通常是原问题的降维版本。

由于一些简单范围（如区间、树链、树的子树）均有对应的简单数据结构来维护，等价类分治一般只用于处理较复杂的范围，如矩形、半平面、圆、双曲线。

除了复杂范围之外，下文还将介绍一些应用等价类分治算法的问题。

## 3 分治过程中处理等价类的例子

**例题 3.1.** 离线动态 *MST* [1]

给定 $n$ 个点 $m$ 条边的无向图，边有边权。

$q$ 次修改，每次修改一条边的权值，然后查询这张图的最小生成树边权和。

在本文的例题中，如无特殊说明，都认为 $n, m, q$ 同阶，此时另外的变量都以 $n$ 代指。

仍然考虑对操作序列进行分治。为了方便，不妨假设所有边权均不同。称执行前 $i$ 次修改后的图状态为版本 $i$。

假设分治到了 $F(L, R)$，此时有 $c = O(R - L)$ 条边的边权会产生修改。

考虑令这 $c$ 条边的边权为 $-\infty$，再使用 Kruskal 算法得出一棵生成树 $T_0$；再令这 $O(c)$ 条边的边权为 $+\infty$，得出另一棵生成树 $T_1$。

有结论：除去这变化的 $c$ 条边，在 $T_0$ 中的边在版本 $t \in [L, R)$ 中总是存在；不在 $T_1$ 中的边在版本 $t \in [L, R)$ 中总是不存在。

上述结论可以使用 Exchange Argument 证明：在边权不同的情况下使用 Kruskal 算法，得到的最小生成树唯一，且增加一条边 $e$ 的边权只可能使另一条边 $e'$ 替换边 $e$；减小边权则反过来。

这样我们能得到，除了修改的 $c$ 条边之外，最多有 $c$ 条边不确定（即在 $T_1$ 中但不在 $T_0$ 中出现），其余边要么确定在生成树上，要么不在。我们在分治过程中，只保留这些不确定的边。

递归到 $L + 1 = R$ 时即终止，此时只有 $O(1)$ 条需要处理的边，容易计算答案。

在分治过程中，需要处理当前固定的边构成的生成树的连通性，在分治完成整个区间之后还需回退这个部分的所有操作。这可以使用可撤销并查集维护，单次操作的复杂度是 $O(\log n)$。由于分治过程中，保证了每个区间中有效的边数不超过其长度，因此总的复杂度可以写作 $T(n) = O(n \log n) + 2T(\frac{n}{2})$，解得 $T(n) = O(n \log^2 n)$。

在这个例子中，虽然没有明显的等价类，但是使用了分治的核心思想：在分治过程中，保证区间内有效元素的数量。

**例题 3.2.** *Communication Towers* [2]

给定 $n$ 个点 $m$ 条边的无向图，每个点 $u$ 有点权区间 $[l_u, r_u]$。

称两点 $a, b$ **可达**，当且仅当存在一条图上的路径 $a = v_1, v_2, \ldots, v_k = b$，满足路径上这 $k$ 个点的点权区间的交非空。

求点 1 可达的所有点。

暴力的做法是，考虑每一个值 $x$，找出点权区间包含 $x$ 的那些连通块，并把结点 1 所在连通块中的所有点都标记。

[1] https://www.luogu.com.cn/problem/P3206。

[2] https://www.luogu.com.cn/problem/CF1814F。

对值域进行分治，并在分治的过程中处理连通块。

对于一个区间 $[l_u, r_u]$，把这个区间拆分后挂载在分治树上对应的 $O(\log n)$ 个区间上，然后在分治过程中处理即可。事实上，这就是对值域进行线段树分治算法。

由于分治的过程需要回退，这里我们仍然使用可撤销并查集维护所有点的连通性。同时，每次递归到 $L+1=R$ 并结束时，要对 1 所在的连通块上打标记。分治结构回退时，需要对并查集做撤销操作，这会断开并查集结构的一些连接。我们只需在断开连接时下放父亲结点的标记，就能正确维护所有结点的标记。

对于其复杂度：每个操作会挂载到 $O(\log n)$ 个区间，每次处理操作需要 $O(\log n)$ 的代价，总时间复杂度 $O(n \log^2 n)$。

**例题 3.3.** *D2T2* [3]

给定一个长度为 $n$ 的整数序列。

$q$ 次询问 $l, r, L, R$，表示保留值在 $L, R$ 之间的元素，其余元素设为 0 之后，区间 $l, r$ 的最大子段和。

**解法 1：传统的分块做法**

假设没有值域的限制，传统的做法是：使用数据结构维护区间信息（最大子段和，最大前缀和，最大后缀和，区间和）。这个信息可以合并且具有结合律。

把整数序列每 $B$ 个元素分一块。由于每块内只有 $B$ 个不同的值，在值域上本质不同（即保留元素的集合不同）的询问只有 $O(B^2)$ 个。

我们预处理出这 $O(B^2)$ 个信息。接下来，对于所有询问，如果询问区间与当前块部分重叠，就作为散块暴力处理；如果覆盖了整个块，就查表处理预处理中对应的信息，可以做到 $O(1)$。

对于预处理的过程，可以对分好的每一块继续分治：对于左闭右开区间 $[L, R)$，取中点 $M$，然后先处理 $[L, M)$ 和 $[M, R)$ 的信息，这些信息分别有不超过 $(\frac{R-L}{2})^2 + O(R-L)$ 个。接下来，对于这个区间涉及的 $O((R-L)^2)$ 个本质不同信息，只需要定位每个信息在两边对应的内容，就可以 $O(1)$ 合并。

对于分治的过程，可以计算复杂度为 $T(n) = O(n^2) + 2T(\frac{n}{2})$，解得 $T(n) = O(n^2)$，即预处理一块的复杂度为 $O(B^2)$。这样我们得到了复杂度 $O(\frac{n^2}{B} + nB)$ 的算法，取 $B = \sqrt{n}$ 即得到最优复杂度 $O(n\sqrt{n})$。

不同于上述两个例子，在这个例子中，我们先使用分块，凭借每块内询问的等价类数量是 $O(B^2)$，使用分治处理这些等价类。

**解法 2：转置元素与询问**

使用贡献法，把元素和询问的地位转置，即考虑单个元素对涉及它的询问做贡献。

[3] https://www.luogu.com.cn/problem/P5611。

我们把有效的元素视为操作，那么一个点（询问）的答案就是它经历的操作的最大子段和。这个操作显然具有结合律。

采用离线扫描线处理区间的限制：对序列扫描线，在 $p = l$ 时插入一个询问（点），在 $p = r + 1$ 时查询该点的信息。

对于一个元素 $x$，它应该对当前已插入且未被删除的询问中，$L \le x \le R$ 的那些产生贡献。把 $(L, R)$ 视作二维平面上的一点，$L \le x \le R$ 对应的范围是一个矩形。

因此，这个问题被描述为：$O(n)$ 次矩形修改，$O(n)$ 次单点查询。至此，可以使用 KDT 维护这个过程，复杂度 $O(n\sqrt{n})$，但常数因子略大。

事实上，对于第二个做法，考虑把“矩形修改，单点查询”的过程使用对矩形的等价类分治处理。我们先对元素序列分块后，每块做一遍等价类分治，仔细观察可以发现：除去散块暴力的部分，该过程与上面的序列分块的维护方式是几乎一致的。

在这个例子中，我们用两种不同的思考方向，得出了几乎一致的结果。这提示我们，等价类分治算法并不一定严格按上述过程维护有根森林，对操作分治过程中等价类的观察才是更重要的。

# 4 范围并问题

## 4.1 问题形式

在这一节的问题中，我们都讨论询问可离线的情况。

### 4.1.1 区间范围并

给定空间 $\Omega$ 中，大小为 $n$ 的点集 $|P| = n$，以及一个长度为 $m$ 的范围序列 $R_m$。多次询问一个区间的范围的点集并信息和。

信息同样可加，且满足交换律和结合律。

一些简单的例子：

- **静态区间数颜色**：每种颜色（范围）$R_i$ 对应一个点，信息是点权（即该颜色对应的权值），询问就是范围的点集并的权值和。
  对于下文的“数颜色”问题，我们都假定颜色是带权的，即不考虑 bitset 等做法。
- **区间的区间/矩形并**：每个范围是一个区间/矩形，每次询问求一个区间中所有区间/矩形的点集并的信息和。

### 4.1.2 其他的范围并

类似地给定空间中的点集与 $m$ 个范围，但 $m$ 个范围不一定构成序列，而是构成树上结点或平面上的点等较复杂的结构。

查询即查询树链的范围并，或矩形、半平面的范围并等。

例如树链数颜色、矩形数颜色等问题，都属于这里。

## 4.2 区间范围并的处理

对范围序列扫描线，扫到 $r$ 时，使用数据结构维护每个点在考虑 $[1,r]$ 时，所属的范围中编号最大的那个。

维护的过程相当于在扫描线从 $r$ 到 $r+1$ 时，给范围 $R_{r+1}$ 覆盖一个编号 $r+1$。

若把“最后一次被覆盖的时间”相同的两点，视为同一等价类。范围覆盖编号的过程，相当于对等价类进行动态的拆分和重组。

一般来说，我们还需维护另一个数据结构，维护等价类的信息（后缀）和。这个等价类要维护的信息，与具体的范围无关，只与等价类的拆分重组有关。

**例题 4.1.** 区间数颜色 [4]

给一个序列，每个元素有一个颜色，每次查询一个区间中出现过的颜色的权值和。

这是经典问题。对序列扫描线，扫到 $r$ 时处理形如 $[l,r]$ 区间的所有询问。在扫描线时，维护所有颜色最后一次出现的位置，求和时只对这些位置求和。可以使用树状数组快速修改与求和。

如果把一种颜色视作一个点，序列中的一个元素就是一个仅含一个点的范围，以这样的思路套用上面的处理方法：同样是维护一个颜色最后一次出现的时间，等价类的后缀和使用树状数组维护。

上述两个做法是完全一致的，时间复杂度 $O(n\log n)$。

**例题 4.2.** *Tourism* [5]

给定一棵 $n$ 个结点的树。给定一个长度为 $m$ 的序列 $a$，每个元素是树上的一个结点。有 $q$ 次询问，每次询问 $a$ 的一个区间，求最小的包含这个区间中所有点的连通块大小。

把序列每个元素看作到根的链，可以发现一个区间的答案是：区间中所有结点到根链的并，再减去区间内所有点的 LCA 到根的那一段链。

对树链染色的数据结构，可以使用轻重链剖分 + 颜色段均摊。另一种想法是，LCT 的 access 过程就是对一个点到根的路径做了一次实链覆盖，因此每次链覆盖都直接用一次 access 解决即可。

[4] https://www.luogu.com.cn/problem/P1972，这个问题中颜色不带权。

[5] https://www.luogu.com.cn/problem/P9340。

还需要额外的一个数据结构维护后缀和。若使用树状数组维护，可以得到 $O(n\log^2 n)$ 的时间复杂度。

**例题 4.3.** *rrusq* [6]

给定平面上 $n$ 个点，点有点权。

给定一个长为 $m$ 的矩形序列 $a$。

有 $q$ 次询问，每次询问 $a$ 的一个区间，求区间内矩形的并包含的点的点权和。

套用上面的做法，我们应该实现一个矩形覆盖的数据结构。

使用 KDT 处理上述矩形覆盖操作。

实现细节上，对于范围覆盖，我们需要在对应数据结构上打覆盖标记。打覆盖标记时，除了保证当前点到数据结构的根没有其他标记，还需保证子树内没有标记。

因此我们需要对一个子树做标记的回收，这个过程只需要暴力递归清空，在子树内完全无标记的情况下就剪枝跳出。可以发现，这样暴力清空的总复杂度不会高于下放标记的总复杂度，因此这个暴力操作不改变整个问题的复杂度。

此外还需要另一个数据结构处理 $O(n\sqrt{n})$ 次修改和 $O(n)$ 次后缀和查询，可以使用分块做到 $O(1)$ 修改 $O(\sqrt{n})$ 查询。

时间复杂度 $O(n\sqrt{n})$。

**例题 4.4.** 《十字神名的预言者》慈悲（色彩）[7]

给定平面上 $n$ 个点，点有点权。

给定一个长为 $m$ 的半平面序列 $a$。

有 $q$ 次询问，每次询问 $a$ 的一个区间，求区间内半平面的交包含的点的点权和。

$n \le 2\times 10^5, m \le 10^4, q \le 10^5$。要求在代数结构上的操作次数（即点权的合并次数）复杂度 $O(n + m\sqrt{n} + q\sqrt{m})$。

半平面的交不好处理。把所有半平面取反，原问题的半平面交就是现在的半平面并的补集，因此现在只需要解决半平面并问题。

对半平面的覆盖，可以使用等价类分治算法。这里，仍然先把半平面序列 $B$ 个分一块。

对于半平面范围的等价类分治，如果使用哈希，则需要解决“半平面修改，单点查询”问题。这个问题现在有低于根号复杂度的做法，但常数不小且不容易实现。

另一种略容易实现的做法是，按 $x$ 扫描线，在扫描到 $C$ 时维护所有直线与 $x = C$ 交点的相对顺序。在扫描时会有两条直线顺序产生交换，在交换时，处理新产生的所有区域。这样就构造出了初始的 $O(B^2)$ 个区域。

接下来要处理分治过程，这个过程中需要删去一些半平面，并合并所有应该合并的区间。在这里，可以使用链表维护一个半平面边界经过的所有交点，在删去该边界时，就逐个删去所有交点，并合并两侧的区域。

[6] https://www.luogu.com.cn/problem/P6783。

[7] https://www.luogu.com.cn/problem/P11691。

整个过程可以做到 $O(B^2 \log B)$ 的时间复杂度，取 $B = O(\sqrt{n})$，就可以做到时间复杂度 $O(m\sqrt{n}\log n)$，操作代价 $O(m\sqrt{n})$。

另外，我们仍然需要做“标记回收”。观察上一个例子中的“标记回收”过程，可以发现在等价类分治的有根森林结构中，同样暴力修改的复杂度仍然有保证，所以还是可以直接暴力。

还需要另一个数据结构需要支持 $O(1) - O(\sqrt{m})$ 的修改查询，可以使用分块解决。这样就得到代数结构上的操作次数复杂度 $O(n + m\sqrt{n} + q\sqrt{m})$。

## 4.3 一些其他的范围并

这部分问题没有较通用的做法。在这里，一种常见的做法是对每个点考虑贡献，因为这样得到的信息更容易合并。

**例题 4.5** (树链数颜色)**.** [8]

给定一棵 $n$ 个点的树，树上每个点有 $m$ 种颜色之一，每种颜色有权值。

$q$ 次询问，每次给出一条树上路径，计算出现在这条路径上的颜色的权值和。

**解法 1：树上莫队/树分块**

这个题目可以用树上莫队直接解决。

进一步地，把树上莫队的过程改写为树分块，还能以同样的复杂度解决该问题强制在线的版本。

具体来说，使用 Top Cluster 树分块，把树划分为 $O(\frac{n}{B})$ 个簇，对每个点找到它祖先中离它最近的关键点，并认为找到关键点一样的两个点属于同一个块。这里还要求每个块的深度不超过 $B$，这个限制可以用简单的贪心算法解决。

然后，对于两端点在同一块内的询问，暴力遍历链的复杂度是 $O(B)$。

接下来，我们预处理 $\left(\frac{n}{B}\right)^2$ 对关键点之间的颜色数。对于未处理的那些路径询问，它一定能拆成：$O(B)$ 个散点，和一条关键点之间的路径。

这样，对于一个询问，我们只需对 $O(B)$ 个散点，逐一判定它们的颜色有没有在树链上出现过即可。对于判定的过程，可以预处理 $O(\frac{n}{B})$ 个关键点到根的链上，$m$ 种颜色分别出现了多少次，查询时使用树上前缀和相减即得到出现次数。

至此，我们得到时间复杂度与空间复杂度均为 $O(n\sqrt{n})$ 的算法。

更进一步地，利用“$O(B)$ 个散点，最多存在 $O(B)$ 种颜色”的性质：在预处理时记录每块中出现的所有颜色种类，并在预处理每对关键点的信息时，只记录在两个块中出现过的颜色的信息。这个信息可以使用 bitset 直接记录，可以优化空间常数。

**解法 2：逐颜色分块处理**

[8] https://www.luogu.com.cn/problem/P6177，这个问题中强制在线。

基于逐颜色处理的思路，一种可能的做法是，对每个颜色的点建出虚树，然后阈值分治来处理。这种方式仍然可以做到 $O(n\sqrt{n})$ 的时间复杂度，但实现难度可能略大。

把树上结点按颜色排序后分块，使得每块内有不超过 $B$ 个点，且不同块之间不存在同一种颜色。

$B$ 个点能建出 $O(B)$ 个点的虚树，因此，在这个块内本质不同的路径数量是 $O(B^2)$。

对于每条路径，我们暴力预处理：出现在路径上的不同颜色数。对于每个询问，只要可以 $O(1)$ 定位它对应虚树上的哪条路径，然后计算贡献，就能以 $O(\frac{n}{B})$ 的时间代价处理每个询问。

这个分块无法处理单个颜色出现次数 $> B$ 的情况，但对这种情况可以使用树上前缀和 $O(n)$ 处理。

取 $B = \sqrt{n}$，就得到复杂度 $O(n\sqrt{n})$。

**例题 4.6.** 袜子 [9]

给定平面上 $n$ 个点，每个点有颜色。

$q$ 次询问，每次给出一个半平面，查询半平面中每种颜色出现次数的平方和。

**解法 1：逐颜色分块处理**

类似之前的解法 2，按颜色对点排序，然后把点序列 $B$ 个分一块。对于 $B$ 个点，本质不同的半平面数量只有 $O(B^2)$ 个，利用这个性质类似地处理即可。

不同的是，在这个问题中，出现次数 $> B$ 的颜色可以处理，但相对麻烦。更加容易实现的做法是：我们预处理在每个块内，这些半平面的信息：颜色出现次数平方和，块内最小编号的颜色的出现次数，块内最大编号的颜色的出现次数。这个信息可以合并。

预处理和查询可以使用下面的旋转扫描线算法解决：

我们处理一个斜率 $k$ 从 $-inf$（一个充分小的值）扫描到 $+inf$ 的过程，并维护过程中所有点按截距排序的顺序。

在扫描过程中，两个点 $(x_0, y_0)$ 与 $(x_1, y_1)$ 的相对顺序会发生交换，当且仅当斜率 $k$ 扫描经过了 $\frac{y_1-y_0}{x_1-x_0}$。因此，枚举所有点对，计算它们的斜率并对这些斜率排序，接着只使用交换操作就能维护点的截距顺序关系。

对于查询，我们计算一个半平面边界对应的斜率，并在扫描线扫到对应斜率时处理它。此时，我们需要处理的是一个前缀或后缀的信息，并可以二分得出需要处理多长的前缀或后缀。可以发现，上述的交换过程总可以描述为 $O(B^2)$ 次相邻点的交换，而对于相邻点的交换，只有 $O(1)$ 个前后缀信息会产生变化，因此这个过程也可以维护。

由于过程中需要排序和二分等操作，一个块的复杂度是 $O((B^2+n)\log B)$，这样我们最终得到 $O(n\sqrt{n}\log n)$ 的算法。

**解法 2：转置点与半平面**

[9] https://www.luogu.com.cn/problem/P8261，这个问题中保证点坐标随机。

考虑点 $(x,y)$ 与半平面 $Ax+By+C<0$ 的关系。

$C=0$ 的情况下，半平面边界一定经过原点，这种情况是简单的，可以极角排序后双指针解决。

其他情况，不妨假设 $C>0$，把半平面改写成 $\frac{A}{C}x+\frac{B}{C}y+1<0$，我们就得到：点 $(X,Y)$ 在 $\frac{A}{C}x+\frac{B}{C}y+1<0$ 上，等价于点 $(\frac{A}{C},\frac{B}{C})$ 在半平面 $Xx+Yy+1<0$ 上。这完成了元素与询问的转置。当 $C<0$ 时，前面的不等式方向会改变，但仍然能类似处理。

因此，我们现在要处理的问题是：半平面修改，单点查询。

如果把半平面序列分块，我们就得到解法 1。

还可以直接使用等价类分治。由于 $n$ 条直线能把平面划分为 $O(n^2)$ 个部分，同时考虑点定位的复杂度，最终得到的时间复杂度仍然是 $O(n\sqrt{n}\log n)$ 的。

如果把这个等价类分治做法再转置回去，还能得到一个类似“合并旋转扫描线”的做法。

## 5 总结

本文简要介绍了数据结构问题中的等价类思想。内容主要包括等价类分治算法，区间范围并问题的处理技巧，以及一些其它范围的范围并问题的处理方式。

以上这些算法与思想都涉及等价类思想，并存在一定的共性。本文综合描述了这些算法的等价类思想，并简要描述了这种等价类思想在一些简单数据结构问题中的应用。

希望本文能起到抛砖引玉的作用，帮助读者总结处理这类问题，并引起读者研究更多算法问题的共性的兴趣。

## 6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢家人对我的陪伴和支持。

感谢黄新军、刘雨俊老师的关心和指导。

感谢李欣隆学长与我交流讨论，给我指导。

感谢其他所有给我帮助的同学。

## 参考文献

[1] 沈吉濒. 《范围修改查询问题相关算法及其应用》. IOI2024 集训队论文集, 2024.

[2] Xinlong Li and Chengze Cai. Offline optimal range query and update algorithm. 2021.

# 对一类二叉 DAG 合并结构的归纳

福建省福州第一中学 卢华睿

## 摘 要

可持久化数据结构中常见的“复制结点、继承子树”等实现方式，本质上将树结构推广为有向无环图（DAG）。为统一刻画这类结构及其应用，本文提出并系统研究了一种通用模型——二叉 DAG 合并结构。该结构以幺半群为代数基础，在叶子处存储信息，通过二叉合并结点维护序列拼接意义下的整体信息，从而将多种可持久化数据结构与区间合并类问题纳入同一理论框架。

本文首先给出二叉 DAG 合并结构的形式化定义，讨论其深度、大小及展开树的序列语义，并证明了在该结构上进行区间查询与区间修改均可分解为与结构深度成正比的结点操作。进一步地，在引入幺双半群与懒标记机制后，给出了二合并结构支持持久化区间修改的一般实现方法。也针对结构拼接与分裂可能导致深度退化的问题提出解决方案。

在此基础上，本文进一步研究了静态二叉 DAG 合并结构的性质，给出了其合并复杂度分析，并提出一种基于 DAG 剖分的查询优化方法，使得在进行线性规模预处理后，查询复杂度可降至 $O(\log S)$。文中通过大量典型例题，展示了该结构在各种问题中的广泛应用。

本文从统一抽象的角度揭示了多种复杂数据结构背后的共性，为设计与分析高效的可持久化算法提供了一种新的视角与工具。

## 1 前言

初学者在接触可持久化线段树、可持久化平衡树等可持久化数据结构时，往往难以直观理解代码中复制一个结点或可持久化线段树单点插入中继承前一个结点这种较隐式的结构。

事实上，这种通过复制或继承结点实现持久化的方式，本质上将原本的树形结构转化为 DAG （Directed Acyclic Graph，有向无环图）。

因此，对此类 DAG 结构的性质及其应用进行系统研究显得尤为必要。基于对相关问题的深入分析与归纳，本文提出了一种通用的数据结构模型——二叉 DAG 合并结构。

## 2 二叉 DAG 合并结构的定义

二叉 DAG 合并结构，是常规可持久化数据结构与一些二合并类问题统一之后的结果，它在叶子处存储信息，并利用辅助 DAG 结点维护若干信息的拼接。下文中简称其为“二合并结构”。

**定义 2.1** (半群). 对于集合 $S$ 以及 $S$ 上的二元运算 $\circ : S \times S \to S$，若 $\forall a,b,c \in S$，均有 $(a \circ b) \circ c = a \circ (b \circ c)$，则称 $\langle S, \circ \rangle$ 为半群。

**定义 2.2** (幺半群). 对于半群 $\langle S, \circ \rangle$，若存在 $e \in S$，使得 $\forall x \in S$，均有 $x \circ e = e \circ x = x$，则称 $\langle S, \circ, e \rangle$ 为幺半群。

**定义 2.3** (二叉 DAG 合并结构). 对于有向无环图 $G = (V, E)$，若其满足以下条件，则称其为二叉 DAG 合并结构（下简称二合并结构）：

- 对于任意结点 $P \in V$，其出度满足 $\deg(P) \in \{0, 2\}$；
  - 若 $\deg(P) = 0$，则称结点 $P$ 为叶子；
  - 若 $\deg(P) = 2$，则将其左右子结点分别记作 $ls_P, rs_P$；
- 存在一个幺半群 $\langle S, \circ, e \rangle$ 与映射 $w : V \cup \{\epsilon\} \to S$，满足 $w(\epsilon) = e$，且对于所有出度为 2 的结点 $P$，均有 $w(P) = w(ls_P) \circ w(rs_P)$。

以下称 $G$ 为基于幺半群 $\langle S, \circ, e \rangle$ 构建的二合并结构，并记 $w_P = w(P)$。

**定义 2.4** (二叉 DAG 合并结构的深度). 对于二叉 DAG 合并结构 $G$，定义结点 $P$ 的深度 $H(P)$ 为 $G$ 上以 $P$ 为起点的最长路。定义 $G$ 的深度 $H(G) = \max\limits_{P \in V(G)} H(P)$。

**定义 2.5** (二叉 DAG 合并结构的子树). 对于二叉 DAG 合并结构 $G$，定义结点 $P$ 的子树 $S_P$ 如下：

- 若 $\deg(P) = 0$，则 $S_P$ 仅包含结点 $P$；
- 若 $\deg(P) = 2$，记 $S_{ls_P} = (V_1, E_1), S_{rs_P} = (V_2, E_2)$，定义 $V$ 为 $V_1, V_2$ 的不交并（即 $V_1 \sqcup V_2$），并加入新结点 $u$。定义 $E$ 为 $E_1, E_2$ 的不交并，并加入边 $(u, \mathrm{root}(S_{ls_P})), (u, \mathrm{root}(S_{rs_P}))$。此时 $S_P = (V, E)$，其根为 $u$。

**定义 2.6** (二叉 DAG 合并结构的大小). 对于二叉 DAG 合并结构 $G$，定义结点 $P$ 的大小 $siz_P$ 如下：

$$siz_P = \begin{cases} 1, & \deg(P) = 0, \\ siz_{ls_P} + siz_{rs_P}, & \deg(P) = 2. \end{cases} \tag{1}$$

# 3 二叉 DAG 合并结构的查询

在讨论二合并结构的查询之前，首先需要明确该结构所维护信息的逻辑形式。

## 3.1 二叉 DAG 合并结构的序列意义

设 $S$ 为所有序列构成的集合，定义运算 $\circ$ 为序列拼接，$e$ 为空序列，则 $\langle S, \circ, e\rangle$ 构成一个幺半群。若二合并结构 $G$ 基于该幺半群构建，则对于 $G$ 中的任意结点 $P$，$w_P$ 即等价于其子树 $S_P$ 中所有叶子结点的权值按先序遍历顺序拼接而成的序列。

在实际应用中，为了保证时空复杂度，通常不再显式维护完整的序列 $w_P$，而是仅维护该序列的特定统计特征（如区间和、区间最值等）。

尽管形式上有所简化，但在逻辑上，结点 $P$ 始终被视为其展开树 $S_P$ 所有叶子结点构成的线性序列。基于这一视角，序列上的区间查询操作可被自然地推广至二合并结构。

## 3.2 二叉 DAG 合并结构的查询

对于二合并结构 $G$ 及其中的任意结点 $P$，定义查询 $Q(G, P, l, r)$ 为 $w_P$ 中下标区间为 $[l, r]$ 的子序列。

**定理 3.1.** *$Q(G, P, l, r)$ 可由 $G$ 中 $O(H(P))$ 个结点的信息通过运算 $\circ$ 组合得到。*

*证明.* 记 $P$ 的子树中所有完全被区间 $[l, r]$ 包含的极大子树的根结点构成的集合为 $\mathcal{M}$。根据极大性的定义，$\mathcal{M}$ 中的子树互不包含，且其叶子节点区间的并集恰为 $[l, r]$。因此，$Q(G, P, l, r)$ 可由 $\mathcal{M}$ 中各结点的信息按序合并得到。以下证明 $|\mathcal{M}| = O(H(P))$。

考虑 $\mathcal{M}$ 中某一深度存在两个作为左子结点的元素 $ls_p, ls_q$，不妨设在遍历顺序中 $p$ 先于 $q$，则在遍历顺序中 $rs_p$ 位于 $ls_p$ 和 $ls_q$ 之间。由于 $ls_p$ 和 $ls_q$ 均被区间 $[l, r]$ 包含，则 $rs_p$ 也必然被其包含。此时 $ls_p$ 和 $rs_p$ 均被包含，意味着 $p$ 被完全包含，因此 $ls_p$ 不应作为极大子树的根出现，与 $\mathcal{M}$ 的定义矛盾。同理可证，每一深度至多存在一个作为右子结点的元素。综上，$\mathcal{M}$ 中每一层深度至多包含两个结点，故总结点数为 $O(H(P))$。 □

根据上述证明，直接递归实现查询 $Q(G, P, l, r)$ 即可，时间复杂度为 $O(H(P))$。

**例题 3.2** (观众小 P[1])**.** 有一场石头剪刀布比赛，共有 $2^n$ 个人参加，每个人每轮的决策是固定的且在输入中给出（实际上是给出了三种决策分别的个数）。

分为 $n$ 轮，在每轮中，第 0 位选手和第 1 位选手对战，胜者作为新的第 0 位选手，第 2 位和第 3 位对战，胜者作为新的第 1 位选手，以此类推。

[1]来源：集训队作业 2018

求从头到尾都没有平局的初始状态中，字典序最小的状态，只需要输出答案的 hash 值以及第 $l$ 到 $r$ 位。

hash 值可以理解为 $B$ 进制下取模。

数据范围：$1 \le n \le 3 \times 10^5$，$0 \le r - l \le 3 \times 10^5$。

首先把比赛转成二叉树，并先忽略每种决策有多少个人的限制。

注意到如果不区分左右儿子，答案只有三种，因为可以通过一个点推出它的儿子状态，即找到它唯一能赢的决策，那么能够自由选择的就只有根的三种情况。

判断是哪一种情况后即可确定不区分左右儿子的方案，不过还要满足字典序的要求。

可以考虑构造出一个二合并结构，结点包含 $R_i, P_i, S_i$，分别表示三种状态子树最小字典序对应的方案，并记录 $R, P, S$ 两两间大小关系，每次往上连边。

于是只需在一个深度 $O(n)$ 的二合并结构上求出 hash 值和 $l$ 到 $r$ 位的值，即二合并结构的区间查询。该做法可以拓展到求第 $l$ 到 $r$ 位的哈希值。

## 4 二叉 DAG 合并结构的修改

**定义 4.1** (幺双半群). 对于幺半群 $\langle S, \circ, e\rangle$ 与 $\langle T, \times, \epsilon\rangle$，若运算 $\times : T \times S \to S$ 满足：

1. 结合律：对于 $\forall t_1, t_2 \in T, s \in S$，均有 $t_1 \times (t_2 \times s) = (t_1 \times t_2) \times s$；

2. 分配律：对于 $\forall t \in T, s_1, s_2 \in S$，均有 $t \times (s_1 \circ s_2) = (t \times s_1) \circ (t \times s_2)$，

则称 $\langle S, \circ, e, T, \times, \epsilon\rangle$ 为幺双半群。

**定义 4.2** (二叉 DAG 合并结构的修改). 给定幺双半群 $\langle S, \circ, e, T, \times, \epsilon\rangle$ 与基于幺半群 $\langle S, \circ, e\rangle$ 构建的二合并结构 $G$。对于结点 $P$ 以及给定的 $t \in T$，定义修改操作 $M(G, P, l, r, t)$ 为：对于 $w_P$ 中下标区间为 $[l, r]$ 的子序列内的所有信息 $v_i$，执行修改 $v_i \leftarrow t \times v_i$。

**定理 4.3.** $M(G, P, l, r, t)$ 可拆分为 $G$ 中 $O(H(P))$ 个结点的子树的修改。

证明. 同定理 3.1。 □

在将修改操作拆分为 $O(H(P))$ 个子树修改后，为保证时间复杂度，类比线段树等数据结构的延迟更新思想，引入如下定义：

**定义 4.4** (懒标记). 对于非叶结点 $P$，定义懒标记 $T_P$ 表示

$$w_P = T_P \times (w_{ls_P} \circ w_{rs_P}). \tag{2}$$

基于 $T$ 对 $S$ 的分配律，懒标记 $T_P$ 的含义是：$w_P$ 所对应序列中的所有元素均需左乘 $T_P$。

值得注意的是，二合并结构作为持久化数据结构，其既有结点的信息是不可变的。因此，所谓的“修改”实际上是构建包含更新信息的新结点 $P'$ 来代表修改后的状态。为确保修改过程中标记复合顺序的正确性，必须执行标记下传操作：

**定义 4.5** (标记下传). 对于结点 $P$，创建其副本 $P'$ 及其左右子结点的副本 $L', R'$。更新子结点副本的标记：$T_{L'} \leftarrow T_P \times T_{ls_P}$，$T_{R'} \leftarrow T_P \times T_{rs_P}$，并将新结点 $P'$ 的标记清空（$T_{P'} \leftarrow \epsilon$）。

标记下传的本质是生成一个已应用（并清除）当前层懒标记的等价新结点。区间修改操作仅需沿着访问路径，对途径的结点执行标记下传与复制，直至到达目标区间进行标记更新，最终返回新版本的根结点。该过程的时间复杂度为 $O(H(P))$。

至此，二合并结构的基础理论框架已构建完毕。利用该结构可以高效解决多种区间信息维护问题。

**例题 4.6** (七影蝶[2]). 给定 $n, q$ 和长为 $n$ 的序列 $a$，$q$ 次查询，每次询问给出非负整数 $L, R$，求

$$\max_{x=L}^{R} \left( \sum_{i=1}^{n} \text{popcount}(a_i + x) \right).$$

数据范围：$1 \le n, q \le 5 \times 10^5$，$0 \le L \le R \le 10^{11}$，$0 \le a_i \le V \le 10^{11}$。

问题实际上是求一个序列的区间最大值。考虑直接维护出这个序列。由于

$$\sum_{i=1}^{n} \text{popcount}(a_i + x) = \sum_{i=1}^{n} \sum_{j>0} [a_i + x \bmod 2^j \ge 2^{j-1}],$$

因此答案可以被拆分成 $\bmod 2^k$ 意义下的若干次区间加。

考虑递推维护二合并结构上对应序列的结点 $P$，从下往上每次把序列拓宽一倍即对应建立左右子结点均为 $P$ 的结点 $Q$，然后令 $P \leftarrow Q$，接下来做 $O(n)$ 次原区间加。

注意到原序列的深度为 $O(\log V)$，于是可以在 $O(n \log^2 V)$ 的时间内求出了该序列对应的二合并结构，然后在 $O(qH) = O(q \log V)$ 的时间复杂度内查询。

该做法仅能通过 $n \le 10^5$ 的部分，完整解法与本文无关，感兴趣的读者可自行查看原题题解。

**例题 4.7** (Equal Mex[3]). 定义整数序列 $[a_1, \ldots, a_m]$ 的美丽值为满足以下 s 条件的正整数 $k$ 的个数：可以将该序列划分为 $k$ 个互不重叠的子区间，使得：

1. 每个元素恰好属于一个子区间；

2. 所有子区间中未出现的最小正整数均相等。

[2]来源：https://www.luogu.com.cn/problem/P11696

[3]来源：CEOI2025

给定整数序列 $[v_1,\ldots,v_n]$。有 $m$ 次询问，每次询问给定 $l,r$，求序列 $[v_l,\ldots,v_r]$ 的美丽值。

数据范围：$1\le n,m\le 6\times 10^5$。

以下记 $\operatorname{mex}(S)$ 为 $S$ 中未出现的最小正整数。注意到若 $\operatorname{mex}(A)=\operatorname{mex}(B)$，则对于 $A,B$ 的拼接 $A+B$ 也有 $\operatorname{mex}(A+B)=\operatorname{mex}(A)$，因此划分出的每个子区间的 mex 恰为询问区间的 mex。也即，设 $w=\operatorname{mex}([v_l,\ldots,v_r])$，则需要将 $[v_l,\ldots,v_r]$ 划分为尽可能多的每段均包含 $1\sim w-1$ 的子区间。

对于 $l$ 从大到小进行扫描线，维护 $f_l[r]$ 表示区间 $[l,r]$ 的答案，考虑新加入一个数 $v_p$ 的变化。对于 $v_p$ 上一次出现的位置 $q$，即最小的满足 $q>p$ 且 $v_q=v_p$ 的 $q$，若 $\operatorname{mex}([v_p,\ldots,v_q])<v_p$，则加入 $v_p$ 不会改变任何区间的答案。否则有两类区间的答案会发生改变：

- 包含 $p$ 但不包含 $q$ 的区间：找到最小的 $x$ 满足 $\operatorname{mex}([v_p,\ldots,v_x])>v_p$，则对于所有 $r\in[x,q-1]$，均有 $f_p[r]=1$；
- 同时包含 $p,q$ 的区间：找到最大的 $y$ 满足 $\operatorname{mex}([v_q,\ldots,v_y])<\operatorname{mex}([v_p,\ldots,v_{q-1}])$ 与最大的 $z$ 满足 $\operatorname{mex}([v_p,\ldots,v_z])=\operatorname{mex}([v_p,\ldots,v_{q-1}])$，则对于所有 $r\in[y+1,z]$，均有 $f_p[r]=f_q[r]+1$。

对于不在这两段区间中的 $r$，均有 $f_p[r]=f_{p+1}[r]$。因此可以直接用二合并结构维护 $f$，时间复杂度 $O((n+m)\log n)$。

## 5 二叉 DAG 合并结构的拼接与分裂

二合并结构的拼接可以通过新建结点，并将两个待拼接的结构分别设为新结点的左右子结点来实现。然而，这种朴素的拼接方式会导致结构深度随操作次数显著增加，在极端情况下可能退化为线性深度，从而导致修改和查询的时间复杂度不可接受。

为了控制结构深度，通常需要引入平衡树机制。在经典平衡树中，Treap、Splay 等虽应用广泛，但其基于随机或均摊复杂度的特性在可持久化场景下可能失效，而另一些平衡树在维护 DAG 结构时难以保证效率。相比之下，WBLT（Weight-Balanced Leafy-Tree）[4] 是一种更为适用的优化方案。

WBLT，是一种重量平衡树，可以支持分裂合并可持久化，具有常数小且易于实现的特点，合并的复杂度为 $O(\log N)$。相比 WBT（Weighted-Balanced Tree），WBLT 采用 Leafy 的结构，与二合并结构一致。由于 WBLT 时刻保证左右子树大小的比例，因此高度稳定为 $O(\log siz)$，也易于进行分裂操作。

进一步地，对于所有二合并结构，均可以从下往上使用 WBLT 合并的方式进行合并，深度不超过 $O(\log N)$，结点数不超过 $O(\log N)$ 倍的原树结点数。合并后的结构上修改和查询的时间复杂度均可以接受。

[4] 来源：https://en.wikipedia.org/wiki/Weight-balanced_tree

**例题 5.1** (考古[5]). 一个文件中的信息被编码成了一个由英文字母组成的字符串文本 $t$。文本相当长，并且包含了许多重复的内容，因此文件以压缩形式存储在硬盘上。解压缩文件使用以下算法：

- 块 1 w，其中 $w$ 是一个字符串。处理此类块时，在字符串 $t$ 的末尾添加字符串 $w$。
- 块 2 pos len，其中 $pos$ 和 $len$ 是正整数。假设字符串 $t$ 中的字符从 1 开始编号。处理此类块时，将字符串 $t$ 中从位置 $pos$ 开始的 $len$ 个连续字符依次添加到字符串 $t$ 的末尾。如果 $len$ 的值足够大，刚刚添加的一些字符可能会在处理同一块时再次使用。

并给定匹配串 $p$，求 $p$ 在 $t$ 中的出现次数。

数据范围：$|t| \le 10^{15}$，$|p| \le 2 \times 10^5$，$n \le 10^4$。

直接考虑维护最终串的二合并结构。

问题的第二个操作会用到把一个串复制若干次，即 $S^k$，由于二合并结构每次可以合并两个数，可以用快速幂的思路去建立一个二合并结构，即先倍增维护出 $S^{2^i}$，然后从小到大枚举 $i$ 去合并出 $S^k$，这样每次的合并复杂度是 $O(\log k + \log |S|)$ 的。

这样会得到一个 $O(n \log|t| + \sum|w|)$ 大小的二合并结构来表示字符串。

最后考虑在二合并结构这个 DAG 上倒着拓扑排序，利用后缀自动机维护出每个点对应的匹配信息，在拼接时利用后缀数组上二分的方法可以求出答案，维护方式与本文无关，故略去。

总时间复杂度 $O((m + \sum|w| + n \log L) \log m)$。

# 6 静态二叉 DAG 合并结构

在动态问题的处理方面，WBLT 已经展现出较为优异的性能。然而，当二合并结构的形态已确定时，仍然存在若干新的应用场景。对于一个形态已知的二合并结构，若仅改变其所承载的信息，我们称之为静态二合并结构。

## 6.1 静态二叉 DAG 合并结构的合并

二合并结构天然支持合并操作。所谓合并，是指处理出一个新的二合并结构，用来表示两个二合并结构对应序列中每个位置的信息做 $\circ$ 得到的序列。

然而，若两个二合并结构展开的树形态不一致，即使合并序列也无法合并 DAG 结构，所以需要初始阶段给定一个二合并结构，而将合并操作定义为将对应的 DAG 展开为树后再执行合并。

若合并满足一个二合并结构不被合并多次，即合并是树形的，有如下定理：

[5] 来源：ROI2019

**定理 6.1.** 静态二合并结构的合并的时间复杂度是 $O(nH)$ 的，其中 $n$ 为合并的次数，$H$ 为结构深度。

证明. 考察最终展开的二合并结构中每个点 $P$ 对复杂度的贡献，若如果合并时两棵树有一颗在 $P$ 处为空，就会停止往下递归。

换言之，把 $P$ 内包含的点在树形合并中对应的结点找出来，只有他们组成的虚树有贡献，因此总复杂度为：

$$\sum_P siz_P = \sum_i H_i = nH$$

□

该结论表明，在发现新的静态二合并结构时，可以自然地考虑其合并操作。典型的静态二合并结构有线段树，边分树，静态 top tree[6] 等。

**例题 6.2** (暴力写挂[7]). 给定两棵树，边有边权，对于所有 $x, y$，求

$$\mathrm{dep}_x + \mathrm{dep}_y - \mathrm{dep}_{\mathrm{lca}(x,y)} - \hat{\mathrm{dep}}_{\hat{\mathrm{lca}}(x,y)}$$

的最大值。

数据范围：$n \le 366666$。

把 $\mathrm{dep}_{\mathrm{lca}(x,y)}$ 拆成有关 $\mathrm{dep}_x, \mathrm{dep}_y, \mathrm{dis}(x,y)$ 的项。

对第一颗树建立边分树的二合并结构，对于第二颗树 dfs，每次把儿子的结构合并在一起。

贡献只和 $\mathrm{dep}_x, \mathrm{dep}_y, \mathrm{dis}(x,y)$ 有关，考虑在边分树上如何上传信息。

我们要求的是 $\min \mathrm{dep}_x + \mathrm{dep}_y + \mathrm{dis}(x,y)$，对于 $(x,y)$ 路径上的点 $e$ 可以拆分为 $\min \mathrm{dep}_x + \mathrm{dis}(x,e) + \mathrm{dep}_y + \mathrm{dis}(y,e)$。

于是只要插入时对边分树上的祖先贡献上 $\mathrm{dep}_x + \mathrm{dis}(x,e)$，这里的贡献区分左右，合并直接 checkmax 就可以了。

时间复杂度为 $O(nH) = O(n \log n)$。

## 6.2 静态二叉 DAG 合并结构的更优查询

前文中，WBLT 通过结点数乘以一个 log 压缩了二合并结构的深度。

然而，WBLT 本质上维护的是动态结构。经验表明，若将动态问题转化为静态问题，往往可以去除额外的 log。

那么自然会产生如下的疑问：给出一个静态二合并结构，并要求做二合并结构的查询，能否在预处理关于二合并结构点数线性的情况下 log 查询。

[6] https://oi-wiki.org/ds/top-tree/

[7] 来源：CTSC2018

在此之前，先给出查询的复杂度理论下界。

**定理 6.3.** 对静态二合并结构 $G = (V, E)$ 的查询复杂度不可能低于 $\Theta(\log S)$，其中 $S = \max_{P \in V} siz_P$。

证明. 二合并结构的查询 $P, l, r$ 中，$r$ 的大小可以到 $\Theta(S)$，所以输入复杂度就是 $\Theta(\log S)$。 □

为了解决这个问题，我们提出基于 DAG 剖分的优化方法，即模仿对树建立的二合并结构——Top tree。

定义 $f_u, g_u$ 分别为以 $u$ 为起点的路径数量和以 $u$ 为终点的路径数量。

考虑对于每个点 $u$ 选择出边 $(u, v)$ 中 $f_v$ 最大的边（若有多个最大值则任选一个），给这些边加上 1 的边权。

再对于每个点 $u$ 选择入边 $(v, u)$ 中 $g_v$ 最大的边（若有多个最大值则任选一个），给这些边加上 1 的边权。

最后取出那些边权为 2 的边，设定这些边为重边，其他的边为轻边。

对重边连向的节点定义为这个点的重儿子，轻儿子同理。

**定理 6.4.** 二合并任意一条路径只会经过 $O(\log S)$ 条轻边。

证明. 从上往下走，每一次经过轻边，若不是 $f_v$ 最大的边则 $f$ 减半，否则 $g$ 翻倍，故不会超过 $O(\log S)$ 次。 □

为避免因用额外的数据结构维护重链信息导致复杂度增大，可以考虑使用建立类似全局平衡二叉树的结构。

对于一条重链 $a_1, a_2, \ldots, a_k$，取出链上节点的 $f$ 值，记为 $s_i$，为维护出 $a_1$ 子树内信息，按照如下方法递归建立二合并结构：

- 每次选出极大的 $p$ 满足 $2\sum_{i=1}^{k} s_i \le \sum_{i=1}^{k} s_i$。
- 递归建立 $a_1, a_2, \ldots, a_p$ 与 $a_{p+1}, a_{p+2} \ldots a_n$，并取他们的拼接作为新的二合并结构。

可以证明将这些树叠加上轻链的边，总深度将被控制在 $O(\log S)$ 内[8]，接下来考虑怎么在这个结构上查询。

对原二合并结构逆拓扑序扫描，可以处理出新结构上每个点的子树信息和作为已知信息。

对查询问题递归求解 $u, l, r$ 表示查询 $u$ 子树中 $l \sim r$ 的权值之和，这可以被表示为这条重链链顶的一段区间和。

**定理 6.5.** 直接在重链建立的二合并结构递归查询，复杂度为 $O(\log S)$。

[8]证明见 [1]

证明. 由于整个重链二合并结构叠加上轻链的深度为 $O(\log S)$，定理等价于递归的每一层的询问个数为 $O(1)$。

在重链调用二合并结构的查询时，若询问分裂，则必然裂成 $a_i, a_j$ 分别的后缀和前缀，下证后缀和前缀不会再被拆分。

注意这里的前后缀是对于重链某个节点 $a_p$ 的子树定义的，而这个子树又对应 $a_p, a_{p+1}, a_n$ 的轻子树信息和。

对于前缀，由于查询为 $a_p, a_{p+1}, a_n$ 的轻子树信息和的前缀上，在拆分后会变成若干已知信息和一个 $a_q$ 的前缀。

对于后缀，显然与前缀同理，查询在拆分后会变成若干已知信息和一个 $a_q$ 的前缀。

即我们证明了对于新结构，递归到每一层的查询个数最多只有两个。 □

综上所述，高深度二合并结构的查询在线性预处理的前提下做到了 $O(\log S)$，这将进一步解决很多问题。

**例题 6.6** (程序校验 II[9]). 有 $n$ 个操作 $x_i, y_i, a_i, b_i$，$m$ 次查询 $l, r, X$，初始化 $A = 1, B = 0$，遍历 $l \le i \le r$，每次若 $X \le x_i$ 则将 $X, A, B$ 修改为 $X + y_i, a_i A, a_i B + b_i$，然后输出结束后的 $X, A, B$。强制在线。

数据范围：$n, m \le 2 \times 10^5$，$y_i, X, A, B \le V = 10^9$。

把后两维看作一个半群。

如果可以离线，这个问题可以采取经典的左端点加入右端点删除的做法，每次将平衡树分裂为 $\le x_i$ 的部分和剩下的部分，对再对两部分有交合并。

强制在线考虑可持久化，从 $l$ 扫描线，定义 $f_{i,j}$ 表示位置为 $i$，$X$ 为 $j$，在遍历 $i \sim n$ 的操作后，发生了题中所说 $X \le x_i$ 的位置集合。

用 WBLT 维护 $f_i$ 构成的二合并结构，为满足 $r$ 可变的要求，需要将所有的半群信息不合并的维护出来，这是另一个序列信息，可以使用 WBLT 维护这个信息的二合并结构，对于 $f_i$ 的二合并结构下传标记时做内层二合并结构的拼接。

查询是对若干个内层二合并结构做拼接取一个前缀，而不完整的查询只需要对一个内层二合并结构做，这样导出一个 $O(n \log V \log n + m \log nV)$ 的做法。

用静态二合并结构的查询优化上述做法，上述做法中，虽然二合并结构的深度很优，但是结点数太多，注意到内层二合并结构不需要支持分裂，也就不需要时刻保证深度，可以先不用 WBLT，直接建出内层树的二合并结构。

问题转化 $n \log V$ 个点的二合并结构，同时 $S$（上文的最大子树大小）不超过 $n$（因为每个点的信息是下标序列，序列大小最多为 $n$），$m$ 次查询。

对这个二合并结构调用上文二合并结构的静态查询可以做到 $O(n \log V + m \log V)$。

[9]来源：https://uoj.ac/problem/1017

# 7 二叉 DAG 合并结构的其他应用

## 7.1 万能欧几里得

万能欧几里得用于解决以下形式的求和式：

$$\sum_{i=1}^{n} f\left(\left\lfloor \frac{pi+r}{q} \right\rfloor\right) g(i).$$

上述求和式一般转化为 $\prod_{i=1}^{n} U^{f(i)-f(i-1)}R$ 进行计算，其中 $U,R$ 为具有结合律的信息，

$$f(i)=\begin{cases}0, & i=0,\\ \left\lfloor \frac{pi+r}{q} \right\rfloor, & i\geq 1.\end{cases}$$

定义如上问题的答案为 $f(n,p,q,r,U,R)$。

若 $\max(p,r)\geq q$，由于

$$\left\lfloor \frac{pi+r}{q} \right\rfloor = \left\lfloor \frac{(p \bmod q)\cdot i+(r \bmod q)}{q} \right\rfloor + \left\lfloor \frac{p}{q} \right\rfloor i + \left\lfloor \frac{r}{q} \right\rfloor,$$

因此可以将 $p,r$ 对 $q$ 取模，即

$$f(n,p,q,r,U,R)=U^{\lfloor \frac{r}{q} \rfloor} f(n,p \bmod q,q,r \bmod q,U,U^{\lfloor \frac{p}{q} \rfloor}R).$$

否则 $p,r$ 均小于 $q$。此时可以枚举第 $y$ 个 $U$，设其前共有 $x$ 个 $R$，则

$$y>\left\lfloor \frac{px+r}{q} \right\rfloor \implies y\geq \frac{px+r+1}{q} \implies x\leq \left\lfloor \frac{qy-r-1}{p} \right\rfloor.$$

对于上式中的 $qy-r-1$，由于 $q-r-1\geq 0$，因此可以转化为

$$\left\lfloor \frac{qy-r-1}{p} \right\rfloor = \left\lfloor \frac{q-r-1+(q-1)y}{p} \right\rfloor = \left\lfloor \frac{q-r-1}{p} \right\rfloor + \left\lfloor \frac{(q-1)\cdot y+(q-r-1) \bmod p}{p} \right\rfloor.$$

而原式中最后一段 $R$ 可以单独计算。记 $m=\left\lfloor \frac{pn+r}{q} \right\rfloor$，则有

$$f(n,p,q,r,U,R)=R^{\lfloor \frac{q-r-1}{p} \rfloor} U f(m-1,q,p,(q-r-1) \bmod p,R,U) R^{n-\lfloor \frac{qm-r-1}{p} \rfloor}.$$

使用辗转相除法递归求值即可。

**定理 7.1.** 上述算法只需要 $O(\log \max(n,p,q))$ 次乘积。

证明. 辗转相除法的递归次数为 $O(\log n)$。余下复杂度由快速幂贡献，在每次 $(p,q)\to(q,p \bmod q)$ 的过程中调用了三次复杂度分别为 $O\left(\log \frac{p}{q}\right), O\left(\log \frac{q}{p \bmod q}\right), O(1)$ 的快速幂，求和得到 $O(\log \max(n,p,q))$。 □

上述算法满足二合并结构的条件，得到的二合并结构对应 $\prod_{i=1}^{n} U^{f(i)-f(i-1)}R$ 展成的序列，深度和结点数均为 $O(\log \max(n,p,q))$。

**例题 7.2** (随机数据[10]). 有 $n$ 件物品排成一个环，用 $0,\cdots,n-1$ 对它们编号，规定物品 $i$ 的价值为 $v_i = w_{i \bmod m}$。

有两人正在做轮流从编号为 0 到 $n-1$ 的物品中取走可用物品的博弈，每次 A 先取一件，B 可从正反相距 $d$ 的位置选取一件或跳过，不能重复取走同一个物品。

支持 $q$ 次切换物品可用状态，要求每次操作后计算 B 的最大总价值。

数据范围：$1 \le d < n \le 10^{16}$，$1 \le m \le 2 \times 10^4$，$q \le 10^5$，$1 \le w_i \le 400$。

把 $(i+d) \bmod n$ 与 $i$ 连边，边权为两点权值 min。

**引理 7.3.** 答案为这张图的最大权匹配。

证明. 对于最大权匹配中的每一对匹配，A 取其中一个点时 B 可以取另外一个点，只要 B 保持这样的操作就能使得最后的权值不小于最大权匹配。

那么 A 是否能保证取到这个值呢？

每一轮 A 可以选择当前图中未被选择的最大值，考虑最终图中 B 选的点，必然都和一个比它大的 A 选的点连接，于是 A 也能保证 B 最后的权值不大于最大权匹配。

答案分别不大于和不小于最大权匹配，所以答案等于最大权匹配。 □

最大权匹配在图固定的情况下可以 dp 求出。

修改物品不可用相当于设权值为 0，直接用线段树维护动态 dp 可以把复杂度做到 $O(n+q\log n)$，考虑优化。

现在余下的问题是处理出没有修改的长段信息，也就是所有 $(di \bmod n) \bmod m$ 位的矩阵信息的积。

这个值可以转化为 $\left(di - n\left\lfloor\frac{d\times i}{n}\right\rfloor\right) \bmod m$，与上文万能欧几里得的形式类似，考虑用万能欧几里得。

所需信息与通常的半群信息不太一样，是以 $\left(di - n\left\lfloor\frac{d\times i}{n}\right\rfloor\right) \bmod m$ 作为下标的信息，所以记录的 $U, R$ 应该是一个序列，表示下标从这个数开始经过一段固定的变化，矩阵乘积是多少，并还要记录一个偏移量。

那么 $U$ 就对应 $-n$，$R$ 对应 $+d$ 并对于每个位置还有一次乘上初始 $w_i$ 矩阵的变化。

对这个问题处理出万能欧几里得，查询实际上就是在查询万能欧几里得的区间信息，使用上文的方法处理即可。

注意查询时不需要对信息合并，只需要记录当前位置，复杂度可以做到 $O(\log n)$。

总时间复杂度 $O((m+q)\log n)$。

**例题 7.4** (轮盘赌游戏[11]). 给定长度为 $n$ 的环，在第 $i$ 个点有 $1-p_i$ 的概率停止，否则走到 $(i+d) \bmod n$。

[10] 来源：北大集训 2021 D4T3

[11] 来源：集训队互测 2025

初始给定 $p'_0 \sim p'_{m-1}$，$p_i = p'_{i \bmod m}$，然后给出一棵树，$q$ 个叶子，每个叶子表示对 $p$ 的单点修改（修改的位置不同）。

对每个结点，求出进行其子树内所有叶子代表的修改后，随机从一个点出发后期望走多少步停止（开始算一步）。

数据范围：$n \le 10^{16}, m \le 5000, q \le 10^5$，$\gcd(n,d)=1$。

考虑问题的动态规划形式：

$$f_i = 1 + p_i f_{i+1},$$

通过将所有数用 $f_0$ 表示并环绕一圈后即可解得方程。

在支持子树修改的场景下，可以在树上启发式地加入修改，并调用前文所述的万能欧几里得区间查询。时间复杂度为 $O(m \log n + q \log n \log q)$。

**例题 7.5** (轮盘赌游戏)**.** 问题同上，但要求查询复杂度为 $O(\log n)$。

仍然考虑建出二合并结构，为了支持子树所有点对应的修改，考虑静态二合并结构的合并。

但直接按照上文建出的二合并结构结构每个点维护的是一个数组，很明显不能对这个二合并结构直接做合并。

但注意到如果把上面每个信息数组中的数都提取出来，在维护万能欧几里得过程中对这些信息的合并建出二合并结构，则取出万能欧几里得合并到最后得到的完整信息的数组第一个值，它作为根表示出的二合并结构就是 $p_0 \sim p_{n-1}$。

使用静态二合并结构合并维护查询，时间复杂度 $O((m+q)\log n)$。

## 7.2 WBLT 之外：可并堆

可并堆代表的二合并结构并无所谓的“合并”，而满足结构中每个点带权 $w_p$，且每个结点的权值比它的孩子小，并不区分左右儿子的顺序。

下介绍一种可并堆：左偏树。

**定义 7.6** (左偏树)**.** 左偏树是一种二合并结构，并有 $d_p = d_{rs_p} + 1$。

且满足条件 $d_{ls_p} \ge d_{rs_p}, w_p \le \min(w_{ls_p}, w_{rs_p})$。

**定理 7.7.** 左偏树的 $d = O(\log siz)$。

**证明.** 考虑 $d_p = n$ 的最小点数 $f_n$，必须满足 $d_{ls_p} \ge d_{rs_p} = d_p - 1 = n-1$，则 $f_n = 2f_{n-1} + 1$。

故 $f_n = 2^n - 1$，$n = O(\log f_n)$。 □

接下来讨论左偏树的合并，为保证复杂度，可以利用右链深度 $d$ 是 $O(\log)$ 的性质，对右链做归并排序后调整树结构。

具体的，对于 $x,y$ 结点，不断执行如下流程 $M(x,y)$，并返回合并后的根结点：

- 若 $x=\epsilon$，则返回 $y$。
- 若 $y=\epsilon$，则返回 $x$。
- 若 $w_x > w_y$，返回 $M(y,x)$。
- 新建结点 $p$，$ls_p \leftarrow ls_x, rs_p \leftarrow M(rs_x,y), w_p \leftarrow w_x$，返回 $p$。

**定理 7.8.** *$M(x,y)$ 的时间复杂度为 $O(\log siz_x + \log siz_y)$。*

*证明.* 考虑设计势能函数 $\varphi(x,y) = d_x + d_y$。

而递归复杂度 $1 = \Delta\varphi(x,y) = d_x + d_y - (d_{rs_x} + d_y)$。

故总复杂度 $\le \varphi(x,y) - 0 = \log siz_x + \log siz_y$。 □

**例题 7.9** ($k$ 短路). 给定一张有向图，$n$ 个点 $m$ 条边，边权为正，求这张图 $s \to t$ 本质不同的路径中权值第 $k$ 小的。

数据范围：$n,m \le 5\times 10^5$。

建出 $t \to s$ 最短路树，则任意一条从 $s \to t$ 的路径可以被划分为树边和非树边。

只考虑非树边，相邻的非树边的限制可以表示为后一条边的起点是前一条边的终点的祖先，边权定义为 $d_v - d_u + w$，那么一条路径的代价可以只用非树边表示，

接下来我们考虑对这个非树边序列求出其边权和的前 $k$ 小值，考虑先刻画一个边序列的抵达方式：

- 在序列末尾添加一条边。

但这样会导致可选择的方案太多，考虑换一种方式：

- 在序列末尾添加一条最小边。
- 将序列末尾的边替换。

而序列末尾边唯一的限制是它的起点要是某个点的祖先，那么我们用可并堆维护出每个点祖先的出边集合（定义每个二合并结构结点的权值为这条边对应的边权）。

于是每一种边序列等价于下图中的一条路径：

- 对于二合并结构上的点连向它出边的点对应的二合并结构的根结点 $root_v$，边权为 $root_v$ 的权值。
- 二合并结构上每个点 $p$ 向它的孩子 $s$ 连边，边权为 $s$ 的权值减去 $p$ 的权值。

由于这张图边权均为正，可以使用小根堆维护当前所有点，每次取出最小的并扩展所有状态。总复杂度 $O(m\log m + k\log k)$。

## 8 总结

本文尝试从更基本的角度理解数据结构，提出了二叉 DAG 合并结构，从基础的修改查询开始介绍，并通过一些例题展示了从这个角度思考问题的优点。

笔者期望本文能起到抛砖引玉的作用，在启发读者思考的基础上，帮助其提炼出更为精炼的结构与方法，并探索更深入的应用方向。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练任舍予的指导。

感谢家人与朋友的支持与鼓励。

感谢福建省福州第一中学的栽培，林国军老师的教导和同学们的帮助。

感谢沈吉颁学长为本文提供启发。

感谢黄佳旭学长、林祺昊同学、王茂骅同学为本文审稿。

感谢陈翔乐学长的答疑解惑。

感谢徐骁扬学长在集训队互测 2025 中命制的题目《轮盘赌游戏》，展示了二合并结构的雏形。

## 参考文献

[1] 戴江齐. (2022). IOI2022 国家集训队论文，《浅谈一类特殊 DAG 上的路径信息维护问题》。

[2] 徐骁扬. (2024). 集训队互测 2025，《轮盘赌游戏》。

[3] fjy666. (2025). JRKSJ ExR，《七影蝶》。

[4] 李欣隆. (2025). UOJ Long Round 3，程序校验 II。

# 综述卷积类问题的一些处理方法

中国人民大学附属中学 彭亦宸

## 摘要

本文介绍了解决 OI 中卷积类问题的几种结构和处理方法，并将很多看似难以想到、相互独立的卷积类问题联系到了一起。

## 1 高维广义卷积

### 1.1 定义

**定义 1.1** (高维广义卷积)**.** 令维数 $k \in \mathbb{Z}^+$，$f : \prod_{i=1}^{k} U_i \to F, g : \prod_{i=1}^{k} V_i \to F$，卷积结果为 $h : \prod_{i=1}^{k} W_i \to F$，其中 $F$ 为一种域，$U_i, V_i, W_i$ 为有限集合。

对每一维 $i$，令第 $i$ 维贡献函数 $c_i : U_i \times V_i \times W_i \to F$。

令**每维独立贡献函数** $c : \left(\prod_{i=1}^{k} U_i\right) \times \left(\prod_{i=1}^{k} V_i\right) \times \left(\prod_{i=1}^{k} W_i\right) \to F$ 满足：

$$c(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \prod_{i=1}^{k} c_i(u_i, v_i, w_i), \qquad \forall \mathbf{u} \in \prod_{i=1}^{k} U_i, \mathbf{v} \in \prod_{i=1}^{k} V_i, \mathbf{w} \in \prod_{i=1}^{k} W_i.$$

那么 $f$ 与 $g$ 在 $c$ 贡献函数下的高维广义卷积结果 $h$ 满足：

$$h(\mathbf{w}) = \sum_{\mathbf{u} \in \prod_{i=1}^{k} U_i} \sum_{\mathbf{v} \in \prod_{i=1}^{k} V_i} f(\mathbf{u})\, g(\mathbf{v})\, c(\mathbf{u}, \mathbf{v}, \mathbf{w}), \qquad \forall \mathbf{w} \in \prod_{i=1}^{k} W_i.$$

容易发现，对于每个元素独立的集合幂级数卷积（例如交、并、对称差、子集卷积等），都可以归约到高维广义卷积。

**定义 1.2** (张量)**.** 在域 $F$ 上定义 $n$ 阶张量 $\mathbf{X}$：

$$\mathbf{X} \in F^{I_1 \times I_2 \times \cdots \times I_n}.$$

其中 $I_j$ 为第 $j$ 个维度的大小（$j = 1, 2, \ldots, n$）。

张量可以被视为一个 $n$ 维数组，其中元素 $(i_1, ..., i_n)$ 记作 $X_{i_1, i_2, ..., i_n}$，要求 $\forall p, 0 \le i_p < I_p$。

二阶张量与矩阵可以相互转换，在后文中我们对这两者不加以区分。

**定义 1.3** (张量加法)**.** 令 $\mathbf{X}, \mathbf{Y} \in F^{I_1 \times I_2 \times \cdots \times I_n}$。
则它们的加法 $\mathbf{X} + \mathbf{Y} = \mathbf{Z} \in F^{I_1 \times I_2 \times \cdots \times I_n}$，满足：

$$Z_{i_1,i_2,\cdots,i_n} = X_{i_1,i_2,\cdots,i_n} + Y_{i_1,i_2,\cdots,i_n}, \qquad \forall 0 \le i_p < I_p.$$

**定义 1.4** (张量直积)**.** 令 $\mathbf{X} \in F^{I_1 \times I_2 \times \cdots \times I_n}$，$\mathbf{Y} \in F^{J_1 \times J_2 \times \cdots \times J_m}$。
则它们的直积 $\mathbf{X} \circ \mathbf{Y} = \mathbf{Z} \in F^{I_1 \times I_2 \times \cdots \times I_n \times J_1 \times J_2 \times \cdots \times J_m}$ 为一个 $n+m$ 阶张量，满足：

$$Z_{i_1,i_2,\cdots,i_n,j_1,j_2,\cdots,j_m} = X_{i_1,i_2,\cdots,i_n} Y_{j_1,j_2,\cdots,j_m}, \qquad \forall 0 \le i_p < I_p, 0 \le j_q < J_q.$$

**定义 1.5** (张量直和)**.** 令 $\mathbf{X} \in F^{I_1 \times I_2 \times \cdots \times I_n}$，$\mathbf{Y} \in F^{J_1 \times J_2 \times \cdots \times J_n}$。
则它们的直和 $\mathbf{X} \oplus \mathbf{Y} = \mathbf{Z} \in F^{(I_1+J_1) \times (I_2+J_2) \times \cdots \times (I_n+J_n)}$，满足：

$$Z_{i_1,i_2,\cdots,i_n} = \begin{cases} X_{i_1,i_2,\cdots,i_n} & \forall p, 0 \le i_p < I_p \\ Y_{(i_1-I_1),(i_2-I_2),\cdots,(i_n-I_n)} & \forall p, I_p \le i_p < I_p + J_p \\ 0 & \text{其他情形} \end{cases} \qquad \forall 0 \le i_p < I_p + J_p.$$

**定义 1.6** (秩一张量)**.** 秩一张量 $\mathbf{X} \in F^{I_1 \times I_2 \times \cdots \times I_n}$ 可以表示为 $n$ 个一阶张量的直积：

$$\mathbf{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(n)}.$$

那么秩一张量的元素满足：

$$X_{i_1,i_2,\ldots,i_n} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(n)}_{i_n}, \qquad \forall 0 \le i_p < I_p.$$

**定义 1.7** (张量的秩)**.** 一个张量 $\mathbf{X} \in F^{I_1 \times I_2 \times \cdots \times I_n}$ 的秩为最少使用的秩一张量数，使得这些秩一张量的和为 $\mathbf{X}$。

**定义 1.8** (张量扩展 Kronecker 乘法)**.** 令 $\mathbf{X} \in F^{I_1 \times I_2 \times \cdots \times I_n}$，$\mathbf{Y} \in F^{J_1 \times J_2 \times \cdots \times J_n}$。
则它们的扩展 Kronecker 乘法 $\mathbf{X} \otimes \mathbf{Y} = \mathbf{Z} \in F^{(I_1 * J_1) \times (I_2 * J_2) \times \cdots \times (I_n * J_n)}$，满足：

$$Z_{i_1,i_2,\cdots,i_n} = X_{\left\lfloor \frac{i_1}{J_1} \right\rfloor, \left\lfloor \frac{i_2}{J_2} \right\rfloor, \cdots, \left\lfloor \frac{i_n}{J_n} \right\rfloor} * Y_{i_1 \mod J_1, i_2 \mod J_2, \cdots, i_n \mod J_n}, \qquad \forall 0 \le i_p < I_p * J_p.$$

为了简练，后文中将“扩展 Kronecker 乘法”简称为“乘法”。

这些运算有很多平凡的性质，例如：加法具有交换律、结合律，直积具有结合律，直积对加法具有分配律，直积对直和具有分配律，直和具有结合律，乘法具有结合律，乘法对加法具有分配律，秩一张量的直和、直积、乘法后还是秩一张量。

## 1.2 高维广义卷积的设计思路

考虑在计算高维广义卷积的过程中，使用的所有“乘法”满足的性质。

注意到 $h$ 为关于 $f$ 、$g$ 的双线性函数。

因此，在计算过程中，任何有意义的乘法操作都必然是 $f$ 的某个线性组合与 $g$ 的某个线性组合的乘积。

我们假设一共进行 $k$ 次乘法操作。那么 $h$ 一定得由这 $k$ 个乘积通过线性组合得到。

于是求解卷积的过程可以划分为三个有严格顺序的步骤：

1. 求出 $k$ 次乘法操作所需要的所有 $k$ 个 $f$ 的线性组合与 $g$ 的线性组合。
2. 进行 $k$ 次乘法操作。
3. 将第二步的 $k$ 个乘积通过线性组合得到卷积结果 $h$。

该如何找到满足上述要求的一系列乘法呢？

对于一个高维卷积对应的 $c$ 贡献函数，将 $\prod_{i=1}^{k} U_i$、$\prod_{i=1}^{k} V_i$、$\prod_{i=1}^{k} W_i$ 均展平，即可视为一个 3 阶张量 $C$。

容易发现，一组 $f$ 的线性组合 $*$ $g$ 的线性组合再分配给 $h$ 的过程，其贡献为一个 3 维秩一张量。

于是只需要找出一些 3 维秩一张量，使得它们的和为 $C$。我们称该过程为对 $C$ 寻找张量分解。

张量分解中得到第 $i$ 个秩一张量的 $a^{(1)}$、$a^{(2)}$ 分别指出了第 $i$ 次乘法需要的 $f$、$g$ 线性组合，$a^{(3)}$ 指出了第 $i$ 次乘法的结果该分别以多少的系数贡献给 $h$ 的每个元素。

然而，直接根据一组张量分解仍然无法快速得到所需要使用的 $f$、$g$ 线性组合的具体值，也无法快速将乘积结果贡献给 $h$ 得到答案。

于是考虑利用“每维独立”的性质，试图一维一维处理。

根据高维广义卷积的定义，可以将 $C$ 写为 $C^{(1)} \otimes C^{(2)} \otimes \cdots \otimes C^{(k)}$。我们将 $C^{(i)}$ 分解为 $cnt_i$ 个秩一张量，得到 $\left(T_0^{(1)} + ... + T_{cnt_1-1}^{(1)}\right) \otimes \left(T_0^{(2)} + ... + T_{cnt_2-1}^{(2)}\right) \otimes ... \otimes \left(T_0^{(k)} + ... + T_{cnt_k-1}^{(k)}\right)$。

之后可以设计下面两种算法基于这个式子进行求解。

## 1.3 高维广义卷积的算法设计

在 1.3.1 中，我们将逐步得到 FMT/FWT 类算法，并在此过程中将求解卷积所涉及的三个变换转换为同样的形式。

在 1.3.2 中，我们将介绍“分治乘法”算法。

在 1.3.3 中，我们将从整体上介绍如何设计张量分解。

我们称 1.2 中介绍的三步中，$f, g$ 经过线性组合得到的数组分别为 $f', g'$，$f', g'$ 对位相乘得到的数组为 $mul$（本文中用 $mul = f' \cdot g'$ 表示）。

为了直观理解算法流程，我们提前对每一维所处的“状态”进行如下定义：

- 状态 A：该维大小为 $|U_i|$ 或 $|V_i|$，其中 $p$ 元素统计的是 $p$ 的值。
- 状态 B：该维大小为 $cnt_i$，每个元素都是初始元素的一种线性组合。
- 状态 C：该维大小为 $cnt_i$，每个元素都是某个 $f$ 的线性组合 $*\ g$ 的线性组合。
- 状态 D：该维大小为 $|W_i|$，其中 $p$ 元素统计的是所有 $x \circ y = p$ 的 $f_x * g_y$ 的和，也是一些 ($f$ 的线性组合 * $g$ 的线性组合) 的线性组合。

注意状态 A,B 的维是“一次式”，而状态 C,D 的维是“二次式”。

高维空间中每一维要么均为 A/B 状态，要么均为 C/D 状态。

将 A,B 状态的维用一个**二阶张量（$f$ / $g$, 元素）表示**，值为该元素包含的该 $f/g$ 的系数。A 状态对应的二阶张量为单位矩阵。

将 C,D 状态的维也用一个**二阶张量（$mul$, 元素）表示**，值为该元素包含的该 $mul$ 的系数。C 状态对应的二阶张量为单位矩阵。

根据前面的分析，表示整个高维空间的张量为表示每一维的张量的乘积。

### 1.3.1 FMT/FWT 类算法

我们对每一维进行张量分解后，考虑第 $i$ 维的 $\left(T_0^{(i)} + ... + T_{cnt_i-1}^{(i)}\right)$ 中的每个 $T_j^{(i)}$ 均可拆为 $A \circ B \circ C$。

我们将 $A_0^{(i)}, A_1^{(i)}, ..., A_{cnt_i-1}^{(i)}$ 竖着拼接成一个二阶张量 $MF^{(i)}$，其中 $MF_{p,q}^{(i)} = A_q^{(i)}{}_p$。

同样，将 $B_0^{(i)}, B_1^{(i)}, ..., B_{cnt_i-1}^{(i)}$ 竖着拼接成一个二阶张量 $MG^{(i)}$，其中 $MG_{p,q}^{(i)} = B_q^{(i)}{}_p$。

那么 $MF^{(i)}$ 就是 $f \to f'$ 中第 $i$ 维状态 B 的张量表示，$MG^{(i)}$ 同理。

考虑到 $A, B$ 均描述“新元素由原始元素按照什么系数组合得到”，而 $C$ 描述“原始元素按照什么系数分配给新元素”，因此需要“转置”：将 $C_0^{(i)}, C_1^{(i)}, ..., C_{cnt_i-1}^{(i)}$ **横着**拼接成一个二阶张量 $MH^{(i)}$，其中 $MH_{p,q}^{(i)} = C_p^{(i)}{}_q$。

于是 $MH^{(i)}$ 就是 $mul \to h$ 中第 $i$ 维状态 D 的张量表示。

将 $f, f'$ 都写成一行的矩阵，则 $f' = f * (MF^{(1)} \otimes MF^{(2)} \otimes ... \otimes MF^{(k)})$，其中 $*$ 为矩阵乘法。

同理，$g' = g * (MG^{(1)} \otimes MG^{(2)} \otimes ... \otimes MG^{(k)})$，$h = mul * (MH^{(1)} \otimes MH^{(2)} \otimes ... \otimes MH^{(k)})$，于是三个变换都被转换为同一种形式了。

该形式的求解方案为：先将 $k$ 个二阶张量都设为单位矩阵，之后再按任意顺序将 $k$ 个二阶张量依次修改为正确的 $MF/MG/MH$，并时刻维护最终的结果序列。

因为每次只将一维从单位矩阵修改为需要的张量（等价于从状态 A/C 修改为状态 B/D），因此是容易直接在最终序列上进行更新的。

如果所有操作顺序均为 $1, ..., k$，那么对于 $f \to f', g \to g'$ 的变换，状态变化过程为：

$$AA...A \to BA...A \to BBA...A \to ... \to BBB...BA \to BB...B$$

$f' \cdot g' \to mul$ 的过程，状态变化为：

$$BB...B \to CC...C$$

对于 $mul \to h$ 的变换，状态变化过程为：

$$CC...C \to DC...C \to DDC...C \to ... \to DDD...DC \to DD...D$$

三个过程的维度操作顺序都是任意的，需要根据每维初始的大小和最终的大小来综合决定维度操作顺序（并不一定是按照两者的比例升序/降序操作！）。

### 1.3.2 分治乘法

**分治乘法实质上是求解带顺序限制的“FMT/FWT 类算法”的一种递归实现**——要求 $f \to f', g \to g'$ 均按照 $p$ 的顺序操作，$mul \to h$ 按照 $p$ 的反序操作。

现在要计算 $f$ 与 $g$ 的卷积结果，但 $f$ 与 $g$ 均只有 $n-i+1$ 个维度，分别为 $p_i, p_{i+1}, ..., p_n$。

做法是先将 $f$ 与 $g$ 的在第 $p_i$ 维上线性组合，变成 B 状态，之后递归到 $cnt_{p_i}$ 个 $i+1$ 的子问题，求解完后，第 $p_i$ 维就变为 C 状态了，于是再将答案在这一维上线性组合一下就变为 D 状态了。

如果 $p$ 描述的顺序为 $1, 2, ..., n$，则状态总变化过程为：

$$AA...A \to BA...A \to ... \to BB...B \to CC...C \to CC...D \to ... \to DD...D$$

在某一维处理时的状态变化过程为：

$$B...BAA...A \to B...BBA...A \to C...CCD...D \to C...CDD...D$$

分治乘法不仅对三个变换的顺序产生了限制使得可能的复杂度提升，其结构还拥有局限性——由于其递归特性，在“叶子”处只能“单点”与“单点”相乘，导致无法实现部分后文提到的“对称卷积”与“线性组合归约”。

因此，最通用的写法还是直接按照设计思路，显式地执行 $f \to f'$，$g \to g'$，$f', g' \to mul$，$mul \to h$ 四个过程。

### 1.3.3 设计张量分解

我们需要对每一维进行张量分解。设这一维是一个 $I \times J \times K$ 的张量。

因为三阶张量较难画出，我们一般将其转为 $K$ 个 $I \times J$ 的目标张量。

张量分解的部分也可降为二维，变成：设计尽量少的 $I \times J$ 的秩一张量，使得它们可以通过**线性组合**得到 $K$ 个目标张量。

## 1.4 常见的高维广义卷积

在上一节中，我们得出：只要能够对每一维的目标张量进行张量分解，就可以快速求出该高维广义卷积的结果。而张量分解的大小越小，复杂度越低。

在本节中，我们将举一些经典的例子，通过张量分解得出卷积算法，并进一步得到其 FMT/FWT 类变换的表达式。我们还将探讨一些常见的 $p$ 进制卷积，以及 FFT/NTT。

### 1.4.1 OR 卷积

每一维的目标张量均为：

$$\boldsymbol{C_0} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{C_1} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

容易构造 2 个 $2 \times 2$ 秩一张量如下：

$$\boldsymbol{M_0} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{M_1} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$C_0 = M_0$，$C_1 = M_1 - M_0$，因此 $n$ 维 OR 卷积的复杂度为 $O(2^n n)$。

因为 $M_0 = [1,0] \circ [1,0], M_1 = [1,1] \circ [1,1]$，那么 $f, g$ 的线性变换方式相同，且最终 0 处的值是初始的 0，最终的 1 处的值初始 0/1 都会有贡献。

结合 $C_0 = M_0$，$C_1 = M_1 - M_0$，可以得到表达式：$f'_x = \sum_{i \subseteq x} f_i$，$g'_x = \sum_{i \subseteq x} g_i$，$h_x = \sum_{i \subseteq x} (-1)^{|x|-|i|} mul_i$。

### 1.4.2 AND 卷积

每一维的目标张量均为：

$$\boldsymbol{C_0} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad \boldsymbol{C_1} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

容易构造 2 个 $2 \times 2$ 秩一张量如下：

$$\boldsymbol{M_0} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \boldsymbol{M_1} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$C_0 = M_0 - M_1$，$C_1 = M_1$，因此 $n$ 维 AND 卷积的复杂度为 $O(2^n n)$。

通过与 OR 卷积类似的分析，可以得到表达式：$f'_x = \sum_{x \subseteq i} f_i, g'_x = \sum_{x \subseteq i} g_i, h_x = \sum_{x \subseteq i} (-1)^{|i|-|x|} mul_i$。

### 1.4.3 XOR 卷积

每一维的目标张量均为：

$$\boldsymbol{C_0}=\begin{bmatrix}1&0\\0&1\end{bmatrix},\quad \boldsymbol{C_1}=\begin{bmatrix}0&1\\1&0\end{bmatrix}$$

构造秩一张量的时候，容易考虑先令一个张量为全 1。这样只要构造出 $C_0$ 就可以用全 1 张量 $-C_0$ 构造出 $C_1$。

但是 $C_0$ 本身不是秩一张量，需要修改，但还得保证 $(0,0)$ 位置与 $(1,1)$ 位置相等，$(0,1)$ 位置与 $(1,0)$ 位置相等，且 $(0,0)$ 位置与 $(0,1)$ 位置不等。

于是想到引入 $-1$，得到以下 2 个二维秩一张量：

$$\boldsymbol{M_0}=\begin{bmatrix}1&1\\1&1\end{bmatrix},\quad \boldsymbol{M_1}=\begin{bmatrix}1&-1\\-1&1\end{bmatrix}$$

$C_0=\frac{M_0+M_1}{2}$，$C_1=\frac{M_0-M_1}{2}$，因此 $n$ 维 XOR 卷积的复杂度为 $O(2^n n)$。

同样，可得表达式：$f'_x=\sum_i(-1)^{|i\&x|}f_i$，$g'_x=\sum_i(-1)^{|i\&x|}g_i$，$h_x=\frac{1}{2^n}\sum_i(-1)^{|i\&x|}mul_i$。

### 1.4.4 $p$ 进制卷积

$n$ 位 $p$ 进制卷积等价于 $n$ 维，每维大小为 $p$ 的高维广义卷积。

我们以 $p=3$ 为例介绍 $p$ 进制 max、min、不进位加法卷积。张量分解的方式与二进制类似。

max 卷积每一维的目标张量：

$$\boldsymbol{C_0}=\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix},\quad \boldsymbol{C_1}=\begin{bmatrix}0&1&0\\1&1&0\\0&0&0\end{bmatrix},\quad \boldsymbol{C_2}=\begin{bmatrix}0&0&1\\0&0&1\\1&1&1\end{bmatrix}$$

构造的 3 个秩一张量为：

$$\boldsymbol{M_0}=\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix},\quad \boldsymbol{M_1}=\begin{bmatrix}1&1&0\\1&1&0\\0&0&0\end{bmatrix},\quad \boldsymbol{M_2}=\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

min 卷积每一维的目标张量：

$$\boldsymbol{C_0}=\begin{bmatrix}1&1&1\\1&0&0\\1&0&0\end{bmatrix},\quad \boldsymbol{C_1}=\begin{bmatrix}0&0&0\\0&1&1\\0&1&0\end{bmatrix},\quad \boldsymbol{C_2}=\begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix}$$

构造的 3 个秩一张量为：

$$\boldsymbol{M_0}=\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix},\quad \boldsymbol{M_1}=\begin{bmatrix}0&0&0\\0&1&1\\0&1&1\end{bmatrix},\quad \boldsymbol{M_2}=\begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix}$$

不进位加法卷积每一维的目标张量：

$$\boldsymbol{C_0}=\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix},\quad \boldsymbol{C_1}=\begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix},\quad \boldsymbol{C_2}=\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}$$

考虑设计的所有秩一张量都一定满足 $(0,0)*(1,2)*(2,1)=(0,1)*(1,0)*(2,2)=(0,2)*(1,1)*(2,0)$，容易想到“轮换”。

再配合目标张量中的 1 都恰好处于不同行不同列，容易想到使用单位根构建秩一张量，并使用单位根反演将秩一张量线性组合成 $C_0,C_1,C_2$。因此得到如下分解：

$$\boldsymbol{M_0}=\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix},\quad \boldsymbol{M_1}=\begin{bmatrix}1&\omega&\omega^2\\\omega&\omega^2&1\\\omega^2&1&\omega\end{bmatrix},\quad \boldsymbol{M_2}=\begin{bmatrix}1&\omega^2&\omega\\\omega^2&\omega&1\\\omega&1&\omega^2\end{bmatrix}.$$

其中 $\omega$ 为 3 次单位根。通过单位根反演，可以得出：

$$C_0=\frac{M_0+M_1+M_2}{3},C_1=\frac{M_0+\omega^2M_1+\omega M_2}{3},C_2=\frac{M_0+\omega M_1+\omega^2M_2}{3}.$$

$n$ 位 $p$ 进制 min，max 卷积的复杂度均为 $O(p^n n)$，对应的线性变换是在进行**高维前缀和**与**高维差分**。

对于不进位加法卷积，如果是求模意义下的结果且模意义下存在 $p$ 次单位根，那么就直接将 $\omega$ 变为模意义下的 $p$ 次单位根即可。如果不存在，则需要使用 double 或 long double 存储实部和虚部。两种方法均可做到复杂度 $O(p^{n+1}n)$。

实质上，$p$ 进制不进位加法卷积就是对每一位进行长度为 $p$ 的循环卷积 DFT 与 IDFT。

熟知的 FFT 与 NTT 求解循环卷积与 1 位 $2^k$ 进制不进位加法卷积一致，DFT 求点值的过程就是 $f\to f'$，$g\to g'$ 寻找线性组合的过程，IDFT 的过程就是 $mul\to h$ 重分配答案的过程。

FFT 只是使用分治和蝴蝶变换加速了直接根据张量分解暴力实现的 DFT/IDFT 过程，将序列构成的 $1\times 2^k$ 矩阵乘 $2^k\times 2^k$ 的贡献矩阵的复杂度降到了 $O(2^k k)$。

使用 Chirp-Z 变换可以在 $O(n\log n)$ 的复杂度内求任意 $n$ 的 DFT/IDFT 结果，进而将不进位加法卷积的复杂度优化为 $O(p^n n\log p)$。

**例题 1.1.** 欧伊昔[1]

[1]QOJ9562

给出一个 $3\times 3$ 的，位运算表 $op$，$op$ 的每个元素都是 $0,1,2$ 中的一个。

给出两个长度为 $3^n$ 的序列 $a,b$，求它们在该位运算下的卷积结果 $c$。

$1\le n\le 11$，$op$ 随机生成。

我们希望求出运算表 $op$ 的张量分解，但求最小张量分解是 NP-hard 的。

考虑暴力枚举一些 $3\times 3$ 的秩一张量，并判断它们是否可以通过线性组合得到需要的 3 个 $3\times 3$ 的目标张量。

判定问题可以直接高斯消元：如果有 $k$ 个秩一张量，则对每个目标张量，都可设计 $k$ 个变量和 9 个方程，需要判断是否有解。

3 个目标张量对应的方程只有结果值不同，因此可以并行消元，优化一些常数。

观察经典卷积问题所构造的所有二维秩一张量，除了不进位加法卷积之外，均有如下性质：

- 全 1 张量最常见。
- 只有 $0,1$ 的张量也比较常见。
- 所有张量都仅包含 0、1、$-1$。

因此我们将全 1 的张量放在第一个，只包含 $0,1$ 的秩一张量紧随其后，最后放所有包含 0、1、$-1$ 的秩一张量。

张量 $T$ 与张量 $-T$ 只需保留一个，全 0 张量不需保留。加入这个优化后，总共仅有 $cnt=169$ 个满足要求的秩一张量。

枚举的时候依次枚举最靠左、次靠左、...、最靠右的张量。因为全 1 张量最常见，因此这样枚举有很大概率枚举量仅 $O(cnt^3)$。

在本题中，按照上述方式进行张量分解可以快速找到一组大小为 4 的张量分解。因此在不考虑张量分解的复杂度时，总复杂度为 $O(4^n)$。

## 2 对称卷积

**定义 2.1** (对称卷积)**.** 对于一个 $A*B=C$ 的卷积问题，其中要求该卷积的 $A,B,C$ 的维数相同且每一维大小相同。

如果我们可以设计两个线性变换——$T,invT$，使得 $T(A)*_\circ T(B)=T(C)$ 且 $invT(T(A))=A$，其中 $*_\circ$ 为一种卷积，则我们称该卷积为“**在 $\circ$ 卷积意义下的对称卷积**”。

注意对称卷积不一定是“对称”的，也就是不一定具有交换律。原因是 $\circ$ 卷积不一定具有交换律。

对于一个对称卷积，我们将 $T$ 称为其“正变换”，$invT$ 称为其“逆变换”。

**对称卷积归约：**如果我们能够设计 $\circ$ 卷积意义下的对称卷积的 $T$ 变换与 $invT$ 变换，则可以在执行 $T$ 变换与 $invT$ 变换的复杂度内，将原始卷积问题归约为（可能不同规模的）$\circ$ 卷积问题。

## 2.1 逆变换求解

对逆变换的限制只有 $invT(T(A)) = A$。考虑若已经有了一个满足 $T(A) *_\circ T(B) = T(A * B)$ 的正变换，该如何判断是否存在 $invT$，以及如果存在，该如何求解 $invT$ 呢？

将 $A$ 中每个元素都设为一个未知数，$T(A)$ 中的每个元素都构成一个“**求逆方程**”。

那么只要方程组没有自由元，也就是存在未知数数量个线性无关方程，则直接高斯消元即可得到 $invT$。

更加简洁的，令 $A, T(A)$ 的每维大小乘积分别为 $N, M$，那如果将 $A$ 写成一个 $1 \times N$ 的矩阵 $M_0$，$T$ 写成一个 $N \times M$ 的贡献系数矩阵 $M_1$，$invT$ 写成一个 $M \times N$ 的贡献系数矩阵 $M_2$。

那么 $M_2$ 需要满足：$\forall M_0, M_0 * M_1 * M_2 = M_0$，即：$M_1 * M_2 = I_N$。

特殊的，如果 $T$ 变换矩阵 $M_1$ 可以表示为 $M_1^{(1)} \otimes M_1^{(2)} \otimes \cdots \otimes M_1^{(k)}$，那么设计 $M_2$ 的时候可以直接令 $M_2 = M_2^{(1)} \otimes M_2^{(2)} \otimes \cdots \otimes M_2^{(k)}$，并要求 $M_1^{(i)} * M_2^{(i)} = I$，再使用“求逆方程”解出 $M_2^{(i)}$ 即可。

## 2.2 对称卷积连乘

对于一个对称卷积，$T(A) *_\circ T(B) = T(A * B)$，所以 $T(A) *_\circ T(B) *_\circ T(C) = T(A * B) *_\circ T(C) = T(A * B * C)$。

进一步归纳可得：$T(A_1) *_\circ T(A_2) *_\circ \cdots *_\circ T(A_m) = T(A_1 * A_2 * ... * A_m)$。

又因为 $invT(T(A)) = A$，所以求 $A_1 * ... * A_m$ 的问题被转换为求 $invT(T(A_1) *_\circ T(A_2) *_\circ \cdots *_\circ T(A_m))$ 进而转换为求 $T(A_1) *_\circ T(A_2) *_\circ \cdots *_\circ T(A_m)$ 的问题。

如果原本的卷积难以处理，但是 $\circ$ 卷积好处理（例如是对位相乘），则这样转换就有很大优势。

**例题 2.1.** New Year and Boolean Bridges[2]

定义 $f(u, v)$ 表示是否存在从节点 $u$ 到节点 $v$ 的有向路径。对于每对不同点 $u, v$，如下三个中至少有一个为真：

- $f(u, v)$ AND $f(v, u)$
- $f(u, v)$ OR $f(v, u)$
- $f(u, v)$ XOR $f(v, u)$

给定任意两点之间成立的某**一个**条件，求是否存在满足要求的有向图。

若不存在，输出 $-1$；否则，输出最少需要多少条边可以满足条件。

$1 \le n \le 47$。

[2] CF908H

首先容易发现，满足题意的边数最少的图在 *scc* 缩点后一定为一个链。

如果已知 *scc* 构成的链中每个 *scc* 的大小，则最少边数为多少？

一个大小为 $k \geq 2$ 的 *scc* 需要 $k$ 条边才能形成 *scc*，而大小为 1 的 *scc* 则不用一条边。因此最少边数为 $n+ \geq 2$ 个点的 *scc* 数量 $-1$，于是转换为最小化点数 $\geq 2$ 的 *scc* 数量。

对于 AND 要求，将边的两端缩起来，OR 的要求天然满足，XOR 的要求则是边的两端不能在一个 *scc* 中。

那么容易发现，缩完 AND 边后合并 *scc* 只有可能使本身合法的变不合法，而不会使不合法的变合法，于是直接让所有大小为 1 的 *scc* 单独成块。

我们在缩完所有 AND 边后，将每个大小 $\geq 2$ 的块重标号为一个点，并对所有跨越两个不同大小 $\geq 2$ 的块的 XOR 边，将两端所属的新点之间连边。如果某个块内部存在 XOR 边，则不合法。

因为目前所有点原本的大小就 $\geq 2$，所以几个点缩成一个的大小也 $\geq 2$。于是问题转换为：计算最小的 $k$，使得存在 $S_1, S_2, ..., S_k$，任意两个 $S_i$ 不交，所有 $S_i$ 的并为全集，且每个 $S_i$ 都构成一个独立集。

那么在处理出所有合法独立集 $S$ 之后，只需要计算最少子集卷积多少次可以变为全集即可。暴力实现复杂度为 $O(2^{\lfloor \frac{n}{2} \rfloor} n^3)$，但有如下优化：

1. 子集卷积可以变为 OR 卷积，因为若两个集合有交，将交从其中一个集合中去掉显然仍合法。

2. OR 卷积是对位相乘意义下的对称卷积，于是先进行变换 $T$，之后使用对位相乘即可快速得出 $k$ 次乘法的结果 $Mul$ 的 $T(Mul)$ 值。

3. 判定 $k$ 是否合法不需要求 $invT(T(Mul))$ 得到所有集合的系数，只需要得知全集的系数，于是直接使用 $invT$ 的表达式即可线性判定。

加入以上 3 个优化后，复杂度降为 $O(2^{\lfloor \frac{n}{2} \rfloor} n)$。

## 2.3 稀疏对称卷积连乘

对于稀疏对称卷积连乘，我们一般会考虑直接计算乘积在 $T$ 运算后的每个位置上的值，之后再 $invT$ 一下即可得到最终答案。

**例题 2.2.** 算力训练[3]

给定 $n$ 个数 $a_1, ..., a_n$，满足 $0 \leq a_i < k^m$。

求 $\prod_{i=1}^{n}(1 + x^{a_i})$ 的每一项系数，其中 $x^p$ 与 $x^q$ 相乘得到 $x^r$，满足 $r$ 为 $p, q$ 进行 $m$ 位 $k$ 进制不进位加法的结果。

$n \leq 10^6$，$k = 5, m \leq 7$ 或 $k = 6, m \leq 6$。

[3]洛谷 P5577

根据 $k$ 进制不进位加法的张量分解，可以得到：正变换中 $S$ 位置对 $T$ 位置产生的贡献系数为 $\omega^{\sum_{j=0}^{m-1} S_j * T_j}$，其中 $S_j, T_j$ 分别为 $S, T$ 的 $k$ 进制表示中第 $j$ 位的值，$\omega$ 为 $k$ 次单位根。

那么 $1 + x^{a_i}$ 执行正变换后 $S$ 位置的值为：$1 + \omega^{\sum_{j=0}^{m-1} a_{ij} * S_j}$。

我们显然不能对每个 $1 + x^{a_i}$ 都进行正变换，于是考虑**同时进行所有 $1 + x^{a_i}$ 的正变换**，即：快速计算每个位置 $S$ 上到底乘了多少个 $1 + \omega^i, \forall i \in [0, k-1]$。

我们令 $f_i = a$ 序列中 $i$ 的出现次数。对 $f$ 进行 $k$ 进制不进位加法正变换后 $S$ 位置多项式 $\omega^i$ 的系数即为 $S$ 位置最终正变换总乘积中 $(1 + \omega^i)$ 的指数。

使用快速幂/光速幂/预处理即可快速计算 $(1 + \omega^i)^p$，该部分的复杂度根据选择的方法不同而不同。其余部分的总复杂度为 $O(k^{m+2}m)$。

## 2.4 对称卷积逆、ln、exp

给定 $A$，你希望计算 $B$，使得 $A * B = I$。

因为 $T(A) *_\circ T(B) = T(I)$，所以如果 $\circ$ 卷积具有逆，则可以求出 $T(B)$，进而通过 $invT(T(B))$ 得到真实的 $B$。

于是我们将对原本的卷积求逆转换为对 $\circ$ 卷积求逆。类似的，我们还可以将对原本的卷积求 ln、exp 转换为对 $\circ$ 卷积求 ln、exp。

需要注意的是，不是所有的对称卷积都具有逆、ln、exp，因为一些 $T(A)$ 在 $\circ$ 卷积意义下不存在逆、ln、exp。当然，也不是只有对称卷积才可能具有逆、ln、exp。

# 3 手动张量分解的方法

虽然最小张量分解是 NP-hard，但还是存在一些启发式算法求张量分解。这不在我们的讨论范围内，感兴趣的读者可以阅读本论文中的参考文献。

在本节中，我们将聚焦手动张量分解的一些方法。

## 3.1 边界秩

往张量分解中引入 $x$，并在 $x \to 0$ 的时候，所有分解出的秩一张量的和 $\to$ 目标张量。

这种情况下，分解出的秩一张量数量的最小值被称为目标张量的“边界秩”。

其一种等价表述为：往张量分解中引入 $x$，使得所有元素**均为多项式**，并在线性组合中可以自如地乘除 $x$，最后每个位置只保留常数项系数即为目标张量。

在实际运用过程中，我们一般不进行“除 $x$”的操作，导致最终答案不再是每个位置的常数项系数，而是每个位置“**有参数的最低次项系数**”。设最后 $i$ 位置关心的是 $x^{Base_i}$ 的系数。

使用边界秩会导致目标张量 $C_i$ 的修改：每个位置的 $x^0, x^1, ..., x^{w_i-1}$ 的系数均为 0，且 $x^{w_i}$ 的系数为其原始值。$w_i$ 的值可以任意设定。

边界秩的优势是，对 $x^{>w_i}$ 的系数没有限制，因此更容易满足要求。

以子集卷积为例，每一维的目标二阶张量均为：

$$\boldsymbol{C_0} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{C_1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

因为总共只有 3 个 1，所以分解为 3 个秩一张量是容易的，对应复杂度 $O(3^n)$。

我们发现无法做到 2 个普通秩一张量的分解，于是考虑设计边界秩。

我们将目标张量修改为：

$$\boldsymbol{C_0} = \begin{bmatrix} x & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{C_1} = \begin{bmatrix} 0 & x \\ x & x^2 \end{bmatrix}$$

或

$$\boldsymbol{C_0} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{C_1} = \begin{bmatrix} 0 & x \\ x & x^2 \end{bmatrix}$$

构造如下两个秩一张量即可通过线性组合得到新的 $C_0, C_1$：

$$\boldsymbol{M_0} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{M_1} = \begin{bmatrix} 1 & x \\ x & x^2 \end{bmatrix}$$

如果使用第二种新目标张量，可得表达式：$f'[S] = \sum_{T \subseteq S} x^{|T|} f[T]$，$g'[S] = \sum_{T \subseteq S} x^{|T|} g[T]$，$h[S] = \sum_{T \subseteq S} (-1)^{|S|-|T|} mul[T]$，$h[S]$ 的最终值为其 $x^{|S|}$ 的系数。

这样也可以得到子集卷积与 OR 卷积的关联——$h[z]$ 的值为 $|x| + |y| = |z|$ 且 $x \cup y = z$ 的 $f[x] * g[y]$ 的和。

因为总共需要进行 $O(2^n n)$ 次多项式求和以及 $O(2^n)$ 次多项式乘法，多项式长度为 $O(n)$，所以复杂度为 $O(2^n n^2)$。

为什么只关心最低次项的系数却还要维护多项式，而不是只维护最低次项及其系数？

因为算法只保证结束的时候最低次项的系数为答案，而中途最低次项到最后可能消成 0 了，非最低次项就有用了。因此不能只记录最低次项的信息。

使用边界秩后，第 2 节探讨的“对称卷积”定义中“$invT(T(A)) = A$”的要求变为了：$invT(T(A))$ 的每个位置 $i$ 均保留 $x^{Base_i}$ 的系数后 $= A$。

而构造逆变换的时候，“$M_1 * M_2 = I_N$”的要求变为：$M_1 * M_2$ 的第 $i$ 列每个位置只保留**整列最低次项**的系数后，得到的矩阵为 $I_N$。容易发现，$Base_i$ 就是第 $i$ 列“整列最低次项”的次数。

边界秩除了能够降低复杂度，还可能使没什么性质的卷积变为对位乘法意义下的对称卷积。而因为每个元素都是一个多项式，则在满足 $T(A)$ 每个位置的多项式常数项要求的情况下即可轻松求解逆、ln、exp。

子集卷积为对位乘法意义下的对称卷积，因为上面的做法就给出了一组合法的正逆变换。

那么求解子集卷积的逆/ln/exp，只需要先执行子集卷积的正变换，之后对每个位置的多项式求逆/ln/exp，最后再执行逆变换，并对 $i$ 位置保留 $x^{Base_i}$（$Base_i$ 与选择的新目标张量有关）的系数即可。

## 3.2 张量归约

对于复杂的张量，我们一般可以考虑将张量归约为简单的张量。

### 3.2.1 运算规律归约

使用“运算规律”归约：例如归约为张量直和、直积、乘积、和等。

对于“每维独立”的张量，可以使用直积、乘积等划分到每一维上，方便进一步分解。这在前面的高维广义卷积算法设计中也有所涉及。

对于“两部分独立”的张量，可以使用直和，直接将两个部分拼接起来。

有时，不存在明显的“每维独立”或“两部分独立”时，也可以尝试强行分拆。

例如：大整数乘法的 Karatsuba 算法与矩阵乘法的 Strassen 算法都是强行将张量分拆成一些小张量的乘积，也就是将大小为 $2^k$ 的一维拆成 $k$ 维，每维大小为 2。

使用 3.1 中边界秩的技巧以及本节的运算规律归约，可以进一步得到矩阵乘法的大约 $O(n^{2.780})$ 复杂度、$O(n^{2.522})$ 复杂度与 $O(n^{2.496})$ 复杂度的做法。感兴趣的读者可以阅读参考文献中的 [2]、[3] 与 [4]。

### 3.2.2 线性组合归约

与“对称卷积”类似的，我们可以对 1.2 中描述的高维广义卷积的设计思路加以扩展：

将第二步的 $f', g'$ 对位相乘得到 $mul$ 的过程变为将 $f', g'$ 通过 $\circ$ 卷积得到 $mul$，其中该卷积也得是高维广义卷积。

那么就在计算 $f \to f'$、$g \to g'$ 与 $mul \to h$ 的复杂度内将 $< f, g, h >$ 张量归约为 $< f', g', mul >$ 张量了。

## 3.3 群表示法

如果每一种值都可以表示一种“变换”，且变换构成群，并且两个值卷积的结果与它们变换复合的结果一致。分解这种卷积对应的张量时可以考虑使用**群表示法对张量进行归约**。

一个群表示实质上就是将所有值映射到 $d$ 维空间中的一个线性变换，使得“卷积”与“线性变换复合”相对应。

$d$ 维空间的线性变换对应一个 $d\times d$ 的矩阵，“线性变换复合”对应“矩阵乘法”。

一个 $d$ 维群表示会贡献 $d^2$ 个“求逆方程”，而任意一个大小为 $n$ 的有限群都存在一组不可约表示，维度分别为 $d_1,...,d_m$，满足 $\sum d_i^2=n$。于是一定存在合法的逆变换，也就是可以构成（$m$ 个矩阵乘法张量的直和对应卷积）意义下的对称卷积。

该对称卷积的正变换为：将 $n$ 个元素线性组合，并写成 $m$ 个边长分别为 $d_1,d_2,...,d_m$ 的矩阵；逆变换为：将矩阵分割，扣下来，排成一个长度为 $n$ 的序列。

因此所有有限群复合构成的张量都可以设计一组群表示，并通过对称卷积，将复杂的群张量归约为一系列矩阵乘法张量。

群不具有乘法交换律，恰好与矩阵乘法不具有乘法交换律对应。不过因为矩阵可以快速乘法和求逆，所以使用群表示法设计的卷积也可以以较高的效率进行卷积和求逆。

**例题 3.1.** djq 学生物[4]

有 $n$ 个长度为 3 排列，初始均为 $(1,2,3)$。

给定 $m$ 个操作，其中有 1 个为终止操作，其余 $m-1$ 个为 $n$ 个长度为 3 的排列的序列（一共有 $6^n$ 种排列的序列）。

只要没有终止，就不断从 $m$ 个操作中等概率随机一个操作执行：

- 如果操作为“终止操作”，则立即终止。
- 否则，$\forall i\in[1,n]$，将第 $i$ 个排列复合上操作排列序列的第 $i$ 个排列——$s_1s_2s_3$ 变为 $s_{p_1}s_{p_2}s_{p_3}$。

用 $a_i=0...5$ 分别表示第 $i$ 个排列为 $(1,2,3),(1,3,2),(2,1,3),(2,3,1),(3,1,2),(3,2,1)$。

用 $n$ 位六进制数 $\overline{a_na_{n-1}a_{n-2}\cdots a_1}$ 来表示一个排列的序列。

给定 $c_0,c_1,...,c_{6^n-1}$ 表示每种排列的序列在操作中的出现次数，$m=\sum c_i+1$。求终止时每种局面（共 $6^n$ 种）的出现概率，对 998244353 取模。

$1\le n\le 8$，$998244353\nmid m$，$c_i\in[0,998244353)$。

令 $f_S$ 表示在终止前，状态 $S$ 期望出现的次数，$p_S$ 表示一次操作选择 $S$ 这个排列序列的概率。终止时局面为 $S$ 的概率就是 $\frac{1}{m}f_S$。

$f_S$ 的转移考虑除了初始状态外，每个状态都有前继状态，于是枚举状态 $S$ 的前继状态，可得转移式：$f_S=[S=0]+\sum_{x\circ y=S}f_xp_y$。于是 $f=1+f*_\circ p$，即：$f=\frac{1}{1-p}$。

[4]QOJ4907

因为要求逆，所以考虑设计对称卷积。而该运算每位独立（一位就是一个排列，每个排列独立），于是只需要解决一位张量的分解即可通过乘法得到整个张量的分解。

根据题意，需要分解的张量为：

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 4 & 5 & 2 & 3 \\ 2 & 3 & 0 & 1 & 5 & 4 \\ 3 & 2 & 5 & 4 & 0 & 1 \\ 4 & 5 & 1 & 0 & 3 & 2 \\ 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

一位的所有元素实际上构成 $S_3$ 置换群，卷积为置换的复合，于是考虑通过寻找群表示法进行张量分解。

对于一维的群表示，有 1 以及 $(-1)^{inv(p)}$ 两个不可约表示，其中 $inv(p)$ 为 $p$ 的逆序对数量。

对于二维的群表示，我们要针对 $S_3$ 设计二维线性变换，可以考虑在二维平面中找三个点 $A, B, C$，使得它们的横坐标和为 0，纵坐标和为 0，并将每个变换都对应为三个点之间的置换所需要的线性变换。

比如令 $A = (1, 0), B = (0, 1), C = (-1, -1)$。

那么：

$$\boldsymbol{M_0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{M_1} = \begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix}, \quad \boldsymbol{M_2} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

$$\boldsymbol{M_3} = \begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix}, \quad \boldsymbol{M_4} = \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix}, \quad \boldsymbol{M_5} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix}.$$

空间的线性变换可以被表示为矩阵左乘，因此 $M_x$ 变换的基础上进行 $M_y$ 变换，等价于 $M_y * M_x$ 变换。

于是，我们使用 2 组一维群表示和 1 组二维群表示，将题目中 $6^n \times 6^n$ 的张量转换为 $\binom{n}{i}2^i$ 个边长为 $2^{n-i}$ 的矩阵乘法张量。

那么卷积求逆就被转换为矩阵求逆。如果暴力实现矩阵求逆，则复杂度为 $O(10^n)$。若使用 $O(n^\omega)$ 的矩阵求逆，则可以做到 $O((2^\omega + 2)^n) \approx O(7.1743^n)$ 的复杂度。

## 4 高维下闭集合上的扩展

我们将高维空间扩展为高维下闭集合，也就是“不完整但连续”的高维空间。

高维下闭集合上的卷积问题的一般处理方式为：先将其“补全”成完整高维空间进行分析，之后每次只操作“重要”的位置以保证复杂度不退化。

正因如此，部分高维下闭集合上的卷积问题由于“重要”的位置很多，所以无法做到很优的复杂度。

其实我们可以进一步扩展到一般的高维偏序集。不过对于大部分卷积问题，都至少需要将其补全成下闭集合进行求解。因此该扩展意义不大，我们不进行讨论。

## 4.1 定义

**定义 4.1** (高维下闭集合)**.** 令 $P = \mathbb{N}^k$，在其上定义偏序关系：$a \le b \iff a_i \le b_i, \forall i = 1, 2, ..., k$。

高维下闭集合为一个 $S \subseteq P$，满足 $\forall x \in S, \forall y \in P, y \le x \Rightarrow y \in S$。

其等价表述为：$\forall x \in S$，所有每一维都不超过 $x$ 对应维的非负整数坐标 $y$ 均有 $y \in S$。

维数相同的高维下闭集合的交、并均仍然为同一维数的高维下闭集合。

**定义 4.2** (高维下闭集合上的广义卷积)**.** 令 $f : U \to F, g : V \to F$，卷积结果为 $h : W \to F$，其中 $F$ 为一种域，$U, V, W$ 为高维下闭集合。

对每一维 $i$，令第 $i$ 维贡献函数 $c_i : U_i \times V_i \times W_i \to F$，其中 $U_i, V_i, W_i$ 分别为集合 $U, V, W$ 中所有元素第 $i$ 维坐标构成的集合。

令每维独立贡献函数 $c : \left(\prod_{i=1}^{k} U_i\right) \times \left(\prod_{i=1}^{k} V_i\right) \times \left(\prod_{i=1}^{k} W_i\right) \to F$ 满足：

$$c(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \prod_{i=1}^{k} c_i(u_i, v_i, w_i), \qquad \forall \mathbf{u} \in \prod_{i=1}^{k} U_i, \mathbf{v} \in \prod_{i=1}^{k} V_i, \mathbf{w} \in \prod_{i=1}^{k} W_i.$$

那么 $f$ 与 $g$ 在 $c$ 贡献函数下的卷积结果 $h$ 满足：

$$h(\mathbf{w}) = \sum_{\mathbf{u} \in U} \sum_{\mathbf{v} \in V} f(\mathbf{u})\, g(\mathbf{v})\, c(\mathbf{u}, \mathbf{v}, \mathbf{w}), \qquad \forall \mathbf{w} \in W.$$

高维下闭集合上的广义卷积等价于：先将 $f, g$ 的定义域 $U, V$ 补全成完整高维空间，之后对 $f, g$ 进行高维广义卷积，最后切出一个下闭集合 $W$ 作为定义域，得到最终的 $h$。

**定义 4.3** (质因子指数集)**.** 设 $n \in \mathbb{Z}^+$，记 $p_1, p_2, \ldots, p_k$ 为 $\le n$ 的所有质数。

对于 $m \in \{1, 2, ..., n\}$，将其唯一分解为

$$m = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}.$$

所有 $m \in \{1, 2, ..., n\}$ 唯一分解得到的 $(a_1, a_2, ..., a_k)$ 构成的集合被称为质因子指数集。

对于质因子指数集中的一个元素 $(a_1, a_2, ..., a_k)$，我们有时会用该元素对应的 $m$ 来表示。

容易发现，质因子指数集为一个高维下闭集合。

## 4.2 高维下闭集合上的前缀和、后缀和、差分

我们在进行高维 max 卷积的时候，$f \to f'$ 与 $g \to g'$ 的过程就是高维前缀和，而 $mul \to h$ 的过程就是高维差分。

高维前缀和的算法流程为逐维进行前缀和。因此在高维下闭集合上，考虑将其补全成完整高维空间后逐维进行前缀和。

进行高维前缀和的过程中，对 $x$ 产生影响的所有 $y$ 均满足 $y \le x$，因此不会出现不在集合中的 $x$ 对在集合中的 $y$ 的值产生贡献。

同理，对于高维后缀和，所有不在集合中的元素对应的权值永远为 0。

因此高维下闭集合上的前缀和/后缀和在求解的时候，只需要执行那些 $x,y$ 均在集合中的 $f_x \to f_y$ 转移。前缀/后缀的差分与此类似。

以大小为 $n$ 的质因子指数集上的前缀和（狄利克雷前缀和）为例：$g_i = \sum_{j|i} f_j$。

因为每个维度都是质因子，所以可以按任意顺序枚举 $\le n$ 的所有质因子 $p$，并执行 $f_x \to f_{px}$。最后令 $g_i = f_i$，复杂度 $O(n \log\log n)$。

## 4.3 高维下闭集合上的 max、min 卷积

我们令 $U,V,W$ 分别为参与卷积的 $f,g$ 与卷积结果 $h$ 对应的下闭集合。

max 卷积前缀和：需要均在 $W$ 上进行前缀和（$f',g' : W \to F$），$f,g$ 分别仅有 $U \cap W$ 与 $V \cap W$ 上的值有用。

max 卷积前缀差分：在 $W$ 上进行前缀差分。

min 卷积后缀和：分别在 $U,V$ 上进行后缀和。

min 卷积后缀差分：需要在 $U \cap V$ 进行后缀差分，初始非 0 值也都在 $U \cap V$ 上。

注意 max 卷积前缀和不能分别在 $U,V$ 上进行，因为 $W$ 上的前缀和都是关心的；min 卷积后缀差分也不能在 $W$ 上进行，因为不在 $W$ 中但在 $U \cap V$ 中的元素也是有值的。

以 $|U| = |V| = |W| = n$ 的质因子指数集上卷积为例，其 max、min 卷积：$h_i = \sum_{lcm(j,k)=i} f_j g_k$、$h_i = \sum_{gcd(j,k)=i} f_j g_k$ 都可以通过前缀和/后缀和 + 前缀差分/后缀差分做到复杂度 $O(n \log\log n)$。

## 4.4 高维下闭集合上的子集卷积

高维下闭集合上的子集卷积要求对于每一维，$x,y$ 必须有至少一者为 0，此时 $f_x * g_y$ 才能对 $h_{x+y}$ 产生贡献（其中“+”为 $k$ 维坐标对维相加）。

我们仿照普通子集卷积使用边界秩，最终的算法可以等价于：初始令 $f_i = f_i * x^{\omega(i)}$，$g_i = g_i * x^{\omega(i)}$，之后对 $f,g$ 做 max 卷积，最后保留 $h_i$ 的 $x^{\omega(i)}$ 系数。$\omega(i)$ 为 $i$ 的非 0 维个数。

仍以 $|U| = |V| = |W| = n$ 的质因子指数集上卷积为例，其子集卷积为：$h_i = \sum_{jk=i,\ \gcd(j,k)=1} f_j g_k$。在此例子中，因为每一维都表示一个不同的质因子，所以 $\omega(i)$ 的具体含义为 $i$ 包含的不同质因子数量。

使用类似普通子集卷积的传统实现方式，则 $f \to f'$，$g \to g'$，$mul \to h$ 都为进行 $O(\max_{i=1}^{n}\{\omega(i)\})$ 次质因子指数集上前缀和/差分，而 $mul = f' \cdot g'$ 为进行 $O(n)$ 次长度为 $O(\max_{i=1}^{n}\{\omega(i)\})$ 的多项式乘法。

因为 $O(\max_{i=1}^{n}\{\omega(i)\}) = O(\frac{\log n}{\log\log n})$，所以 $f \to f'$，$g \to g'$，$mul \to h$ 的复杂度为 $O(n\log n)$。$mul = f' \cdot g'$ 暴力卷积的复杂度为 $O(n\frac{\log^2 n}{(\log\log n)^2})$，使用 FFT/NTT 卷积可以优化为 $O(n\log n)$，得到总复杂度 $O(n\log n)$。

但如果直接维护每个位置的多项式，那么容易发现，$f, g, f', g', mul, h$ 中 $i$ 位置多项式的次数始终都 $\le O(\omega(i))$。

在狄利克雷前缀和/差分中，$i$ 位置会被更新 $O(\omega(i))$ 次，而每次更新 $i$ 都是使用一个被 $i$ 偏序的 $j$（$j \le i$）。因为 $\omega(j) \le \omega(i)$，所以单次更新 $i$ 的复杂度 $\le O(\omega(i))$。因此 $f \to f'$，$g \to g'$，$mul \to h$ 的复杂度均为 $O(\sum_{i=1}^{n}\omega(i)^2)$。

$mul = f' \cdot g'$ 中 $i$ 位置就是两个 $O(\omega(i))$ 的多项式乘法。使用暴力乘法可得到 $O(\sum_{i=1}^{n}\omega(i)^2)$ 的复杂度。

至此可得总复杂度 $O(\sum_{i=1}^{n}\omega(i)^2) = O(n(\log\log n)^2)$。

这也说明了对于边界秩引出的算法，最通用复杂度最低的实现方式为直接维护多项式，而非像传统子集卷积那样拆成多个不带变量的问题。

## 4.5 正交狄利克雷卷积与狄利克雷卷积

正交狄利克雷卷积：$h_i = \sum_{jk=i, gcd(j,k)=1} f_j g_k$。

狄利克雷卷积：$h_i = \sum_{jk=i} f_j g_k$。

其中序列 $f, g, h$ 均为从 1 开始标号，且长度为 $n$。

我们在 4.4 中介绍了正交狄利克雷卷积的 $O(n(\log\log n)^2)$ 复杂度的简单做法。

下面我们将狄利克雷卷积与正交狄利克雷卷积进行相互归约。

求狄利克雷卷积的时候可以枚举 $gcd(j,k) = d$，并转换为长度为 $\lfloor \frac{n}{d^2} \rfloor$ 的正交狄利克雷卷积问题。

于是狄利克雷卷积可以转换为求总长度为 $\sum_{i=1}^{n}\lfloor \frac{n}{i^2} \rfloor = O(n)$ 的 $n$ 次正交狄利克雷卷积。

类似的，正交狄利克雷卷积可以同理通过莫比乌斯反演转换为求总长度为 $\sum_{i=1}^{n}\lfloor \frac{n}{i^2} \rfloor = O(n)$ 的 $n$ 次狄利克雷卷积（实际上可以更少，因为 $i$ 需要不包含平方因子）。

至此，狄利克雷卷积也可以做到 $O(n(\log\log n)^2)$ 的复杂度了。

在一般的高维下闭集合上，当然也可以使用这种方法在“子集卷积”与“对维相加卷积”之间转换，不过就不一定有如此好的相互归约性质了。

# 5 “积性”函数卷积类问题

我们仍然考虑一般的情况——高维下闭集合上的卷积问题。

**定义 5.1** (“积性”函数)**.** 令 $f : S \to F$，其中 $S$ 为 $k$ 维下闭集合，$F$ 为一种域。
对每一维 $i$，令 $a_i : S_i \to F$，其中 $S_i$ 为 $S$ 中第 $i$ 维所有出现的元素构成的集合。
则称 $f$ 为“积性”函数，当且仅当存在满足要求的 $a_1, a_2, ..., a_k$，使得：

$$f(\mathbf{x}) = \prod_{i=1}^{k} a_i(x_i) \qquad \forall \mathbf{x} \in S.$$

并称满足要求的 $a_i$ 为 $f$ 第 $i$ 维的权值函数。

## 5.1 “积性”函数卷“积性”函数

我们令 $U, V, W$ 分别为参与卷积的 $f, g$ 与卷积结果 $h$ 对应的下闭集合。
要求：$\forall \mathbf{u} \in \prod_{i=1}^{k} U_i, \mathbf{v} \in \prod_{i=1}^{k} V_i, \mathbf{w} \in W$，若 $c(\mathbf{u}, \mathbf{v}, \mathbf{w}) \neq 0$，则 $\mathbf{u} \in U, \mathbf{v} \in V$。
若满足以上要求，则可将 $f, g$ 的定义域补全成 $\prod_{i=1}^{k} U_i, \prod_{i=1}^{k} V_i$ 并沿用相同的权值函数。
令 $f, g$ 第 $i$ 维的权值函数分别为 $af_i, ag_i$。
那么，

$$\begin{aligned} h(\mathbf{w}) &= \sum_{\mathbf{u} \in \prod_{i=1}^{k} U_i} \sum_{\mathbf{v} \in \prod_{i=1}^{k} V_i} f(\mathbf{u})\, g(\mathbf{v}) \prod_{i=1}^{k} c_i(u_i, v_i, w_i) \\ &= \sum_{\mathbf{u} \in \prod_{i=1}^{k} U_i} \sum_{\mathbf{v} \in \prod_{i=1}^{k} V_i} \prod_{i=1}^{k} (af_i(u_i)\, ag_i(v_i)\, c_i(u_i, v_i, w_i)) \\ &= \prod_{i=1}^{k} \sum_{u_i \in U_i} \sum_{v_i \in V_i} (af_i(u_i)\, ag_i(v_i)\, c_i(u_i, v_i, w_i)) . \end{aligned}$$

于是卷积结果 $h$ 也是“积性”函数，只需要处理出 $h$ 对应的权值函数 $ah_1, ah_2, ..., ah_k$ 即可按照 $h$ 的一种拓扑序顺序线性求出 $h$ 每个元素的权值（找到这个元素的某一非 0 维即可通过将该维降至 0 的状态的答案和该维的权值函数得知其答案）。

以质因子指数集为例，其上的 $|U| = |V| = |W| = n$ 的 lcm 卷积、正交狄利克雷卷积、狄利克雷卷积均满足使用该方法的要求，于是均可做到 $O(n)$ 复杂度求解两个积性函数的卷积结果。

## 5.2 “积性”函数卷一般函数

我们令 $U, V, W$ 分别为参与卷积的 $f, g$ 与卷积结果 $h$ 对应的下闭集合，其中 $f$ 为“积性”函数。

要求：$\forall \mathbf{u} \in \prod_{i=1}^{k} U_i, \mathbf{v} \in V, \mathbf{w} \in W$，若 $c(\mathbf{u}, \mathbf{v}, \mathbf{w}) \neq 0$，则 $\mathbf{u} \in U$。

类似的，若满足以上要求，则可将 $f$ 的定义域补全成 $\prod_{i=1}^{k} U_i$ 并沿用相同的权值函数。

那么就转换为 $g \to h$ 的一个逐维独立线性变换问题了。可以使用类似高维广义卷积 $f \to f'$ 的过程依次操作每一维。

以质因子指数集为例，其上的 $|U| = |V| = |W| = n$ 的 lcm 卷积、正交狄利克雷卷积、狄利克雷卷积均满足使用该方法的要求。而由于这些卷积满足对 $y$ 产生贡献的所有 $x$，均有 $x \le y$（偏序），所以操作每一维都不需要新增“重要”的位置，于是均可在与求前缀和相同的 $O(n \log\log n)$ 复杂度内求解积性函数与普通函数的卷积结果。

综合来看，“积性”函数卷“积性”函数将卷积问题转换为了每维卷积，再乘起来的问题；“积性”函数卷一般函数将卷积问题转换为了每维独立线性变换问题，而且线性变换的方式是已知的。

## 6 总结

本文介绍了 OI 中卷积类问题所使用的几种结构，并逐步推出了 OR 卷积、AND 卷积、XOR 卷积，$p$ 进制 max 卷积、min 卷积、不进位加法卷积，FFT/NTT 的 DFT/IDFT 以及子集卷积。之后进一步扩展到高维下闭集合上，推出了质因子指数集上的前后缀和、前后缀差分、lcm 卷积、gcd 卷积以及正交狄利克雷卷积、狄利克雷卷积。

本文使用“对称卷积”的结构，将卷积的连乘、逆、ln、exp 等进行归约。本文还讨论了手动张量分解的三种方法——边界秩、张量归约、群表示法，并提到其中一些方法在优化大整数乘法与矩阵乘法上的应用。本文最后探讨了高维下闭集合上“积性”函数相关卷积问题的处理方法。

事实上限于篇幅，有许多本文并未深入提及的内容，比如具体如何通过分治和蝴蝶变换求解 FFT/NTT 的 DFT/IDFT，张量分解的启发式算法，狄利克雷卷积对应的牛顿迭代、求逆、ln、exp 等。

感兴趣的读者可以阅读参考文献中的对应资料或自行查找相关资料进行学习。

## 7 致谢

感谢中国计算机学会提供学习、交流、竞赛的平台。

感谢人大附中叶金毅老师、梁霄老师、佟松龄老师的关心和指导。

感谢学校、家人、朋友对我的支持与鼓励。

感谢叶李蹊同学、陈翰文同学与我的讨论。

感谢叶李蹊同学对本论文的审阅。

# 参考文献

[1] V. Strassen, Gaussian Elimination is Not Optimal, Numerische Mathematik, 1969.

[2] D. Bini, M. Capovani, F. Romani, and G. Lotti, $O(n^{2.7799})$ Complexity for Matrix Multiplication, Information Processing Letters, 1979.

[3] A. Schönhage, Partial and Total Matrix Multiplication, SIAM Journal on Computing, 1981.

[4] D. Coppersmith and S. Winograd, On the Asymptotic Complexity of Matrix Multiplication, SIAM Journal on Computing, 1982.

[5] Sum of squares of degrees of irreducible representations equals order of group, https://groupprops.subwiki.org/wiki/Sum_of_squares_of_degrees_of_irreducible_representations_equals_order_of_group.

[6] djq 学生物解题报告, https://qoj.ac/download.php?type=solution&id=4907.

[7] 魏忠浩, 浅谈张量及其在 OI 的运用, IOI2025 集训队论文集.

[8] T. G. Kolda and B. W. Bader, Tensor Decompositions and Applications, SIAM Review, 2009.

[9] 「DGF」与「块筛」浅谈, https://zhuanlan.zhihu.com/p/1970840679002407534.

# 浅谈随机化算法在信息学竞赛中的应用

南京市第一中学 邱梓轩

## 摘 要

本文通过分析随机化算法相对于确定性算法的优势；运用集中不等式对随机化算法进行概率分析的通用方法；以及随机化借助 Schwartz-Zippel Lemma 在理论模型与信息学竞赛实践之间发挥的作用，对随机化算法在构造、交互、线性代数与图论等信息学竞赛常见题型中的应用进行了深入研究。

## 1 概述

在信息学竞赛中，确定性算法是应用极为广泛的算法。此类算法具备严格的正确性证明，能够针对所有数据范围内的输入给出准确的结果。

随机化算法则与之形成鲜明对比。其通常被视作一种风险很高且不易精准分析的算法，故没有在竞赛中得到广泛的应用。与此同时，在利用随机化算法解题时，选手往往不倾向于对其理论效率进行分析，而仅仅凭借直觉进行推导。这使得在很多选手认知中，随机化算法是一种缺乏可靠性的算法，对其有一种天然的畏惧感。

随着人们对随机化算法研究的不断深入，近年逐渐涌现出越来越多能够利用随机化巧妙解决的问题。尤其在国际赛场上，包括 IOI, APIO 等赛事，均出现了具有相当难度的相关题目。由于国内竞赛选手普遍对随机化算法不够重视，相对缺乏深入研究，国内选手在这类问题上并不具备绝对优势。

基于上述现象，本文从随机化算法与确定性算法的异同入手，结合笔者自身的解题经验以及具体例题，阐述随机化算法在信息学竞赛中不同种类的应用。笔者希望本文能够使读者对随机化算法不再感到陌生，并鼓励读者在实践中尝试应用随机化算法。

## 2 定义与约定

在本文中，称一个随机算法以高概率（with high probability）时间复杂度为 $O(T(n))$，当且仅当对于任意 $\epsilon > 0$，均存在常数 $c$ 使得其运行时间超过 $cT(n)$ 的概率不超过 $\epsilon$。

如无特殊说明，本文中使用的模数 $p$ 为足够大的奇素数，有限域 $\mathbb{F}_p$ 表示模素数 $p$ 的剩余类域。

# 3 随机化对确定性算法的优化

在很多信息学竞赛题目中，设计出一个正确的确定性算法并不困难，但因其时间复杂度过高无法通过，或由于算法过于繁琐，难以在限定的时间内完成代码编写。这种情况有时是因为编写的算法必须在任意合法的输入数据集上均可高效运行，从而在分析问题时不得不考虑最坏情况。

相较于确定性算法，随机化算法最大的优势在于能够将最坏情况转化为一般情况。例如，可以对输入数据进行一些随机处理，使得人为构造的最坏情况被打乱，进而得到平均情况下的复杂度。下面首先通过一个广为人知的例子简要解释这一特性。

## 3.1 快速排序算法

快速排序是一种基础的排序算法，下面给出快速排序的伪代码。

**Algorithm 1** QuickSort

```
1:  Procedure QuickSort(A, l, r)
2:  if l < r then
3:      p ← Partition(A, l, r)
4:      QuickSort(A, l, p − 1)
5:      QuickSort(A, p + 1, r)
6:  end if
7:  End Procedure
8:  Procedure Partition(A, l, r)
9:  p ← RandInt(l, r)
10: swap A[p] and A[r]
11: i ← l − 1
12: for j = l to r − 1 do
13:     if A[j] ≤ A[r] then
14:         i ← i + 1
15:         swap A[i] and A[j]
16:     end if
17: end for
18: swap A[i + 1] and A[r]
19: return i + 1
20: End Procedure
```

可以看到，在上述伪代码的第九行中，我们使用 RandInt 函数从 $[l, r]$ 中等概率选取了一个整数，为快速排序引入了随机性。

如果快速排序的实现不使用随机的方式确定 $p$，而是每次取一个固定元素，则无论这个元素是什么，总是存在数据使得这个算法的时间复杂度退化为 $O(n^2)$。换句话说，尽管确定性快速排序算法在平均情况下表现优秀，但它难以应对最坏情况。

然而引入随机化后，无论输入数据是什么，从直觉上看每次选取的下标 $p$ 对应的元素 $A_p$ 期望均为 $A$ 的中位数，因此每次递归时序列长度期望减半。严格的分析 [2] 可以证明该算法以高概率时间复杂度为 $O(n \log n)$。

深入分析两种快速排序算法之间的差异，可以发现随机化算法提供了一个自适应的机会。确定性算法由于其流程固定的特性，可针对该固定的流程设计相应数据，使其陷入最坏情况。而随机化算法由于其随机性，在设计数据时无法预知其具体表现，故无法达到最坏情况。在随机化的应用中，这一特性是分析问题以及选择合适的随机化算法的核心。

## 3.2 简单应用

**例题 3.1** (Three Servers). [1]

有 $n$ 个任务需要处理，第 $i$ 个任务需要 $t_i$ 的时间。现在有三个服务器，要将这些任务分配给三个服务器，使得三个服务器中最大的总用时减去最小的总用时尽可能小。$n \le 400, t_i \le 30$。

解法

容易想到一个动态规划算法：记 $f_{i,j,k}$ 表示考虑前 $i$ 个任务，设当前三个服务器的总用时分别为 $t_1 \le t_2 \le t_3$，则 $j = t_2 - t_1, k = t_3 - t_2$ 是否可行。如果 $j,k$ 可以取到的上界为 $B$，则这个做法的时间复杂度为 $O(nB^2)$。下面对 $B$ 的大小进行估计。

考虑 $j,k$ 中的一个，例如 $j$。相当于给每个 $t_i$ 确定符号 $c_i \in \{-1, 0, 1\}$，则 $j$ 的取值范围为 $c_i t_i$ 所有前缀和绝对值的最大值。由于简单的贪心做法可以使最终 $j$ 不超过 $\max_i t_i \le 30$，所以可将需要计算的上界变为下面这个问题的结果：

有 $n$ 个绝对值不超过 $t = 30$ 的数 $a_i$，且它们的总和绝对值也不超过 $t$。求 $a_i$ 所有前缀和绝对值最大可能是多少。

在最坏情况下，前一半的 $a_i = -t$，后一半的 $a_i = t$。此时可以达到 $\frac{nt}{2} = O(nt)$ 的界。所以在原问题中 $B = O(nt)$，总复杂度即为 $O(n^3t^2)$，不能通过。

然而这个上界是非常浪费的。由于最终所有数之和的绝对值不大，若正负号分布相对平均，在任意时刻的前缀和绝对值也不应太大。在这种情况下，可以考虑随机化做法。注意到任务的顺序并不重要，所以先随机打乱整个序列，这样需要估算的上界就变为这个新问题的结果：

有 $n$ 个绝对值不超过 $t = 30$ 的数 $a_i$，且它们的总和绝对值也不超过 $t$。在随机打乱 $a_i$ 的顺序后，求 $a_i$ 所有前缀和绝对值最大可能是多少。

[1] 来源：Petrozavodsk Winter 2016. Day 8: GP of Saratov, https://qoj.ac/problem/8430

后文会证明一个基本的结论：长度为 $n$ 的随机 $\pm 1$ 序列总和的期望为 $O(\sqrt{n})$。直接应用这个结论至上述问题中，可以得到 $O(\sqrt{nt\log nt})$ 的上界。事实上，更紧的界是 $O(\sqrt{nt})$，但其推导过程较为复杂，具体可见 [8]。由上述结论，在随机打乱后，$a_i$ 所有前缀和绝对值最大值以高概率不超过 $O(\sqrt{nt})$。这样总时间复杂度为 $O(n^2t)$，可以通过。

■

这道题是一个非常经典的模型，具有很强的拓展性。在解题过程中，我们分析了确定性做法的瓶颈，并针对性地设计了随机化做法，使得“最坏情况”带来的瓶颈不会出现，而是变成了更理想的“平均情况”，从而起到了优化复杂度的作用。

**例题 3.2** (Giraffes). [2]

给定一个 $1, 2, \cdots, n$ 的排列 $p_i$，要求重排 $p$ 得到一个新的排列 $q$ 使得其是好的，最小化 $p_i \neq q_i$ 的下标数量。一个排列 $p$ 是好的，当且仅当对于任意 $1 \leq l \leq r \leq n$，下面两个条件不同时满足：

1. 存在 $l < k < r$ 满足 $p_l > p_k < p_r$
2. 存在 $l < k < r$ 满足 $p_l < p_k > p_r$

$n \leq 8000$，保证给定的排列 $p$ 随机生成。

解法

将原题合法的条件做一些转化，可以得到下面这个问题：给定平面上 $n$ 个点 $(i, p_i)$，初始有一个正方形，两个对角顶点分别位于 $(1,1), (n,n)$。每次需将四个顶点中的一个向内移动一个单位，要求最大化给定的 $n$ 个点作为正方形顶点的次数之和。

这个问题有一个直接的动态规划：记 $f_{x,y,i}$ 表示左下角顶点为 $(x,y)$，边长为 $i$ 的正方形向内操作到结束所得到的最大答案。朴素实现这个动态规划复杂度为 $O(n^3)$，无法通过。

此时需注意到题目给定的保证排列随机的性质，这一随机性对上述动态规划并不能起到直接的优化作用，但可以尝试将上述做法转化为能够利用该性质的形式。

原问题本质上可以看作选出若干相互包含的正方形，使得每个正方形均以某个给定的点作为顶点，因此答案一定不超过这个序列中四个方向上的最长上升子序列的长度之和。可以证明一个随机排序的最长上升子序列长度是 $O(\sqrt{n})$ 的，故可以说明答案的量级为 $O(\sqrt{n})$。

从这个角度入手改写动态规划，可以想到将答案和状态互换，即记 $f_{x,y,i}$ 表示以 $(x,y)$ 为左下角的正方形，为了使得答案 $\geq i$，所需的边长至少是多少。有用的 $(x,y)$ 仅有 $O(n)$ 个，且转移可以使用数据结构优化，故这样就做到了 $O(n\sqrt{n}\log n)$ 的复杂度，可以通过。

■

在这个问题中，初始得到的确定性做法并不能立即运用随机性。此时需要结合随机排列的性质，思考原问题中哪些部分可以运用该性质，从而对原确定性做法做出合适的转化，最终得到解法。

[2] 来源：JOI Open 2022, https://qoj.ac/problem/4356

**例题 3.3** (Best Sun). [3]

给定平面上 $n$ 个点，任意三点不共线。要求选择一个点集 $T$，使得 $T$ 凸包上的点集恰为 $T$，且不存在点严格在凸包内部。确定点集 $T$ 后，需要将每个不在 $T$ 中的点与一个 $T$ 中的点之间连一条线段，且在 $T$ 的凸包上相邻两点之间连一条线段，要求线段互不相交且总长度尽可能小。定义 $T$ 的权值为 $T$ 凸包的面积与连接的所有线段长度之和的比值，求所有合法的 $T$ 的最大权值。$n \le 300$。

解法

二分答案 $x$，令点集 $T$ 的凸包面积为 $S_T$，连接的线段长度之和为 $P_T$，则需要检验是否存在 $T$ 满足 $S_T - xP_T \ge 0$，即求 $S_T - xP_T$ 的最大值。

求解这个最大值可以枚举 $T$ 中最左下的点 $p_i = (x_i, y_i)$，并设计一个复杂度为 $O(n^2)$ 的动态规划，按斜率顺序枚举凸包上的边求解。由于这个动态规划与主题关系不大，在此略去具体细节。

这样做的复杂度为 $O(n^3 \log \epsilon^{-1})$，常数较大，难以通过。复杂度的瓶颈主要在于需要枚举凸包左下角的点，但在最优解中只有一个这样的点，造成了非常多的浪费。由于无法预知哪个点是最优的，枚举的复杂度并不能省略，故考虑优化二分的过程。

对于一个起点，如果某次二分 $x$ 时以它为起点的最优权值 $< 0$，那么之后在对任何更大的 $x$ 二分时它都不会贡献到答案，即不用以它为起点进行动态规划。如果先枚举起点然后二分答案，可以在开始二分前对这个起点用已求出的最优解做一次检查，如果它已经不合法，那么这个起点不会更新答案，可直接跳过。

然而很遗憾，如果起点的顺序按照答案递增排列，那么这个优化没有作用，因为每次新枚举一个起点都会更新答案，复杂度仍然为 $O(n^3 \log \epsilon^{-1})$。

可以将上述过程中“更新答案的次数”看作一个序列的严格前缀最大值数目，在最坏情况下，这个数目是 $O(n)$ 的。但可以证明，对于一个随机序列，这个数目以高概率不超过 $O(\log n)$。所以在二分前将所有起点随机打乱然后依次枚举，仅会做 $O(\log n)$ 次二分，复杂度被优化至 $O(n^3 + n^2 \log n \log \epsilon^{-1})$，可以通过。

■

通过上述三个不同类型的题目，可以归纳利用随机化优化确定性算法最坏情况的通用方法。首先，给出一个确定性算法，并找到其复杂度瓶颈。接着，分析瓶颈产生的原因并对应给出若干最坏情况的例子。若最坏情况不易出现，则可以尝试分析平均情况下算法的表现，并结合题目限制运用随机化，通过随机顺序、随机排列、随机权值等方式将最坏情况转化为一般情况。

有时确定性算法无法直接运用随机化，此时需先对其进行一些转化，使其具有可随机的关键形式。有一些常见的平均情况与最坏情况差异很大的随机化方式，通过向这些方向转化可以有效地找到合适的随机方式。除了例题中提到的三个模型以外，还有一些其它技

[3] 来源：Petrozavodsk Summer 2022. Day 7. HSE Koresha Contest, https://qoj.ac/problem/4886

巧，例如随机父亲的有根树期望深度为 $O(\log n)$，随机排列的不动点数目期望为 1 等，限于篇幅在此不一一列举。

## 3.3 构造类问题

构造类问题通常只要求给出任意合法的解。在很多问题中，解的数目极为庞大，此时可以考虑通过以特定方式随机一些解并进行检验。这样做的好处在于无需进行过多额外的思考，可以规避复杂的思考过程。然而其局限性主要体现在较为困难的题目中，直接随机得到解的概率太低。在此情形下，可结合确定性算法的部分思考过程，以更巧妙的方式进行随机。

**例题 3.4** (Divisible by 4 Spanning Tree)**.** [4]

给定一张 $n$ 个点 $m$ 条边的连通无向图，问是否存在一棵生成树，使得度数为奇数的点的数量是 4 的倍数。$n, m \le 2 \times 10^5$。

解法

由于所有点的度数之和为偶数，必然有偶数个度数为奇数的点，所以任意生成树中度数为奇数的点数只有两种可能性：是 4 的倍数，或模 4 余 2。

一个简单的想法是取一个生成树，如果它已经合法，则解决了问题。否则可以尝试对其进行一些调整使其合法。存在一些较为复杂的确定性做法来做这一调整，不过这些做法需要细致的观察和对图论的理解，代码实现也较为复杂。

由于解一共只有两类，并且看起来相当对称，因此有很多生成树是合法的。然而，如果直接朴素地每次随机选择一棵生成树进行判定，正确率非常低。例如一个很大的环外挂着 2 个点，必须将某条与这两个点相邻的边删除才能得到解，所以得到解的概率为 $O(\frac{1}{n})$。

考虑借鉴确定性做法的思路，即在得到一个生成树后对其进行调整，使其合法。不过这里不一定要调整出所有合法情形，而是可以选择一些较简单的操作进行调整，因为可以通过多次随机弥补无法覆盖所有情况的缺陷。

考虑删掉一条边然后加入一条边这种简单的操作。设删掉的边为 $(x, y)$，加入的边为 $(u, v)$，令 $deg_u$ 表示 $u$ 的度数的奇偶性。则如果 $x = u$，在 $deg_y = deg_v$ 时得到一组解。否则如果四个点互不相同，在 $deg_x \oplus deg_y \oplus deg_u \oplus deg_v = 1$ 时得到一组解。这一条件可通过树上差分后枚举被删除的边进行判定。

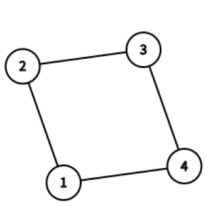
注意到上述条件非常宽松，实际上加入上述条件后重复随机 15 次已经可以通过本题。下面利用确定性做法的结论，说明如果有解，利用上述方法每次随机得到解的概率至少为 $\frac{1}{k}$，其中 $k$ 为小常数。

考虑下图中的三种情况。对于一个环，如果其中存在两个相邻的点 $deg$ 相同，无论初始选择的生成树是什么，将其与环上的另一条边同时翻转状态后均可变得合法，这种情况已经被上述操作覆盖。

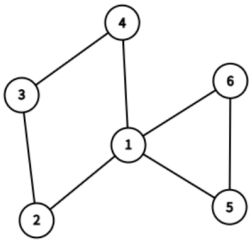
[4] 来源：SEERC 2022, https://qoj.ac/problem/14108

对于多个环有一个相同点的情况，一定可以操作其中一个环使得公共点的度数奇偶性被翻转，因此只需两次操作即可确保合法。这种情况下，只要其中一个环上点的奇偶性不是奇偶交替的，就能找到解，故找到解的概率至少为 $\frac{1}{2}$。

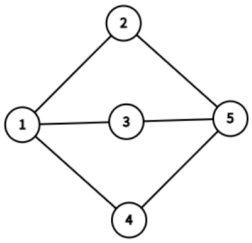
若这两种情况均未出现，可以说明对于任意点数 $> 6$ 的边双连通分量，都不存在任何合法的生成树。而对于点数 $\geq 6$ 的边双连通分量，可以通过暴力验证随机算法的正确率是足够高的。故完成了证明。



环



两个环



小的边双连通分量

■

## 3.4 交互与通信类问题

在交互与通信题中，由于交互库的不确定性，有时难以通过本地测试，对一种方法的性能进行分析，或因为潜在的最坏情况，导致一些优化并无效果。由于在实际应用中，并不需要完整给出一个做法的复杂度或正确性证明，而是只需在实践中表现良好，随机化成为了在这一类问题中非常常见的思路。

通常来说，与其它类型问题相同，还是需先设计出具有正确性的确定性做法。在此基础上，通过对不同对象施加随机化，让交互库的不确定行为变得可以通过随机的方式表示。这样一来，随机化就提供了与之前优化最坏情况时类似的作用：使人为设计的交互库运行方式或输入数据失效，而转变为可控的随机情况。

**例题 3.5** (Interactive Sort)**.** [5]

给定 $n$，令序列 $A, B$ 分别为 $[1, n]$ 中所有偶数和奇数按任意顺序组成的序列。可以进行 $m$ 次询问，每次给出 $i, j$，交互库会回答 $A_i, B_j$ 的大小关系。需要确定每个数。

$n \leq 10^4, m = 3 \times 10^5$，交互库不自适应。

[5] 来源：NERC 2017, https://qoj.ac/problem/11793

解法

对于一个元素，如果在另一个序列中对每个数都询问一次，可以确定其排名。同时也可将另一个序列中的元素分成两个部分，这两个部分之间的大小关系已知，只需确定它们内部的关系。

但对于下一个元素，并不能立即知道其属于这两个部分中的哪一个，需先通过确定性的二分确定其属于哪一个段，然后在段内重复上述过程，将其分为两段。

这样做的问题在于，如果 $A$ 序列按顺序排列，每一次会将一段分为大小极不均匀的两段，使得复杂度退化为 $O(n^2)$。此时可利用交互库不自适应的特点，按照随机顺序枚举 $A$ 中的元素进行分治。这样所需的操作次数可以使用与前文快速排序相同的方式进行分析，是 $O(n \log n)$ 的。从而这种随机询问的方式在 $O(n \log n)$ 次询问内解决了原问题。

■

**例题 3.6** (Broken Device)**.** [6]

A 需要传送一个 $[0, 10^{18}]$ 的数给 B，他可以向 B 发送一个长度为 $n = 150$ 的 01 序列，但这个序列中有 $k$ 个位置是坏的，B 收到的序列中这些位置的值始终为 0，其它位置则为 A 传送的值。A 知道哪些位置是坏的，但是 B 不知道。$k \leq 40$。

解法

将所有好的位分成若干连续段，每一段的开头放一个 1，则 B 看到第一个 1 就知道从这个位置开始，下面会有一段传送信息的区间。然而使用这种方式不能指示这一段到什么地方结束。

考虑提前约定好每一段的长度，这样 B 每看到一个 1，就知道从它开始向后固定长度的位是用来传递信息的。但由于长度必须提前约定好，为了保证能够成功传输，不能将连续段的大小设得太大，导致需增加很多额外的指示位。

此时发现无论如何改进，总是存在针对该做法的对应数据将需要的位卡满。尝试运用随机化，让两人将所有位置按照同样的方式随机打乱，且将需要传输的数异或一个相同的随机数来将坏位置和待传输的数都变得完全均匀随机，从而规避潜在的最坏情况。换句话说，这样随机化削弱了问题的限制：现在给定的信息不再是由输入数据决定的，而是随机的。

将输入全部变成随机后，我们就可以自己实现一个交互库模拟这个过程。接下来的所有优化方式均可通过本地大量模拟分析其正确率与性能。例如刚刚的做法中，取每一段长度为 2，大概率可以保证做到 $k = 25$，在 $k = 30$ 左右时也有相当的成功率。

接下来发现即使某一个位是坏的，它也可以传输一位 0 的信息。所以在尝试向后匹配时，不一定要找完整的空位。对于后两个信息位，如果某一位对应的信息是 0，那么允许这一位是坏的。由于传输的数也是随机的，加上这个优化后，在 $k = 35$ 时也可以几乎确保正确。

[6] 来源：JOISC 2017, https://qoj.ac/problem/364

不过上述方法在 $k = 40$ 时错误率仍有 1/100 左右，在多组测试下难以通过。进一步思考，可以多次随机并取最优的一次。问题在于如何还原出使用了何种随机方式。

可以在第 $i$ 次随机时，要求填完所有信息后，后面仍有 $i-1$ 个空位，将这 $i-1$ 个空位全部填上 1。这样 B 读完所有信息位之后，只需通过后面 1 的个数，即可知道采用的是哪种随机方式。经过本地实验，这些优化已经可以以足够高的概率通过本题。

■

## 3.5 总结

相较于确定性算法，随机化算法凭借其灵活的逻辑和不可预测的运行方式，能够巧妙地规避最坏情况。形象地说，确定性算法可被看作是一个能够应对一切问题的工具，而随机化算法则可看作是一个将普遍问题转化为特定问题的工具。通常情况下，具有随机性的问题比普遍问题更容易解决，所以这一运用方式能大幅降低思考的难度。

结合不同题型的特点，上文已总结出了不同类型题目应用随机化优化确定性做法的一般方式与过程。在实际应用中，随机化的主要难点在于找出合适的随机化对象，以及确定合适的随机化方式。通常可从确定性算法中获得启发，将随机化工具与确定性算法合理结合，以实现效率的突破。

# 4 随机化算法的概率分析

## 4.1 随机变量的期望和方差

**定义 4.1** (概率空间)**.** 定义概率空间 $\mathbf{P} = (U, p)$。其中 $U$ 为一个有限非空集合，表示考察对象的全体，$p : U \to [0,1]$ 为概率函数，满足 $\sum_{u \in U} p_u = 1$。

**定义 4.2** (事件与概率)**.** 定义事件 $T$ 为 $U$ 中的一个子集。定义事件 $T$ 发生的概率 $\Pr(T) = \sum_{t \in T} p(t)$。

**定理 4.1** (Union Bound)**.** 有若干事件 $T_1, T_2, \cdots, T_m$，则对于任意 $T \subseteq \cup_i T_i$ 均有 $\Pr(T) \le \sum_i \Pr(T_i)$。

Union Bound 是概率论中最基本的定理之一。其内容很简单，但在后续分析和计算的过程中会起到很重要的作用。

**定义 4.3** (随机变量与期望)**.** 在固定的概率空间 $\mathbf{P} = (U, p)$ 中，定义随机变量 $X : U \to \mathbb{R}$ 为一个从事件到实数的映射。定义随机变量 $X$ 的期望 $E(X) = \sum_{u \in U} p(u)X(u)$。

随机变量的期望含义为以概率函数 $p$ 为权值，随机变量取值的“平均值”。在随机算法的概率分析中，期望是一个很重要的指标，因为它衡量了在“平均情况”下算法的运行效率或正确率。

**定理 4.2** (期望的线性性)**.** 设随机变量 $X = \sum_{i=1}^{m} c_i X_i$ 为若干随机变量的线性组合，则 $E(X) = \sum_i c_i E(X_i)$。

证明. $E(X) = \sum_{u \in U} p(u)X(u) = \sum_{u \in U} p(u)(\sum_i c_i X_i(u)) = \sum_i c_i (\sum_{u \in U} p(u)X_i(u)) = \sum_i c_i E(X_i)$

□

注意上述定理对于任意 $X, X_i$ 均成立，并不要求事件 $X_i$ 相互独立。下面我们利用期望的线性性计算第二节中的一个问题。

**例题 4.1.** 对于一个随机的 $[1, n]$ 的排列 $p_i$，计算其前缀最大值数目的期望。

解法

前缀最大值的数目可看作一个随机变量 $X$，其可表示为随机变量 $X_1, X_2, \cdots, X_n$ 的和，其中 $X_i$ 在事件“$i$ 为一个前缀最大值”发生时取值 1，否则取值 0。根据期望的线性性，$E(X) = \sum_i E(X_i)$，下面对于一个 $i$ 计算 $E(X_i)$。

由于 $X_i$ 的取值只有 0/1 两种，其期望等于事件“$i$ 为一个前缀最大值”发生的概率。固定后 $n - i$ 个元素，则该事件发生，当且仅当在前 $i$ 个元素中第 $i$ 个元素为最大的一个，即发生的概率为 $\frac{1}{i}$。故 $E(X_i) = \Pr(X_i = 1) = \frac{1}{i}$。

对所有 $i$ 求和，得到 $E(X) = \sum_i E(X_i) = \sum_i \frac{1}{i} = O(\log n)$。

■

## 4.2 集中不等式与 Chernoff Bounds

除了随机变量的期望，设计算法时我们还需关心随机变量的“集中程度”，即它是否会偏离期望很多，这是因为在实践中我们需要关心最坏情况下的表现。

因此除了运行时间或正确率的期望以外，还需一个指标刻画其是否集中。这个指标通常通过方差或标准差表示。

**定义 4.4** (方差与标准差)**.** 定义随机变量 $X$ 的方差 $Var(X) = E((X - E(X))^2)$，标准差 $\sigma(X) = \sqrt{Var(X)}$。

利用期望的线性性展开方差的表达式也可得到 $Var(X) = E(X^2) - E(X)^2$。方差和标准差越小，说明 $X$ 的分布越集中。但在实际应用中，我们更希望知道的不是方差，而是一个随机变量超过某个上界的概率，这一概率主要通过下列不等式进行估计。

**定理 4.3** (Markov Inequality)**.** 若随机变量 $X$ 始终非负，则 $\forall c > 0, \Pr(X > cE(X)) \leq \frac{1}{c}$。

证明. 若 $E(X) = 0$，由于 $X$ 非负，其只能恒等于 0，定理显然成立。

否则 $E(X) = \sum_s \Pr(X = s)s \geq \Pr(X > cE(X))cE(X)$。不等式两边同时除以 $E(X)$，得证。这里最后一步是因为 $X$ 非负，可以将 $[0, cE(X))$ 中的取值放缩至 0，$[cE(X), \infty)$ 中的取值放缩至 $cE(X)$。

□

Markov 不等式非常简单直观，可以做一些基本的估计，并且对后续不等式的证明有很重要的作用。但由于它的界过于宽松了，一般无法对实际问题的分析起到作用。

**定理 4.4** (Chebyshev Inequality)**.** 对于随机变量 $X$，$\forall c > 0, \Pr(|X - E(X)| > c\sigma(X)) \leq \frac{1}{c^2}$。

证明. 对随机变量 $Y = (X - E(X))^2$ 应用 Markov 不等式可以得到 $\Pr(Y > c^2 E(Y)) \leq \frac{1}{c^2}$。而根据定义 $E(Y) = Var(X)$，即 $\Pr((X - E(X))^2 < c^2 Var(X)) \leq \frac{1}{c^2}$。概率内的不等式两边同时开根号，得到 $\Pr(|X - E(X)| > c\sigma(X)) \leq \frac{1}{c^2}$。

□

在实际应用中，$\sigma(X)$ 通常远小于 $E(X)$，此时 Chebyshev 不等式远远强于 Markov 不等式。

**例题 4.2** (硬币问题)**.** 有 $n = 10^4$ 枚均匀的硬币，记随机变量 $X$ 表示朝上的硬币数目。求 $X - E(X) > 500$ 的概率上界。

经过计算可得 $E(X) = \frac{1}{2}n = 5000, \sigma(X) = \frac{1}{2}\sqrt{n} = 50$。

应用 Markov 不等式，$\Pr(X - E(X) > 500) = \Pr(X > 1.1E(X)) \leq \frac{10}{11}$。

应用 Chebyshev 不等式，$\Pr(X - E(X) > 500) \leq \Pr(|X - E(X)| \leq 10\sigma(X)) \leq 0.01$。

可以看出两个界有着本质的差异，不过 0.01 这个界仍然不紧。实际上，利用下面的不等式，在这个问题中可以得到约 $3 \times 10^{-11}$ 的界。

**定理 4.5** (Chernoff Bounds)**.** 令 $X_i$ 为在 $\{0, 1\}$ 中独立取值的随机变量（概率不限），随机变量 $X = \sum_i X_i$，$X$ 的期望 $E(X) = \mu$，则

1. $\forall \delta > 0, \Pr(X \geq (1 + \delta)\mu) \leq \left(\frac{e^{\delta}}{(1 + \delta)^{1+\delta}}\right)^{\mu}$
2. $\forall 0 < \delta < 1, \Pr(X \leq (1 - \delta)\mu) \leq \left(\frac{e^{-\delta}}{(1 - \delta)^{1-\delta}}\right)^{\mu}$

证明. 应用 Markov 不等式，对于任意 $t > 0$，我们有：

$$\begin{aligned}\Pr(X \geq (1 + \delta)\mu) &= \Pr(e^{tX} \geq e^{t(1+\delta)\mu}) \\ &\leq \frac{E(e^{tX})}{e^{t(1+\delta)\mu}} \\ &\leq \frac{e^{(e^t - 1)\mu}}{e^{t(1+\delta)\mu}}\end{aligned}$$

当 $\delta > 0$ 时，取 $t = \ln(1 + \delta) > 0$ 代入，得到第一个不等式。

应用 Markov 不等式，对于任意 $t < 0$，我们有

$$\begin{aligned}\Pr(X \leq (1 - \delta)\mu) &= \Pr(e^{tX} \geq e^{t(1-\delta)\mu}) \\ &\leq \frac{E(e^{tX})}{e^{t(1-\delta)\mu}} \\ &\leq \frac{e^{(e^t - 1)\mu}}{e^{t(1-\delta)\mu}}\end{aligned}$$

当 $0 < \delta < 1$ 时，取 $t = \ln(1-\delta) < 0$ 代入，得到第二个不等式。

□

这两个不等式是 Chernoff Bounds 最强的形式，分别对应两个不等号的方向。但在实际应用中，常用的是下面几个稍弱一些但形式简单的推论：

**推论 4.1** (Chernoff Bounds 常见形式)**.** 令 $X_i$ 为取值仅有 0, 1 的随机变量，随机变量 $X = \sum_i X_i$，$X$ 的期望 $E(X) = \mu$，则

1. $\forall \delta > 0, \Pr(X \geq (1+\delta)\mu) \leq e^{-\frac{1}{3}\delta^2\mu}$
2. $\forall c \geq 6\mu, \Pr(X \geq c) \leq 2^{-c}$
3. $\forall 0 < \delta < 1, \Pr(X \leq (1-\delta)\mu) \leq e^{-\frac{1}{2}\delta^2\mu}$

这几个推论的证明与原定理相似，不再赘述。

## 4.3 Chernoff Bounds 的应用

在第二节中，我们不加证明地进行了若干随机算法的效率估计，现在利用 Chernoff Bounds 给出严格的计算过程。

**例题 4.3** (随机游走问题)**.** 有 $n$ 个随机变量 $X_i$，每个变量等概率取值 $\pm 1$，证明 $X = \sum_i X_i$ 的绝对值为 $O(\sqrt{n})$。

解法

原命题等价于对于任意 $\epsilon > 0$，均存在常数 $c$ 使得 $\Pr(|X| > c\sqrt{n}) < \epsilon$。

计算可得 $E(X) = 0$。由于对称性，只需说明 $\Pr(X > E(X) + c\sqrt{n}) < \epsilon$。在这个问题中，$X_i$ 的取值不是 $\{0,1\}$，而是 $\pm 1$，故先令 $Y_i = \frac{X_i+1}{2}$，$Y = \sum_i Y_i$。此时 $Y_i$ 在 $\{0,1\}$ 中等概率取值，需证明 $\Pr(Y > E(Y) + c\sqrt{n}) < \epsilon$。

应用 Chernoff Bounds 的第一个推论，$\mu = E(Y) = \frac{n}{2}$，取 $\delta = \frac{c}{\sqrt{n}}$ 代入可得 $\Pr(Y \geq (1 + \frac{c}{\sqrt{n}})\frac{n}{2}) \leq e^{-\frac{1}{3}(\frac{c}{\sqrt{n}})^2\frac{n}{2}}$，化简得到 $\Pr(Y - E(Y) > c\sqrt{n}) \leq e^{-\frac{c^2}{6}}$。故对于任意小的 $\epsilon$，均可取 $c = \sqrt{-6\ln\epsilon}$ 使得原命题成立。

■

**例题 4.4** (前缀最大值问题)**.** 证明 $n$ 个元素的随机排列前缀最大值数目为 $O(\log n)$。

解法

令 $X_i$ 指示位置 $i$ 是否为前缀最大值，则 $X_i \in \{0,1\}$ 且计算可得 $\Pr(X_i = 1) = \frac{1}{i}$。需证明对于 $X = \sum_i X_i$ 和任意小常数 $\epsilon > 0$，均存在常数 $c$ 使得 $\Pr(X > c\log n) < \epsilon$。

计算可得 $E(X) = \sum_i \frac{1}{i} = O(\log n)$。由于常数对结论无影响，方便起见下面假定 $E(X) = \log n$。

应用 Chernoff Bounds 的第一个推论，$\mu = \log n$，取 $\delta = \frac{c}{\sqrt{\log n}}$ 代入计算可得 $\Pr(X > \log n + c\sqrt{\log n}) \le e^{-\frac{c^2}{3}}$。在 $n$ 充分大时，$\sqrt{\log n} < \sqrt{n}$，故 $\Pr(X > (c+1)\log n) \le \Pr(X > \log n + c\sqrt{\log n}) \le e^{-\frac{c^2}{3}}$，原命题得证。

事实上，在本问题中，由于 $E(X)$ 与允许的误差量级相同，可以直接使用第二个推论。对于任意 $c > 6\log n$，均有 $\Pr(X > c) \le 2^{-c}$，原命题得证。

■

以上两个例题对第二节中两个问题所用的结论进行了严格证明。从中可以看出使用 Chernoff Bounds 分析正确率和效率的一般方法。

通常来说，需先将待考察的随机变量写成若干取值 $\{0,1\}$ 的随机变量之和的形式，并通过平移放缩的方式做一些转化得到标准形式。然后选用几个不等式中合适的一个，通过计算 $\mu = E(X)$ 的取值，选择合适的 $\delta$ 代入使得不等式右侧为形如 $e^{-c}$ 的常数，得到较为精准的误差分析。

Chernoff Bounds 在需求误差远小于期望的量级时尤为适用，例如上述第一个例题，可以将 $O(n)$ 的期望放缩至 $O(\sqrt{n})$ 的误差量级。

下面通过一个例题介绍 Chernoff Bounds 在分析算法时更综合的运用。

**例题 4.5** (物品传送问题)**.** [7]

一张 $m$ 条边的有向图上有 $n$ 个物品需要被传送，每个物品的传送路线已经被确定，记为 $s_i \to t_i$。每条边一秒只能传送一个物品，物品可以在原地等待，求将所有物品传送至目的地的最短时间。

解法

记 $d$ 为最长路径的长度，$c$ 为一条边被经过次数的最大值，则答案不小于 $\max(c,d) = O(c+d)$。这个下界是可以被达到的，不过其构造十分复杂且与主题无关，在此略去。这里将给出一种十分简单的构造，使得答案为 $O(c + d\log nm)$。

考虑确定一个阈值 $B$，每 $B$ 个时刻分为一组，并强制令每个物品在一组时刻中仅移动一次。这样只需每组开始时任意边的起点上都不存在超过 $B$ 个物品，即可保证合法。再确定一个阈值 $R$，让每个物品等概率随机一个 $[0,R)$ 中的组并从该组开始移动，此时答案为 $B(R+d)$。关键在于如何选取合适的 $B,R$，使得 $B(R+d)$ 不大，且发生冲突的概率足够小。

记 $N_{e,t}$ 为一个随机变量，表示第 $t$ 组时刻开始时，位于边 $e$ 起点的点数目，发生冲突的概率即为 $\Pr(\cup_{e,t}(N_{e,t} > B))$。根据 Union Bound，其不超过 $\sum_{e,t}\Pr(N_{e,t} > B)$。

令随机变量 $X_{i,e,t}$ 表示物品 $i$ 在 $t$ 组时刻开始时是否位于边 $e$ 的起点。则 $N_{e,t} = \sum_i X_{i,e,t}$ 为一个 $\{0,1\}$ 随机变量求和的形式。

[7] 来源：Algorithm Design, Chapter 13[3]

对于固定的 $i,e,t$，只存在至多一个出发时间使得 $X_{i,e,t}=1$，故 $E(X_{i,e,t})\le\frac{1}{R}$。回顾之前定义了 $c$ 表示一条边被经过次数的最大值，则根据期望的线性性 $E(N_{e,t})=\sum_{s_i\to e\to t_i}E(X_{i,e,t})\le\frac{c}{R}$。

应用 Chernoff Bounds 的第一个推论，取 $\delta=2$ 代入，得到当 $R=\frac{c}{q\log nm}$（$q$ 为常数）时，$\Pr(N_{e,t}>\frac{3c}{R})<\frac{1}{(nm)^z}$，其中 $z$ 可以通过选择合适的 $q$ 做到任意大。

取 $B=\frac{3c}{R}$，代入原来的 Union Bound 中，得到失败的概率不超过 $nm\Pr(X>B)<\frac{1}{(nm)^{z-1}}$，其中 $z$ 为任意大常数。此时 $B(R+d)=O(c+d\log nm)$，即这种算法可以以任意大的概率使得运行时间不超过 $O(c+d\log nm)$。

■

## 4.4 总结

本节通过随机变量的期望以及集中不等式，从两个不同角度提供了分析随机算法效率的方法。这些分析方法可以广泛应用于信息学竞赛中，为随机算法的合理性提供了理论依据，使得随机化不再仅仅依赖于感性的设计，而可以通过理论推导进行严谨的证明。

# 5 随机化在理论模型中的应用

在图论、线性代数等领域，有一些与信息学竞赛结合紧密的理论模型。在这些方面，随机化是一个重要的工具，起到了将理论模型转变为具有实践意义的算法的作用。

## 5.1 Schwartz-Zippel Lemma

**定理 5.1** (Schwartz-Zippel Lemma)**.** 令 $P\in[x_1,x_2,\cdots,x_n]$ 为定义在域 $\mathbb{F}$ 上的一个 $d$ 次非零多项式，$S$ 为域 $\mathbb{F}$ 的有限非空子集，则当 $x_1,\cdots,x_n$ 在 $S$ 中独立均匀随机取值时，$\Pr(P(x_1,\cdots,x_n)=0)\le\frac{d}{|S|}$。

证明. 对变量数目 $n$ 进行归纳。当 $n=1$ 时，根据代数基本定理，任意 $d$ 次多项式 $P(x)$ 根的数目不超过 $d$，结论显然成立。

下面假设该结论对于 $n-1$ 个变量成立，可以将 $P(x_1,\cdots,x_n)$ 看作关于 $x_n$ 的多项式 $\sum_{i=0}^{d}x_n^iP_i(x_1,\cdots,x_{n-1})$。由于 $P$ 非零，必然存在一个 $i$ 使得 $P_i$ 非零，不妨取最大的一个 $i$，则 $\deg P_i\le d-i$。

应用归纳假设，随机选取 $x_1,x_2,\cdots,x_{n-1}$ 时，$\Pr(P_i(x_1,\cdots,x_{n-1})=0)\le\frac{d-i}{|S|}$。由于取的是最大的 $i$，当 $P_i(x_1,\cdots,x_{n-1})\neq0$ 时，$P(x_n)$ 为一个 $i$ 次多项式。根据代数基本定理，此时 $\Pr(P(x_n)=0)\le\frac{i}{|S|}$。

记 $P(x_1,\cdots,x_n)=0$ 为事件 $A$，$P_i(x_1,\cdots,x_{n-1})=0$ 为事件 $B$，事件 $B$ 的补集为 $B^c$。可以将之前分析的结果写作 $\Pr(B)\le\frac{d-i}{|S|},\Pr(A\mid B^c)\le\frac{i}{|S|}$。则

$$\begin{aligned}\Pr(A) &= \Pr(A \cup B) + \Pr(A \cup B^c) \\ &= \Pr(B)\Pr(A \mid B) + \Pr(B^c)\Pr(A \mid B^c) \\ &\le \Pr(B) + \Pr(A \mid B^c) \le \frac{d-i}{|S|} + \frac{i}{|S|} = \frac{d}{|S|}\end{aligned}$$

□

在信息学竞赛中，常在有限域 $\mathbb{F}_p$ 中应用 Schwartz-Zippel Lemma，在模质数 $p$ 意义下向多项式中代入随机取值以达到将高次多项式压缩成整数的效果。以下是一道直接应用 Schwartz-Zippel Lemma 的例题。

**例题 5.1** (奇迹)**.** [8]

给定两个长度为 $3^n$ 的序列 $A_i, B_i$，每个元素均在 $[0, p)$ 中等概率随机生成，其中 $p = 998244353$ 为大质数。对于某个未知的运算表 $op : \{0,1,2\}^2 \to \{0,1,2\}$，定义 $(i_0 i_1 \cdots i_{n-1})_3 \oplus (j_0 j_1 \cdots j_{n-1})_3 = (op(i_0, j_0) \cdots op(i_{n-1}, j_{n-1})_3$。现再给定序列 $C_i$ 满足 $C_k = (\sum_{i \oplus j = k} A_i B_j) \bmod p$，构造一个可行的运算表。$n \le 10$。

解法

枚举运算表 $op$，需要高效检验 $op$ 是否合法。由于数据随机的性质，考虑抽取其中的部分进行检验。下面分析对于单个 $C_i$，存在两个或更多 $op$ 同时合法的概率。

将 $A_i, B_i$ 看作总共 $2 \times 3^n$ 个变量，则对于一个固定的 $op$，$C_i$ 可写作在 $\mathbb{F}_p$ 上关于这 $2 \times 3^n$ 个变量的二次多项式。若某两个不同的 $op$ 均合法，说明这两个 $op$ 对应的多项式之差等于 0。

如果这两个多项式不全相同，应用 Schwartz-Zippel Lemma 可以说明冲突的概率不超过 $\frac{2}{p}$，由 Union Bound 可知存在两个 $op$ 冲突的概率不超过 $\frac{2 \times 3^9}{p}$，且这个界很松。

但如果存在某一个数不在 $i$ 的任意一位中出现，那么仅在这一个数上有差异的两个 $op$ 得到的多项式是完全相同的。所以需选择三个不同的 $i$，使得每一个数均在每一位上出现过，才可保证正确率。

直接这样做复杂度仍然太高，但可以发现应用 Schwartz-Zippel Lemma 的条件仅要求待考察的 $C_i$ 能够被写成关于 $A_i, B_i$ 的低次多项式，原始的计算形式是否被保留并不重要。所以将三个序列均仅保留最后一位，对于之前的 $n-1$ 位全部求和进行压缩，再应用 Schwartz-Zippel Lemma. 由于这样压缩只是做了线性组合，在一次乘法后展开仍然是二次多项式的形式，故仍能正常应用 Schwartz-Zippel Lemma. 如此转化可以得到一个 $n = 1$ 的子问题，这个子问题与原问题的正确率是完全一致的，所以解决这个子问题即可。

■

[8] 来源：IOI 2026 国家集训队集中培训 Day 2, https://qoj.ac/problem/15506

## 5.2 Schwartz-Zippel Lemma 在哈希算法中的应用

在信息学竞赛中，哈希算法本质上是一种压缩算法，其以牺牲正确率为代价将庞大的序列、集合、树等对象压缩成一个整数或其它利于储存与运算的形式。为了均衡正确率与效率，其主要难点在于选取合适的哈希函数。在分析哈希算法的正确率时，Schwartz-Zippel Lemma 是非常重要的工具。

### 5.2.1 序列哈希

对于一个有限域 $\mathbb{F}_p$ 下的整数序列 $a_0, a_1, \cdots, a_{n-1}$，定义其哈希值 $H(a) = (\sum_{i=0}^{n-1} a_i B^i) \bmod p$，其中 $B$ 为一个预先确定的整数。

**定理 5.2** (序列哈希正确率)**.** 当 $B$ 在 $[0, p)$ 中均匀随机选取时，对于任意两个不同序列 $a, b$，$\Pr(H(a) = H(b)) \le \frac{n-1}{p}$。

证明. 令序列 $c$ 满足 $c_i = a_i - b_i$，则存在 $c_i \neq 0$。设多项式 $P(x) = \sum_{i=0}^{n-1} c_i x^i$ 为一个 $n-1$ 次多项式。则 $H(a) = H(b)$ 当且仅当 $P(B) = 0$。根据 Schwartz-Zippel Lemma，当 $x$ 在 $[0, p)$ 中随机取值时，$\Pr(P(x) = 0) \le \frac{n-1}{p}$，即 $\Pr(H(a) = H(b)) \le \frac{n-1}{p}$。

□

### 5.2.2 集合哈希

对于一个不可重集合 $S = \{s_1, s_2, \cdots, s_n\}$，在域 $\mathbb{F}_{2^w}$ 下定义其哈希值 $H(S) = \oplus_{i=1}^{n} x_{s_i}$，其中 $x_i$ 为 $[0, 2^w)$ 中的整数，$\oplus$ 为二进制异或。

**定理 5.3** (集合哈希正确率)**.** 当 $x_i$ 在 $[0, 2^w)$ 中均匀随机选取时，对于任意两个不同集合 $A, B$，$\Pr(H(A) = H(B)) \le 2^{-w}$。

证明. 由于 $\oplus$ 运算对每一位独立，只需说明当 $w = 1$ 时结论成立即可。在 $w = 1$ 时，$\oplus$ 与 $+$ 是相同的运算。所以若 $A \neq B$，$H(A) - H(B)$ 可以被写成关于 $x_i$ 的一次多项式，从而冲突概率 $\le \frac{1}{2}$。

□

集合哈希的正确率与集合大小无关，仅与域的大小相关。同时，若将集合哈希的定义修改为求和而非异或和，也可分析得到相同的正确率。

### 5.2.3 树哈希

在图论中，常常需对无标号有根树进行哈希，将其压缩为一个整数。这个问题有多种方式，但在如 Automorphism[9] 等题目中，需要支持单点修改时快速更新树哈希的结果。

考虑这样一种树哈希的方式，对每个点 $u$，定义一个多项式 $P(u) = \prod_{v \in son(u)}(P(v) + x_{sz_u})$，其中 $sz_u$ 为 $u$ 的子树大小，$son(u)$ 为 $u$ 的儿子集合。特别地，对于叶子节点 $u$，$P(u) = 1$。在 $[0, p)$ 中均匀随机取 $x_i$ 代入可以求得具体的哈希值。

**定理 5.4** (树哈希正确率)**.** 当 $x_i$ 在 $[0, p)$ 中独立均匀随机选取时，对于任意两棵大小不超过 $n$ 的不同有根树 $T(u), T(v)$，$\Pr(P(u) = P(v)) \le \frac{n}{p}$。

证明. 首先证明不同有根树 $T(u), T(v)$ 对应的多项式 $P(u) \neq P(v)$。如果 $sz_u \neq sz_v$（不妨设 $sz_u < sz_v$），则 $P(v)$ 中含有 $x_{sz_v}$ 这一项，当 $P(u)$ 不包含，显然 $P(u) \neq P(v)$。否则对两棵树的点数归纳。当 $n = 1$ 时只有一种有根树，显然成立。

通过下标最大的被包含在 $P(u)$ 或 $P(v)$ 中的变量 $x_n$ 可以确定树的大小。根据整系数多项式唯一分解定理，将 $P(u), P(v)$ 看作关于 $x_n$ 的多项式，即可分别唯一分解为 $\prod(x_n + Q_i(u))$ 与 $\prod(x_n + Q_i(v))$，且对应 $Q_i(u)$ 与 $Q_i(v)$ 不全相同。对存在不同的 $Q_i(u)$ 与 $Q_i(v)$ 应用归纳假设即可。

现在对于两棵不同的有根树 $T(u), T(v)$，它们对应的 $P(u), P(v)$ 不相同。根据 $P(u)$ 的计算过程，可以看出其次数不超过 $u$ 的子树大小，即 $n$。故 $P(u) - P(v)$ 为一个次数不超过 $n$ 的多项式，应用 Schwartz-Zippel Lemma 即可。

□

除此之外，还有一些更简洁巧妙的树哈希方式，也可以用 Schwartz-Zippel Lemma 对其正确率进行分析。由于其涉及更多多项式技巧，在此不做展开，可见 [9]。

### 5.2.4 哈希的一般方法

除了上述列出的几种常见哈希算法及正确率分析以外，有时还需对陌生的对象设计哈希函数。此时主要的方式是尝试设计一个函数 $H : S \to \mathbb{F}_p[x_0, x_1, \cdots]$，其中 $S$ 为考察的对象，$p$ 为质数，或形如 $2^w$ 等较为特殊的域。

这里 $H$ 需满足 $\forall s, t \in S, s \neq t, H(s) \neq H(t)$。然后代入域中的随机整数作为 $x_i$ 求值，正确率用 $\frac{\max_{s \in S} \deg H(s)}{p}$ 进行估计。

## 5.3 随机化在图论问题中的应用

许多图论问题的结论需要依托线性代数与幂级数工具进行表达，但在实践中进行大规模的幂级数运算效率十分低下。此时随机化是一个常见的工具，通过 Schwartz-Zippel Lemma

[9] 来源：Petrozavodsk Summer 2018. Day 8: Yuhao Du Contest 5, https://qoj.ac/problem/2203

将多项式压缩为整数运算，大大提高了算法效率。

**定义 5.1** (Tutte 矩阵)**.** 对于无向图 $G = (V, E), V = \{1, 2, \cdots, n\}$，定义其 Tutte 矩阵 $A$。其中若 $(i, j) \notin E, A_{i,j} = 0$，否则若 $i < j, A_{i,j} = x_{i,j}$，若 $i > j, A_{i,j} = -x_{j,i}$。

**定义 5.2** (带权 Tutte 矩阵)**.** 对于有边权的无向图 $G = (V, E)$，设其 Tutte 矩阵为 $A$。定义其带边权的 Tutte 矩阵 $A'$ 满足 $A'_{i,j} = A_{i,j} y^{w_{i,j}}$，其中 $w_{i,j}$ 为边权。

上述定义中 $x_{i,j}, y$ 均为形式元。关于 Tutte 矩阵，有下列几个定理：

**定理 5.5** (一般图完美匹配定理)**.** 无向图 $G$ 存在完美匹配，当且仅当其 *Tutte* 矩阵行列式非零。

**定理 5.6** (一般图最大匹配定理)**.** 无向图 $G$ 的最大匹配大小等于其 *Tutte* 矩阵的秩。

**定理 5.7** (一般图最大权完美匹配定理)**.** 对于有边权的无向图 $G$，若其带权 *Tutte* 矩阵的行列式等于 0，则 $G$ 不存在完美匹配。否则设行列式中 $y$ 的最高次数非零项为 $y^s$，则 $G$ 的最大权完美匹配权值为 $\frac{s}{2}$。

上述定理的证明较为复杂，可以在 [1][7] 找到详细证明。

上述定理很难直接应用在信息学竞赛中，因为行列式的变量数目与无向图的边数 $|E|$ 同阶，按照朴素方法维护多项式计算行列式的时间开销无法接受。此时考虑通过 Schwartz-Zippel Lemma 将多项式转化为有限域 $\mathbb{F}_p$ 下的整数。

**例题 5.2** (Degree Harmony)**.** [10]

给定一张 $n$ 个点，$m$ 条边的无向图 $G$ 和序列 $A_1, A_2, \cdots, A_n$。定义 $G$ 的一个生成子图 $G'$ 是好的，当且仅当 $\forall i, d_i \le A_i, d_i \equiv A_2 \pmod 2$，其中 $d_i$ 为 $G'$ 中点 $i$ 的度数。求出 $G$ 中边数最少的好子图 $G'$。$n \le 150, \sum_i A_i \le 150$。

解法

令 $X = \sum_i A_i$，只需考虑 $2 \mid X$ 的情形。构造一张 $X$ 个点的无向图 $H$，其中对于每个 $1 \le i \le n$，有 $A_i$ 个颜色为 $i$ 的点。在 $H$ 中每一对颜色相同的点之间连一条边权为 0 的边。对于每一条 $G$ 上的边 $(u, v)$，在 $H$ 中所有颜色为 $u$ 和颜色为 $v$ 的点两两之间连一条边权为 1 的边。可以证明，若 $H$ 不存在完美匹配，则原问题不存在合法子图 $G'$。否则答案等于 $H$ 的最大权完美匹配大小。考虑利用带权 Tutte 矩阵解决最大权完美匹配问题。

将带权 Tutte 矩阵的行列式看作关于 $y$ 的多项式。由于我们只关心 $y$ 每一项前的系数是否非零，可以取大质数 $p$，将 $[0, p)$ 的随机整数代入变量 $x_{i,j}$ 中。Tutte 矩阵行列式的每一项由至多 $n$ 个 $x_{i,j}$ 乘起来得到，故根据 Schwartz-Zippel Lemma，这样代入冲突的概率不超过 $\frac{n}{p}$。

至此，计算带权 Tutte 矩阵行列式被转化为单变量的行列式问题。代入 $O(n)$ 个不同变量的取值计算整数的行列式，最后通过拉格朗日插值法还原多项式，解决了原问题。

■

[10] 来源：ABC412G, https://atcoder.jp/contests/abc412/tasks/abc412_g

**例题 5.3** (Cut Cut Cut!)**.** [11]

给定一张 $n$ 个点 $m$ 条边的有向无环图，保证 1 号点的出度 $\leq 20$。对每个 $2 \leq i \leq n$ 求出最少删除多少条边, 才能使得不存在任何一条从 1 到 $i$ 的路径。$n \leq 10^5, m \leq 3 \times 10^5$。

解法

题目的本质是对每个点 $t$ 求出以 1 为源点，$t$ 为汇点的最大流。根据最大流最小割定理，需要求出尽可能多的 $1 \to t$ 的不交路径。求解不交路径相关问题的有力工具是 LGV 引理：

**引理 5.1** (LGV 引理)**.** 在有向无环图 $G = (V, E)$ 中，边有边权，令 $w(P)$ 表示路径 $P$ 上每条边的权值之积。给定大小为 $n$ 的起点集合 $A$ 和终点集合 $B$。定义一组不交路径 $S$ 由 $n$ 条路径 $S_i$ 组成，满足任意两个 $S_i$ 之间没有公共点，且存在一个排列 $\sigma$ 使得 $S_i$ 为从 $A_i$ 到 $B_{\sigma(i)}$ 的路径。定义 $S$ 的权值为 $(-1)^{t(\sigma)} \prod_i w(S_i)$，其中 $t(\sigma)$ 表示 $\sigma$ 的逆序对数目。定义矩阵 $M$，其中 $M_{i,j}$ 表示所有 $A_i \to B_j$ 的路径 $P$ 的权值之和。则 $\det(M) = \sum_S w(S)$。

证明可见 [5]，在此略过。

在本题中，需要考察的是边不交路径。故对每条边 $(u, v)$ 建一个点，在两条边 $(u, v), (v, w)$ 对应的点之间连边。对于一个汇点 $t$，相当于在这张新的有向图上，以 1 的所有出边为起点集合，$t$ 的所有入边为汇点集合，所能选出的最大点不交路径集合大小。

将起点集合看作 $A$，终点集合看作 $B$，则使用 LGV 引理中定义的矩阵 $M$ 刻画不交路径，$M_{i,j}$ 的值容易在遍历图的过程中求出。先考虑当 $|A| = |B|$ 时，如何判定答案是否等于 $|A|$。这并不直接等价于 $\det(M) \neq 0$，因为 $\det(M)$ 求出的是有符号的权值之和，可能会出现两个方向相反的路径相互抵消，导致结果为 0 但实际存在不交路径集合。

考虑对有向图上的每条边 $(u, v)$ 设一个形式元 $x_{u,v}$。在这个前提下，由于任意两条不同路径上的形式元不全相同，它们的权值一定不同且在多项式意义下无法抵消。故如果将形式元看作边权而非 1，求出的 $\det(M)$ 是否非零与是否存在不交路径是完全等价的。这个行列式中每个单项式的次数不超过总边数，所以根据 Schwartz-Zippel Lemma，随机代入 $[0, p)$ 的整数作为 $x_{u,v}$ 求 $\det(M)$ 的冲突概率不超过 $\frac{m}{p}$。

最后回到原问题，需要求出一个矩阵的最大子矩阵，使得行列式非零。这相当于求出矩阵的秩，容易在 $O(k^3)$ 的时间内解决，其中 $k$ 为矩阵大小。由于点 1 的出度不超过 20，$k \leq 20$。总时间复杂度 $O(nk^3 + mk^2)$。

■

在这个例题中，我们利用 Schwartz-Zippel Lemma 在有符号的行列式和无符号的积和式之间做出了转化，没有引入新的变量，也没有改变算法复杂度。类似的方法还可运用在行列式求二分图最大匹配等问题中。

[11] 来源：QOJ，https://qoj.ac/problem/61

## 6 总结与展望

本文重点从算法设计、概率分析、理论模型三个角度详细说明了随机化算法在信息学竞赛中的应用。除此之外，随机化在包括随机采样、NP 问题的启发式优化和近似解等方面仍有很多应用空间，限于篇幅在此不过多展开。

相对于已经发展十分成熟的诸多确定性算法，随机化算法仍有很多未被发掘的性质与用处。随机化和概率方法在理论计算机科学领域是一个正在被深入研究的重要方向，近年在线性规划等问题中也出现了重大突破性成果。不过人们现在对这一方兴未艾领域的探索也只是冰山一角，希望本文能够激起读者对随机化的兴趣，在将来为这一领域的发展做出自己的贡献。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。

感觉父母对我的培养和教育。

感谢南京市第一中学、韩英老师、李翔老师的关心和指导。

感谢戴江齐学长、仇嘉明同学、裴牧达同学为本文审稿并提供了宝贵意见。

## 参考文献

[1] AtCoder. ABC412G Editorial. https://atcoder.jp/contests/abc412/editorial/13395.

[2] Georgy Gimel'farb. Algorithm quicksort: Analysis of complexity.

[3] Jon Kleinberg and Éva Tardos. Algorithm Design, chapter 13. Addison-Wesley, 2005.

[4] Michael Mitzenmacher and Eli Upfal. Probability and Computing. University of Cambridge, 2005.

[5] OI Wiki. LGV 引理. https://oiwiki.org/graph/lgv/.

[6] Wikipedia. Schwartz-zippel lemma. https://en.wikipedia.org/wiki/Schwartz-Zippel_lemma.

[7] Wikipedia. Tutte matrix. https://en.wikipedia.org/wiki/Tutte_matrix.

[8] 李白天. 随机变量前缀和的控制. https://www.cnblogs.com/Elegia/p/16922216.html.

[9] 裴牧达. 一种经典树哈希算法的正确性证明. https://www.luogu.com.cn/article/vfctpagu.

# 再谈信息学竞赛中的空间优化问题

中山纪念中学 全柏锋

## 摘 要

本文从动态规划与多次操作问题两个角度出发，系统整理并分析了若干空间复杂度优化方法，并通过典型例题对相关方法的适用范围与复杂度进行了讨论。

## 1 引言

空间复杂度是算法设计中至关重要的一个部分。在 IOI 2022 国家集训队论文集的《浅谈信息学竞赛中的空间优化问题》中，陈知轩曾对信息学竞赛中常见的空间复杂度优化技巧进行了系统总结。

本文从动态规划问题与带多次操作的问题两个角度出发，对《浅谈信息学竞赛中的空间优化问题》中的部分空间优化方法做了进一步拓展，并在此基础上探讨了若干新的空间复杂度优化思路及其适用范围。

## 2 动态规划问题的空间优化

许多动态规划问题可以抽象为一个有 $n$ 个点 $m$ 条边的有向无环图 $G$。图中每个点对应一个类，记第 $i$ 个点的类为 $D_i$，其大小为 $|D_i|$，称类中存储的内容为信息。设点 $i$ 的前驱集合为 $Pred(i)$，则 $D_i$ 通常由 $Pred(i)$ 中各点的类合并，再进行一些仅与点 $i$ 有关的处理得到。

一种直接的做法是为每个类预先分配空间，并按照拓扑序依次计算各点的 $D_i$。在这种实现下，所需的空间复杂度为 $O(\sum_{i=1}^{n}|D_i|)$。

大多数动态规划问题的答案只和其中一个点的类有关，因此当某个点 $u$ 的所有后继对应的类均已计算完成后，我们就可以释放 $D_u$ 占用的空间。该做法的空间复杂度与其选择的拓扑序密切相关。

在某些情况下，类 $D_i$ 的大小还可能随着计算过程发生变化。

### 2.1 一种选择拓扑序的方法

假设可以将图中所有点划分为 $k$ 层，第 $i$ 层的点集记为 $G_i$，并满足以下条件：

1. $\forall 1 \le i < k$，图中只可能存在从 $G_i$ 指向 $G_{i+1}$ 的有向边，除此之外不存在跨层有向边。

2. $\forall 1 \le i \le k$，存在一种 $G_i$ 中的点的排列顺序 $p_{i,1}, p_{i,2}, ..., p_{i,|G_i|}$，使得 $\forall 1 \le j_1 < j_2 \le |G_i|$，不存在从 $p_{i,j_2}$ 指向 $p_{i,j_1}$ 的有向边。

则可以选取拓扑序为 $p'_1 + p'_2 + \cdots + p'_k$，其中 $p'_i$ 表示序列 $p_{i,1}, p_{i,2}, \ldots, p_{i,|G_i|}$，加号表示序列的拼接。该做法的空间复杂度为 $O(\max_{i=1}^{k}\{\sum_{x\in G_i}|D_x|\})$。

## 2.2 网格图的情形

考虑一个拥有 $n = n_1 \times n_2$ 个顶点的有向图 $G$。若我们将编号为 $(i-1)\times n_2 + j$ 的顶点记为二元组 $(i, j)$，则在满足以下条件时，称 $G$ 为网格图：

1. $\forall 1 \le i \le n_1, 1 \le j < n_2$，存在一条从 $(i, j)$ 指向 $(i, j+1)$ 的有向边。

2. $\forall 1 \le i < n_1, 1 \le j \le n_2$，存在一条从 $(i, j)$ 指向 $(i+1, j)$ 的有向边。

3. 除上述边外，不存在其它的有向边。

不妨设所有点对应的类 $D$ 的大小均为 $M$。若令 $k = n_1$，并取 $G_i = \{(i,1),(i,2),\ldots,(i,n_2)\}$，$p_{i,j} = (i, j)$ 则可得到 $O(n_2M)$ 的空间复杂度；若令 $k = n_2$，并取 $G_i = \{(1,i),(2,i),\ldots,(n_1,i)\}$，$p_{i,j} = (j, i)$ 则可得到 $O(n_1M)$ 的空间复杂度。

根据 $n_1$ 和 $n_2$ 的大小关系分类，可以得到 $O(\min(n_1,n_2)M)$ 的空间复杂度。

## 2.3 树的情形

### 2.3.1 特殊情形：树形背包[1]

#### 2.3.1.1 题目描述

给定一棵 $n$ 个点的无根树，每条边有边权 $a_i$。$\forall i \in [1,n]$，求恰好有 $i$ 个点且包含 1 号点的连通块的边权之和的最大值。

#### 2.3.1.2 数据规模

$1 \le n \le 5000, 0 \le a_i \le 10^{15}$。

#### 2.3.1.3 朴素做法

[1]加强自 https://loj.ac/p/10153

令根节点为 1 号点。对每个节点 $i$ 维护一个数组 $D_i$，其中 $D_{i,j}$ 表示以 $i$ 为当前连通块根、包含 $j$ 个点时的最大边权和。当处理 $u$ 与其子节点 $v$ 时，需要将 $D_v$ 合并至 $D_u$，该过程等价于对 $j$ 维度上做 $(\max,+)$ 卷积。

朴素实现树形背包，时间和空间复杂度均为 $O(n^2)$。

#### 2.3.1.4 空间优化

可以发现本题中每个点 $i$ 对应的类 $D$ 的大小是动态变化的。将 $D_v$ 的信息合并至 $D_u$ 后，$D_v$ 不再被后续计算所使用，而 $D_u$ 的规模相应增加 $|D_v|$。

由于任意一个点在任意时刻至多属于某一个尚未完成合并的类，必有 $\sum_{i=1}^{n}|D_i| \le n$ 成立。

因此，在完成一条边 $(u,v)$ 的合并操作后，可立即释放 $D_v$ 所占用的空间，从而将空间复杂度优化至 $O(n)$。

### 2.3.2 一般情形

#### 2.3.2.1 朴素做法

在前一节中，我们分析了类 $D$ 的大小随合并过程动态变化的情形，并给出了一种 $O(n)$ 空间复杂度的实现方法。

在一般情形中，为便于分析，我们假设所有点对应的类 $D$ 的规模均为 $M$。不妨设进行一次类的合并时间复杂度为 $T$。朴素地实现树形 DP，时间复杂度为 $O(nT)$，空间复杂度为 $O(nM)$。

#### 2.3.2.2 启发式合并做法

对于一个有根树，定义点 $u$ 的重儿子为其所有儿子中子树节点数最大的点，若有多个则任选其一，记作 $\text{heavy}(u)$。

考虑按照如下顺序进行类的合并：对整棵树进行一次深度优先搜索，访问到点 $u$ 时，首先递归处理其重儿子 $\text{heavy}(u)$。在重儿子子树处理完成后，直接令 $D_u$ 继承 $D_{\text{heavy}(u)}$ 的信息。随后依次访问 $u$ 的所有轻儿子，每处理完一个轻儿子 $v$ 的子树后，将 $D_v$ 的信息合并到 $D_u$ 中。

在该算法中，任意时刻同时存在的类的数量，至多为从根到当前访问节点路径上的轻边数量。因此该算法的空间复杂度为 $O(n + M\log n)$，其中 $O(n)$ 用于存储树的结构及相关信息。时间复杂度仍为 $O(nT)$。

### 2.3.3 换根 DP 的空间优化

#### 2.3.3.1 朴素做法

在若干树上问题中，需要分别以每个节点作为根计算对应的答案。

一种朴素的做法如下：首先任选一个点作为根节点，设 $D_u$ 表示 $u$ 的子树内所对应的信息，$U_u$ 表示 $u$ 子树外所对应的信息。我们先进行一次树形 DP，计算出每个点的 $D_u$；随后再进行一次深度优先搜索，并保证在访问到点 $u$ 时，$U_u$ 中的信息已经被正确维护。设 $v$ 为 $u$ 的一个儿子，可以从 $D_u$ 中剔除 $D_v$ 所对应的贡献，从而得到 $u$ 的子树中除 $v$ 子树外部分的信息，记为 $D'_v$。将 $U_u$ 与 $D'_v$ 合并，即可得到 $U_v$。最后，对每个点 $u$，将 $U_u$ 与 $D_u$ 合并即可得到以 $u$ 为根时的答案。

当信息不具有可减性时，可以维护 $u$ 的所有儿子对应的 $D$ 的前缀合并结果与后缀合并结果，从而在不依赖减法的情况下，得到除某个儿子 $v$ 之外其余所有儿子的信息合并结果。

朴素实现上述做法，时间复杂度为 $O(nT)$，空间复杂度为 $O(nM)$。

#### 2.3.3.2 启发式合并做法

我们首先采用 2.3.2.2 节中的启发式合并方法计算出 $D_1$。随后对整棵树进行一次深度优先搜索，并保证在访问到点 $u$ 时，其对应的 $D_u$ 与 $U_u$ 均已被正确维护。

当访问到点 $u$ 时，我们首先枚举 $u$ 的每一个轻儿子 $v$。对每个轻儿子 $v$，利用 2.3.2.2 节中的方法重新计算 $D_v$，并从 $D_u$ 中剔除 $D_v$ 的贡献，再与 $U_u$ 合并即可得到 $U_v$。随后递归处理 $v$ 的子树。在该过程中，$D_v$ 在使用完毕后应该立即释放其所占用的空间，以保证空间复杂度的上界。

在完成对所有轻儿子子树的处理后，为了计算重儿子 $\text{heavy}(u)$ 对应的 $U_{\text{heavy}(u)}$，需要再次枚举 $u$ 的所有轻儿子 $v$，并重新利用 2.3.2.2 节中的方法计算其对应的 $D_v$，从 $D_u$ 中剔除这些贡献后，即可得到 $D_{\text{heavy}(u)}$，从而计算出 $U_{\text{heavy}(u)}$，最后递归处理 $\text{heavy}(u)$ 的子树。

可以发现，对于每一条轻边 $(u, v)$，算法在递归过程中都会使用 2.3.2.2 节中的方法重新计算对应的 $D_v$。由于任意一点到根的路径上轻边数量为 $O(\log n)$，该算法的时间复杂度为 $O(nT \log n)$，空间复杂度为 $O(n + M \log n)$。

需要注意的是，该算法依赖于信息的可减性；而下文所介绍的算法均不再利用这一性质。

#### 2.3.3.3 重链剖分做法

我们首先对树进行重链剖分，并按照各条重链链顶的 DFS 序依次处理每一条重链，同时记录对应链顶节点的 $U$ 信息。

在处理某一条重链时，我们在该重链上按照各轻子树的子树大小之和进行加权分治，以保证递归深度为 $O(\log n)$。每次选取分治中心后，计算其对应的 $D$ 与 $U$，并将重链划分为左右两部分递归处理。在递归过程中，需要维护当前子链顶部节点的 $U$ 信息以及底部节点的 $D$ 信息。

该算法的时间复杂度为 $O(nT \log n)$，空间复杂度为 $O(n + M \log n)$。

#### 2.3.3.4 点分治做法

我们对树进行点分治。找到分治重心 $rt$ 后，将以 $rt$ 为根的各棵子树划分为两个集合 $A, B$。

若存在一棵子树的大小不小于 $\frac{1}{3}n$，则将该子树划入集合 $A$，其余子树划入集合 $B$；否则，所有子树的大小均小于 $\frac{1}{3}n$，此时按任意顺序将子树依次加入集合 $A$，直至 $A$ 中子树大小之和不小于 $\frac{1}{3}n$，其余子树划入集合 $B$。

由构造方式可知，集合 $A$ 与 $B$ 中子树大小之和均落在区间 $\left[\frac{1}{3}n, \frac{2}{3}n\right]$ 内。分别利用 2.3.2.2 节中的启发式合并做法计算集合 $A$ 与 $B$ 对应的信息，随后递归处理两个子问题。

由于点分治的递归深度为 $O(\log n)$，该算法的时间复杂度为 $O(nT \log n)$，空间复杂度为 $O(n + M \log n)$。

#### 2.3.3.5 树分块做法

我们对树进行 Top Cluster 树分块，设簇大小为 $B$，则界点的数量为 $O\left(\frac{n}{B}\right)$。

首先利用 2.3.2.2 节中的方法计算并记录所有界点对应的 $D$ 信息，随后按照自上而下的顺序依次处理每一个簇。

对于任意一个簇，其下界点的 $D$ 已经被记录，因此可以在 $O(B)$ 的时间内，采用朴素树形 DP 的方式计算出簇内每个点对应的 $D$。

进一步地，若已知该簇上界点的 $U$ 信息，则可以在簇内进行一次朴素的换根 DP，计算出每个点的 $U$，从而得到簇中所有点的最终答案。注意到在处理当前簇时，其上方的簇均已被处理完成，因此该簇上界点的 $U$ 始终是已知的。

该算法的时间复杂度为 $O(nT)$，空间复杂度为 $O\left(n + \left(B + \frac{n}{B}\right)M\right)$，取 $B = \Theta(\sqrt{n})$ 即可得到 $O(n + M\sqrt{n})$ 的空间复杂度。

#### 2.3.3.6 迭代树分块做法

我们对树进行 Top Cluster 树分块，令 $K = n^{\frac{1}{k}}$，将树划分为 $O(K)$ 个大小为 $O\left(\frac{n}{K}\right)$ 的簇。

首先，利用 2.3.2.2 节中的方法，自下而上计算并记录所有界点 $u$ 对应的 $D_u$ 信息；随后再按照自上而下的顺序，对每个簇运用同样的方法，计算并记录界点 $u$ 对应的 $U_u$ 信息。

在完成上述处理后，每个簇均可视为一个独立的子问题。对每个子问题递归地应用相同的分块过程（在递归过程中，尽管子问题的规模发生变化，但始终保持参数 $K$ 不变），直至子问题的点数不超过 $K$ 为止。

由于在每一层递归中，各子问题的规模之和为 $O(n)$，故每一层分块均可在 $O(nT)$ 的时间内完成，从而该算法的时间复杂度为 $O(knT)$，空间复杂度为 $O(n + kn^{\frac{1}{k}}M)$。

可以发现，当 $k = \log_2 n$ 时，有 $K = 2$，该算法退化为 2.3.3.4 节中的点分治做法；而当 $k = 2$ 时，有 $K = \sqrt{n}$，该算法与 2.3.3.5 节中的树分块做法一致。

### 2.3.4 输出方案的空间优化

#### 2.3.4.1 适用范围

在若干最优化问题中，除了计算最优解之外，还需要输出对应的决策方案。若直接在动态规划过程中记录所有转移信息，往往会带来不可接受的空间开销。

通过对上述方法进行适当拓展，我们可以在保证空间复杂度可控的前提下，输出具体方案。

#### 2.3.4.2 全局平衡二叉树做法

我们对树建立全局平衡二叉树，并自顶向下依次确定各点的决策。处理到点 $u$ 时，利用 2.3.2.2 节中的方法计算其左右儿子对应的信息，并与点 $u$ 的局部信息合并，从而得到 $u$ 的最优决策。随后递归处理其左右子树即可。

该算法的时间复杂度为 $O(nT\log n)$，空间复杂度为 $O(n+M\log n)$。

#### 2.3.4.3 点分治做法

按照 2.3.3.4 节中的点分治策略，首先在分治重心处计算并确定其在最优方案中的决策。随后将问题递归地划分为两个相互独立的子问题 $A$ 与 $B$，并分别输出其对应的方案。

该算法的时间复杂度为 $O(nT\log n)$，空间复杂度为 $O(n+M\log n)$。

在某些特殊情形下，该算法与 2.3.4.2 节中的算法均可做到 $O(nT)$ 的时间复杂度，具体见 2.3.5 节中的例题。

#### 2.3.4.4 树分块做法

基于 2.3.3.5 节中的树分块策略，可以在时间复杂度 $O(nT)$、空间复杂度 $O(n+M\sqrt{n})$ 的条件下输出方案。

#### 2.3.4.5 迭代树分块做法

采用 2.3.3.6 节的做法，可以在时间复杂度 $O(knT)$、空间复杂度 $O(n+kn^{\frac{1}{k}}M)$ 的条件下输出方案。

### 2.3.5 例题[2]

#### 2.3.5.1 题目描述

给定一棵 $n$ 个点以 1 为根的树和 $n$ 个三元组 $(a_i,b_i,c_i)$。

对于每个 $s\in[1,k]$，你需要求一个长度为 $n$ 的非负整数序列 $h$，满足：

1. $\forall i\in[1,n],h_i\in[0,k]$。
2. $\sum_{i=1}^{n}h_i=s$。
3. $\forall i\in[1,n],(h_i \bmod 2)\geq\sum_{j\in\text{child(i)}}(g_j \bmod 2)$，其中 $\text{child}(i)$ 表示点 $i$ 的儿子集合，$g_j=\sum_{v\in\text{subtree}(j)}h_v$ 表示以 $j$ 为根的子树内 $h$ 的和。

[2]笔者的原创题

并最小化 $\sum_{i=1}^{n} f(a_i, b_i, c_i, h_i)$，其中 $f(a, b, c, x) = ax^2 + bx + c$。

你还需要输出 $s = k$ 时的一组最优方案。

#### 2.3.5.2 数据规模

$1 \le n \le 3 \times 10^4, 1 \le k \le 2 \times 10^3, 0 \le a_i, |b_i|, |c_i| \le 10^6$。

#### 2.3.5.3 朴素做法

注意到：

1. 当 $k$ 是偶数时，约束 $\forall i \in [1, n], (h_i \bmod 2) \ge \sum_{j \in \text{child(i)}} (g_j \bmod 2)$ 成立，当且仅当将所有 $h_i$ 为奇数的点取出，并在树中相邻且均为奇数的两个点之间连边后，所得图存在一组完美匹配。

2. 当 $k$ 为奇数时，上述约束成立，当且仅当 $h_1$ 为奇数，且将所有 $h_i$ 为奇数的点取出，在树中相邻且均为奇数的两个点之间连边，并删去 1 号点后，所得图存在一组完美匹配。

设 $dp_{i,t,j}$ 表示以 $i$ 为根的子树中，当点 $i$ 是否与其父节点匹配的状态为 $t \in \{0, 1\}$，且子树内 $\sum h = j$ 时的最小总代价。朴素实现的时间复杂度为 $O(nk^2)$，空间复杂度为 $O(nk)$。

#### 2.3.5.4 凸性证明

下面通过费用流模型证明 $dp_{i,0}$ 在偶数位置上的凸性。

构造一个包含 $n(k + 2) + 2$ 个点的最小费用流网络，其中点包括源点 $S$、汇点 $T$，点 $P_{i,j} (1 \le i \le n, 1 \le j \le k)$，以及点 $A_i, B_i (1 \le i \le n)$。

定义 $dep_i$ 为节点 $i$ 至根节点路径上的节点数量，按如下方式连边：

1. $\forall i \in [1, n], j \in [1, k]$，若 $dep_i + j$ 为偶数，则从 $S$ 向 $P_{i,j}$ 连一条容量为 1、费用为 $f(a_i, b_i, c_i, j) - f(a_i, b_i, c_i, j - 1)$ 的边；否则从 $P_{i,j}$ 向 $T$ 连一条容量为 1、费用为 $f(a_i, b_i, c_i, j) - f(a_i, b_i, c_i, j - 1)$ 的边。

2. $\forall i \in [1, n], 1 \le j \le \lfloor \frac{k}{2} \rfloor$，若 $dep_i$ 为奇数，则从 $P_{i,2j-1}$ 向 $P_{i,2j}$ 连一条容量为 1，费用为 0 的边，否则从 $P_{i,2j}$ 向 $P_{i,2j-1}$ 连一条容量为 1，费用为 0 的边。

3. $\forall i \in [1, n]$，从 $A_i$ 向 $B_i$ 连一条容量为 1，费用为 0 的边。

4. $\forall i \in [1, n], 1 \le j \le \lfloor \frac{k+1}{2} \rfloor$，若 $dep_i$ 为奇数，则从 $P_{i,2j-1}$ 向 $A_i$ 连一条容量为 1，费用为 0 的边，否则从 $B_i$ 向 $P_{i,2j-1}$ 连一条容量为 1，费用为 0 的边。

5. 对任意树边 $(x, y)$，若 $dep_x$ 为奇数，则从 $B_x$ 向 $A_y$ 连一条容量为 1，费用为 0 的边，否则从 $B_y$ 向 $A_x$ 连一条容量为 1，费用为 0 的边。

第 1 类边的含义是，若从 $S$ 连向 $P_{i,j}$ 的边被流经，或从 $P_{i,j}$ 连向 $T$ 的边被流经，则 $h_i \geq j$。

第 2 ~ 5 类边的含义是限制了若 $h_i$ 为偶数，则必定是 $P_{i,2j-1}, P_{2j}$ 之间相互匹配；若 $h_i$ 为奇数，则最后多出来的 $P_{i,h_i}$ 会和某个 $P_{j,h_j}$ 匹配。

我们还需证明，对任意固定的 $i$，被匹配的点 $P_{i,j}$ 构成一个前缀。考虑反证，由 $f$ 的凸性可知 $f(a_i, b_i, c_i, j) - f(a_i, b_i, c_i, j-1)$ 是关于 $j$ 递增的，故若匹配的不是一个前缀，则必定可以通过交换匹配关系将其调整为前缀形式，且不增加总费用，与最优性矛盾。

因此，流量为 $F$ 的最小费用流的费用，加上 $\sum_{i=1}^{n} f(a_i, b_i, c_i, 0)$，恰好等于 $dp_{1,0,2F}$。由费用流的凸性可得 $dp_{i,0}$ 在偶数位置上的凸性。同理可证 $dp_{i,1}$ 在奇数位置上的凸性。

#### 2.3.5.5 时间优化

利用 $dp$ 数组的凸性，在转移过程中可以通过闵可夫斯基和进行优化，在 $O(k)$ 时间内完成一次合并，从而将时间复杂度降至 $O(nk)$。朴素实现下，空间复杂度为 $O(nk)$。

#### 2.3.5.6 空间优化

在此基础上，直接采用 2.3.2.2 节中的方法，即可在 $O(n + k \log n)$ 的空间复杂度内计算出答案。

使用 2.3.4.2 节或 2.3.4.3 节中的方法来输出最优方案。设两个子问题对应的 $\sum h$ 分别为 $k_1, k_2$，则有 $k_1 + k_2 = k$。相应的时间复杂度满足递推关系 $T(n,k) = O(nk) + \max_{i=0}^{k}\{T(n/2, i) + T(n/2, k-i)\}$。由此可得 $T(n,k) = O(nk)$。因此，该算法的整体时间复杂度仍为 $O(nk)$，空间复杂度为 $O(n + k \log n)$。

# 3 带多次操作的问题的空间优化

## 3.1 离线分块

### 3.1.1 适用范围

许多序列问题需要支持区间修改与区间查询，但若直接维护全局结构，往往会带来较大的空间开销。

当查询的信息具有可拆分性时，可以通过离线分块的方式降低空间复杂度。

### 3.1.2 做法

考虑序列上的区间查询问题。将长度为 $n$ 的序列划分为 $O\left(\frac{n}{B}\right)$ 个长度为 $B$ 的连续块。对任意一次区间查询，其查询区间均可分解为若干个整块区间以及至多两个不完整的块内区间。

因此，每次区间查询可以转化为 $O\left(\frac{n}{B}\right)$ 次针对整块的整体查询，以及 $O(1)$ 次块内区间查询，且每次涉及的区间长度均为 $O(B)$。

将所有询问离线后，可以对每个块独立处理，从而仅需维护单个块规模的数据结构。

### 3.1.3 复杂度分析

设序列长度为 $n$ 时，一次整体查询的时间复杂度为 $T_1(n)$，一次区间查询的时间复杂度为 $T_2(n)$，所需空间复杂度为 $M(n)$。

设操作次数为 $q$，由于每次查询涉及 $O\left(\frac{n}{B}\right)$ 次整体查询以及 $O(1)$ 次区间查询，该算法的总时间复杂度为 $O\left(\frac{nqT_1(n)}{B}+qT_2(n)\right)$。而由于每次仅需维护一个块规模的数据结构，其空间复杂度为 $M(B)$。

当 $q=n$，$T_1(n)=O(1)$，$T_2(n)=O(\log n)$，$M(B)=O(B\log B)$ 时，取 $B=\Theta\left(\frac{n}{\log n}\right)$ 即可得到 $O(n\log n)$ 的时间复杂度和 $O(n)$ 的空间复杂度。

### 3.1.4 例题 rla1rmdq[3]

#### 3.1.4.1 题目描述

给定一棵包含 $n$ 个结点的有根树，树上每条边带有非负边权，以及一个长度为 $n$ 的序列 $a$。

记点 $x$ 的父亲为 $fa(x)$，特别地，规定根节点的父亲为它本身。

定义点 $x$ 的深度 $dep(x)$ 为其到根简单路径上所有边的边权和。

有 $q$ 次操作，每次操作形如：

1. 给定 $l,r$，对所有 $l\le i\le r$，令 $a_i\leftarrow fa(a_i)$。
2. 给定 $l,r$，查询 $\min_{i=l}^{r}\{dep(a_i)\}$。

#### 3.1.4.2 数据规模

$1\le n,q\le 2\times 10^5$，边权范围为 $[0,10^9]$。

#### 3.1.4.3 解题思路

将序列 $a$ 划分为长度为 $B=\Theta(\sqrt{n})$ 的连续块。对每个块维护其内部的信息。

对于同一块内的两个位置 $i,j$，若 $a_i$ 是 $a_j$ 的祖先，由于边权非负，则有 $dep(a_i)\le dep(a_j)$，因此在查询最小深度时，$a_j$ 不可能成为答案。故在同一块内，仅需保留那些其祖先未在本块中出现的点，其余点在任意查询中均不可能成为最小值的候选。需要注意的是，这个性质当修改与改块部分相交时不成立。

基于这一性质，在区间修改操作中：

[3] https://qoj.ac/problem/7460

1. 若修改区间完整覆盖某个块，则仅需对该块内满足其祖先不在本块中出现的元素 $a_i$ 进行暴力修改；若修改后出现相同的 $a$，则将其合并。

2. 若修改区间与某个块部分相交，则直接暴力重构整个块。在重构过程中，需要多次查询某个点的祖先，可使用重链剖分实现。由于每个点到根的轻边数量为 $O(\log n)$，可以分析出 $O(n \log n)$ 的均摊复杂度。

整块修改部分的均摊时间复杂度为 $O(n\sqrt{n})$，散块重构部分的总时间复杂度为 $O(n \log n)$。因此，该算法的总时间复杂度为 $O((n+q)\sqrt{n} + n\log n)$，空间复杂度为 $O(n\sqrt{n})$。

将所有操作离线后按块独立处理，即可将空间复杂度进一步优化至 $O(n+q)$。

## 3.2 底层分块

### 3.2.1 适用范围

在许多问题中，需要在序列上维护复杂的数据结构以支持动态修改与查询，但若直接作用于全体元素，往往会带来较大的空间开销。

此时可以通过底层分块的方式降低空间复杂度。

### 3.2.2 做法

设序列长度为 $n$，操作次数为 $q$。将序列划分为若干个长度为 $B$ 的连续块，并对每个块维护其整体信息，从而将原序列压缩为一个长度为 $\frac{n}{B}$ 的新序列，其中每个元素对应一个块的整体信息。随后在该新序列上维护原本用于全序列的数据结构。

当修改某一位置时，先在其所在的块内进行暴力更新并重新计算该块的信息，再在新序列对应位置上更新数据结构中的值。

当进行查询时，首先通过数据结构获取所有被完整覆盖的整块信息，再对至多两个未被完整覆盖的块进行暴力枚举并合并其元素信息，得到最终答案。

该思想可以通过树分块的方式推广至树上。

### 3.2.3 复杂度分析

设原数据结构在长度为 $n$ 的序列上，单次修改的时间复杂度为 $T_1(n)$，空间复杂度为 $M_1(n)$，块内暴力合并的时间复杂度为 $T_2(B)$。

由于新序列长度为 $\frac{n}{B}$，且每次操作至多涉及一次数据结构操作以及 $O(1)$ 次块内暴力处理，该算法总时间复杂度为 $O\left(qT_1\left(\frac{n}{B}\right) + qT_2(B)\right)$，空间复杂度为 $O\left(n + M_1\left(\frac{n}{B}\right)\right)$，其中 $O(n)$ 用于存储原序列及分块所需的基础信息。

当 $q = n$，$T_1(n) = O(\log n)$，$T_2(B) = O(B)$，$M_1(n) = O(\log n)$ 时，取 $B = \Theta(\log n)$，即可得到 $O(n \log n)$ 的时间复杂度和 $O(n)$ 的空间复杂度。

### 3.2.4 例题 无处存储[4]

#### 3.2.4.1 题目描述

给定一棵包含 $n$ 个点的有根树，根节点为 1。对 $2 \le i \le n$，$i$ 号点的父亲为 $f_i$。每个点具有点权 $a_i$。有 $q$ 次操作，每次操作为以下两种之一：

1. 给定 $x, y, v$，将树上从 $x$ 到 $y$ 的简单路径上所有点的点权增加 $v$；
2. 给定 $x, y$，查询树上从 $x$ 到 $y$ 的简单路径上所有点的点权之和，对 $2^{32}$ 取模。

所有操作要求强制在线。

#### 3.2.4.2 数据规模

$1 \le n \le 7 \times 10^6, 1 \le q \le 5 \times 10^4, 1 \le f_i < i, 0 \le v < 2^{32}$。

#### 3.2.4.3 解题思路

我们对树进行 Top Cluster 树分块，将其划分为 $O\left(\dfrac{n}{\log n}\right)$ 个大小为 $O(\log n)$ 的簇。

在处理一条从 $x$ 到 $y$ 的简单路径时，设 $l = \mathrm{LCA}(x, y)$ 为 $x$ 与 $y$ 的最近公共祖先。

若 $x, y$ 不位于同一簇内，则可以将路径拆分为以下六段：

1. 从 $x$ 到其所在簇中、位于簇路径上的最近节点 $x'$ 的路径。
2. 从 $x'$ 到其所在簇上界点 $x''$ 的路径。
3. 从 $x''$ 到 $l$ 的路径。
4. 从 $y$ 到其所在簇中、位于簇路径上的最近节点 $y'$ 的路径。
5. 从 $y'$ 到其所在簇上界点 $y''$ 的路径。
6. 从 $y''$ 到 $l$ 的路径。

其中，第 3 与第 6 段路径跨越了若干段完整的簇路径，等价于在收缩树上进行路径加与路径查询操作，可用全局平衡二叉树维护。

[4] https://qoj.ac/problem/4059

而第 $1,2,4,5$ 段路径完全位于单个簇内部，由于簇大小为 $O(\log n)$，可以直接对路径上的点进行暴力修改或查询。需要注意的是，对第 2 与第 5 段路径进行修改时，会影响对应簇路径上的点权和，因此需同步在全局平衡二叉树中进行相应更新。

若 $x$ 与 $y$ 位于同一簇内，则整条路径均落在簇内部，直接对路径上的节点暴力处理即可。若该操作影响了簇路径的点权和，同样需要在全局平衡二叉树中进行更新。

初始化所需时间为 $O(n)$，每次操作的时间复杂度为 $O(\log n)$，因此该算法的总时间复杂度为 $O(n+q\log n)$。在空间方面，除去存储父节点数组与点权数组所需的两个长度为 $n$ 的数组外，其余部分为 $O\left(\dfrac{n}{\log n}\right)$。

### 3.2.5 例题 区间第 $k$ 小[5]

#### 3.2.5.1 题目描述

给定长度为 $n$ 的序列 $a_1,a_2,\ldots,a_n$。有 $q$ 次询问，每次询问给定 $l,r,k$，要求在线回答区间 $a_l,a_{l+1},\ldots,a_r$ 中第 $k$ 小的数。

#### 3.2.5.2 数据规模

$1\le n,q\le 10^6, 1\le a_i\le 10^9$。

#### 3.2.5.3 解题思路

该问题一个常见的做法是使用可持久化线段树，其空间复杂度为 $O(n\log n)$。下面介绍一种仅需线性空间的方法。

设 $b_i$ 为排名为 $i$ 的数在原序列中的位置（若存在值相同的数，则按其在原序列中的位置先后确定排名），则问题等价于：查询序列 $b$ 从左到右第 $k$ 个在 $[l,r]$ 中的数的值。

我们对序列 $b$ 建立线段树，线段树每个点 $u$ 存储着一个长度为其管辖的区间的长度的 01 序列 $p_u$，其中 $p_u$ 的第 $i$ 位为 0 当且仅当点 $u$ 管辖的区间内按 $b$ 的值排序后第 $i$ 小的元素位于 $u$ 的左子树内。

查询时，在线段树上自顶向下进行二分。设当前位于点 $u$，则我们需要求出左子树内有多少数落在区间 $[l,r]$ 中。

为此，维护两个变量 $l',r'$，分别表示在点 $u$ 管辖的区间内，小于 $l$ 以及不超过 $r$ 的元素个数。则左子树中位于 $[l,r]$ 内的元素个数，等于 $p_{u,l'+1},p_{u,l'+2},\ldots,p_{u,r'}$ 中 0 的个数。

为了高效统计上述区间内 0 的数量，对每个序列 $p_u$ 进行底层分块，将其划分为若干大小为 $B$ 的块，并对每个块预处理其元素和，同时维护块和的前缀和。对于完全覆盖的整块，可直接通过前缀和查询；对于零散部分，在 Word RAM 模型下，当块大小 $B$ 不超过机器字长 $w$ 时（其中 $w$ 为机器字长），可借助位运算在 $O(1)$ 时间内完成统计。

递归进入左子树或右子树时，需要相应更新 $l'$ 与 $r'$，该过程同样可用上述方法完成。

[5]一道经典题

由于线段树所有点管辖的区间长度总和为$O(n\log n)$，该算法的空间复杂度为$O\left(\frac{n\log n}{B}\right)$，取 $B=\lfloor\log_2 n\rfloor$ 即可得到 $O(n)$ 的空间复杂度。该算法的时间复杂度为 $O(n\log n+q\log n)$。

## 3.3 定期重构

### 3.3.1 适用范围

在许多问题中，需要维护一个支持动态修改与查询的数据结构，但每一次修改都会造成额外的空间开销。当修改次数较多时，可能导致空间复杂度无法接受。

此时可以通过定期重构的方式来优化空间复杂度。

### 3.3.2 做法

设序列长度为 $n$，操作次数为 $q$。我们可以采用定期重构的思想，每进行 $B$ 次操作后，对数据结构进行一次整体重构，并在重构过程中回收此前修改所产生的冗余空间。

在部分题目中，查询结果等于其之前所有修改对该查询的贡献之并。若这些贡献满足可拆分性，则同样可以应用定期重构的策略。具体而言，每 $B$ 次操作进行一次重构，在重构时统一计算此前所有修改的贡献之并；查询时，只需将最近一次重构之前的整体贡献，与重构之后至当前时刻的零散修改贡献合并，即可得到最终答案。

### 3.3.3 复杂度分析

设单次修改的时间复杂度为 $T_1(n)$，空间复杂度为 $M_1(n)$；一次重构的时间复杂度为 $T_2(n)$，空间复杂度为 $M_2(n)$。则该算法的总时间复杂度为 $O\left(qT_1(n)+\frac{qT_2(n)}{B}\right)$，空间复杂度为 $O(BM_1(n)+M_2(n))$。

当 $q=n$，$T_1(n)=O(\log n)$，$M_1(n)=O(\log n)$，$T_2(n)=O(n)$，$M_2(n)=O(n)$ 时，取 $B=\Theta\left(\frac{n}{\log n}\right)$ 即可得到 $O(n\log n)$ 的时间复杂度和 $O(n)$ 的空间复杂度。

### 3.3.4 例题 动态二维数点[6]

#### 3.3.4.1 题目描述

有 $n$ 次操作，第 $i$ 次操作为以下两种之一：

1. 向一个初始为空的可重集 $S$ 中插入一个点 $(x_i,y_i)$。

[6]一道经典题

2. 查询集合 $S$ 中满足 $x \le x_i$ 且 $y \le y_i$ 的点的个数。

要求强制在线。

3.3.4.2 数据规模

$1 \le n \le 2 \times 10^5, 1 \le x_i, y_i \le n$。

3.3.4.3 朴素做法

使用树状数组套平衡树，即可在 $O(n \log^2 n)$ 的时间复杂度内完成所有操作，空间复杂度为 $O(n \log n)$。

3.3.4.4 空间优化

采用定期重构的思想，每 $B$ 次操作进行一次重构。将一次查询的答案拆为两部分：最近一次重构之前所有修改的整体贡献，与重构之后至当前时刻的零散修改的贡献。

对于前一部分，可以直接套用 3.2.5 节中的方法进行重构和查询；对于后一部分，可以沿用朴素的树状数组套平衡树做法进行维护。

该算法的时间复杂度为 $O\left(\frac{n^2 \log n}{B} + n \log^2 n\right)$，空间复杂度为 $O(n + B \log n)$。取 $B = \Theta\left(\frac{n}{\log n}\right)$ 即可得到 $O(n \log^2 n)$ 的时间复杂度和 $O(n)$ 的空间复杂度。

3.3.4.5 带点权的情况[7]

在 3.2.5 节的方法中，需要对序列 $p$ 进行前缀求和。当点带权时，设 $p'_{u,i}$ 表示点 $u$ 管辖区间中第 $i$ 小的点的权值，则问题转化为对序列 $p'_u$ 的前缀和查询。

同样可以对 $p'_u$ 进行底层分块，但此时散块部分无法再借助位运算进行加速。若直接对散块中的元素逐一统计，问题等价于 $O(B \log n)$ 次区间第 $k$ 小查询，其总时间复杂度为 $O(nB \log^2 n)$，空间复杂度为 $O\left(n + \frac{n \log n}{B}\right)$。取 $B = \lfloor \log_2 n \rfloor$ 即可得到 $O(n \log^3 n)$ 的时间复杂度和 $O(n)$ 的空间复杂度。

遗憾的是，笔者目前尚未知晓如何在保持时间复杂度不变的前提下，将该问题的空间复杂度进一步优化至线性。

# 4 总结

本文从动态规划问题与多次操作问题两个角度出发，系统性地总结了若干常见的空间复杂度优化方法，并对其适用条件、实现方式及复杂度进行了分析与讨论。

希望本文中所整理的思路与方法，能够为读者在空间复杂度优化方向上的进一步探索与实践提供一定参考。

[7] https://www.luogu.com.cn/problem/P4148

## 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢家人、朋友、教练对我的支持与鼓励。
感谢中山纪念中学的同学们为本文审稿。
感谢吴同春同学与我讨论交流。

## 参考文献

陈知轩，浅谈信息学竞赛中的空间优化问题，IOI 2022 国家集训队论文集。

# 关于 Dirichlet 卷积问题的一些进展

南京外国语学校 汪飞鸿

## 摘 要

本文中介绍在 $O(n\log^2\log n)$ 时间复杂度内进行 Dirichlet 卷积的方法, 并给出其正确性和时间复杂度的证明, 并给出不改变其时间复杂度的情况下将其空间复杂度优化至 $O(n)$ 的方法

## 1 前置知识

### 1.1 Powerful Number

对于一个正整数 $n$, 若对于其所有质因数 $p$, 均有 $p^2 \mid n$, 则称其为 Powerful Number。$PN$ 为这样的数的集合。

通过对每个不同的质因数 $p$ 分别考虑, 对任意一个这样的数, 可以将其表示为 $a^2b^3$ 的形式, 其中 $a, b \in \mathbb{Z}$。

### 1.2 Square-free Number

对于一个正整数 $n$, 若 $\mu(n) \neq 0$, 称其为 Square-free Number, $SF$ 为这样的数的集合。

通过对每个不同的质因数 $p$ 分别考虑, 对一个正整数, 可以将其表示为 $sd$ 的形式, 其中 $s \in SF, d \in PN$。

### 1.3 子集卷积

子集卷积是给定长为 $2^n$ 的序列 $a, b$, 求序列 $c$ 满足

$$c_i = \sum_{\substack{j\&k=0,\\ j\mid k=i}} a_j b_k$$

的问题。

通过将问题转化为求所有

$$[x^{\text{popcount}(i)}]\sum_{i=j|k}(a_j x^{\text{popcount}(j)})(b_k x^{\text{popcount}(k)})$$

可以在 $O(2^n n^2)$ 的时间内解决子集卷积问题。

通过使用插值进行优化, 可以在时间复杂度不变的情况下用 $O(2^n)$ 的空间复杂度解决子集卷积问题。

## 2 SF 处的 Dirichlet 卷积计算

注意到, 仅考虑 SF 处的值时, Dirichlet 卷积和子集卷积在某种程度上是相似的: 每个质数在一个 SF 中的数中只会出现一次, 每个二进制位在每个数中也只出现一次, 并且对于每个质数/二进制位分别考虑时, 其要求是相同的。考虑对做子集卷积的方法做修改去做 Dirichlet 卷积。

显然, 最终需要制造的是

$$\sum_{st=n} f(s)g(t)$$

其中 $f, g$ 是数列。

类似于子集卷积的, 考虑对 Dirichlet 卷积的式子加入占位多项式, 其次数为其质因数个数 $\omega(i)$ (对应子集卷积中占位多项式的次数 popcount($i$)), 因此, 考虑多项式的序列 $F_i = \sum f(i)x^{\omega(i)}$ 和 $G_i = \sum g(i)x^{\omega(i)}$, 则所求为所有

$$[x^{\omega(i)}]\sum_{i=\text{lcm}(j,k)}(f(j)x^{\omega(j)})(g(k)x^{\omega(k)})$$

对于 $F$ 的前 $n$ 项的总次数, 有:

$$\begin{aligned}
&\sum_{i=1}^{n}\omega(i)\\
=&\sum_{i\in P}\sum_{j=1}\left\lfloor\frac{n}{i^j}\right\rfloor\\
\le&\sum_{i\in P}\sum_{j=1}\frac{n}{i^j}\\
\le&\sum_{i\in P}\frac{n}{i-1}
\end{aligned}$$

注意到, 对于 $P$ 中的数 $i$, 有 $i \geq 2$ 成立, 从而有:

$$\begin{aligned}&\frac{n}{i-1}\\=&\frac{n}{i}\times\frac{i}{i-1}\\=&\frac{n}{i}\times(1+\frac{1}{i-1})\\\leq&\frac{n}{i}\times 2\end{aligned}$$

因此:

$$\begin{aligned}&\sum_{i=1}^{n}\omega(i)\\\leq&\sum_{i\in P}\frac{n}{i-1}\\\leq&\sum_{i\in P}\frac{2n}{i}\\=&O(n\log\log n)\quad\text{(参考 Eratosthenes 筛的时间复杂度证明)}\end{aligned}$$

考虑如何处理 $i=\operatorname{lcm}(j,k)$ 的限制。这要求有 $j\mid i$ 且 $k\mid i$ 成立, 不难想到进行 Dirichlet 前缀和 (对应高维前缀和) 和容斥 (对应容斥), 利用 Mobius 反演 (对应二项式反演) 来保证每对 $j,k$ 仅在 $\operatorname{lcm}(j,k)$ 处有贡献, 即求 $[x^{\omega(i)}]Q(P(F)\otimes P(G))$, 其中

$$P(F)_n=\sum_{d\mid n}F_d,\quad (F\otimes G)_n=F_nG_n,\quad Q(F)_n=\sum_{d\mid n}\mu(n/d)F_d$$

接下来证明计算的结果的正确性, 即使用该方法确实能求出所需的答案:

$$\begin{aligned}&[x^{\omega(n)}]Q(P(F)\otimes P(G))_n\\=&[x^{\omega(n)}]\sum_{d\mid n}\mu(n/d)(P(F)\otimes P(G))_d\quad\text{(Q 的定义)}\\=&[x^{\omega(n)}]\sum_{d\mid n}\mu(n/d)P(F)_dP(G)_d\quad\text{(}\otimes\text{ 的定义)}\\=&[x^{\omega(n)}]\sum_{d\mid n}\mu(n/d)\left(\sum_{s\mid d}F_s\right)\left(\sum_{t\mid d}G_t\right)\quad\text{(P 的定义)}\\=&[x^{\omega(n)}]\sum_{s\mid n,t\mid n}F_sG_t\sum_{\substack{\operatorname{lcm}(s,t)\mid d,\\ d\mid n}}\mu(n/d)\end{aligned}$$

令 $m = n/\mathrm{lcm}(s,t), u = d/\mathrm{lcm}(s,t)$, 则有:

$$\begin{aligned}
&[x^{\omega(n)}]Q(P(F)\otimes P(G))_n\\
=&[x^{\omega(n)}]\sum_{s|n,t|n}F_sG_t\sum_{\substack{\mathrm{lcm}(s,t)|d,\\ d|n}}\mu(n/d)\\
=&[x^{\omega(n)}]\sum_{s|n,t|n}F_sG_t\sum_{u|m}\mu(m/u)
\end{aligned}$$

由 Mobius 反演, 有:

$$\begin{aligned}
&[x^{\omega(n)}]Q(P(F)\otimes P(G))_n\\
=&[x^{\omega(n)}]\sum_{s|n,t|n}F_sG_t\sum_{u|m}\mu(m/u)\\
=&[x^{\omega(n)}]\sum_{s|n,t|n}F_sG_t[m=1]\\
=&[x^{\omega(n)}]\sum_{\mathrm{lcm}(s,t)=n}F_sG_t\\
=&[x^{\omega(n)}]\sum_{\mathrm{lcm}(s,t)=n}f(s)x^{\omega(s)}g(t)x^{\omega(t)}\quad\text{(F 和 G 的定义)}\\
=&[x^{\omega(n)}]\sum_{\mathrm{lcm}(s,t)=n}f(s)g(t)x^{\omega(s)+\omega(t)}\\
=&\sum_{\substack{\mathrm{lcm}(s,t)=n,\\ \omega(s)+\omega(t)=\omega(n)}}f(s)g(t)
\end{aligned}$$

注意到对正整数 $s,t$ 有 $st = \gcd(s,t)\mathrm{lcm}(s,t)$, 因此有 $\omega(s)+\omega(t) = \omega(\gcd(s,t))+\omega(\mathrm{lcm}(s,t))$, 又因为求和的条件中 $\omega(s)+\omega(t) = \omega(n) = \omega(\mathrm{lcm}(s,t))$, 则有 $\omega(\gcd(s,t)) = 0$, 即 $\gcd(s,t) = 1$, 即 $st = \mathrm{lcm}(s,t) = n$, 因此实际上求和的条件为所需的求和的条件 $st = n$ 的充分条件

若 $st = n$, 则 $\omega(s)+\omega(t) = \omega(n)$, 且由于 $n \in SF$, 一定有 $\gcd(s,t) = 1$, 从而 $\mathrm{lcm}(s,t) = n$, 因此实际上的条件也为所需的条件的必要条件

综上, 求和中的条件与 $st = n$ 等价, 即:

$$\begin{aligned}
&[x^{\omega(n)}]Q(P(F)\otimes P(G))_n\\
=&\sum_{\substack{\mathrm{lcm}(s,t)=n,\\ \omega(s)+\omega(t)=\omega(n)}}f(s)g(t)\\
=&\sum_{st=n}f(s)g(t)
\end{aligned}$$

因此所求得的答案正确. 考虑计算进行各种操作的时间复杂度, 对于 $P$, 这相当于对这些多项式做 Dirichlet 前缀和, 易证 $P(F)_i$ 的次数也不超过 $\omega(i)$。考虑对每个质数的贡献求和,

即:

$$
\begin{aligned}
&\sum_{p\in P}\sum_{i\le n/p}\omega(i)\\
=&\sum_{p\in P}\sum_{q\in P}\sum_{i=1}\left\lfloor\frac{n/p}{q^i}\right\rfloor\\
\le&\sum_{p\in P}\sum_{\substack{q\in P,\\ q\le n/p}}\sum_{i=1}\frac{n/p}{q^i}\\
=&\sum_{p\in P}\sum_{\substack{q\in P,\\ q\le n/p}}\frac{n/p}{q-1}\\
\le&\sum_{p\in P}\sum_{\substack{q\in P,\\ q\le n/p}}\frac{2n/p}{q}\\
=&O\left(\sum_{p\in P}n/p\log\log n/p\right)\quad\text{(参考 Eratosthenes 筛的时间复杂度证明)}\\
=&O\left(\left(\sum_{p\in P}n/p\right)\times\log\log n\right)\\
=&O\left(n\log^2\log n\right)\quad\text{(参考 Eratosthenes 筛的时间复杂度证明)}
\end{aligned}
$$

对于 $\otimes$, 这相当于对每个位置上做多项式乘法的时间复杂度求和。每个位置上的相乘的多项式的长度均为 $\omega(i)$, 此后为 $2\omega(i)=O(\omega(i))$, 因此总时间复杂度为:

$$
\begin{aligned}
&\sum_{i=1}^{n}\omega(i)^2\\
=&\sum_{p\in P}\sum_{i=1}^{n/p}\omega(ip)\\
=&\sum_{p\in P}\sum_{i=1}^{n/p}(\omega(i)+1)\\
=&\sum_{p\in P}\sum_{i=1}^{n/p}\omega(i)+\sum_{p\in P}\left\lfloor\frac{n}{p}\right\rfloor
\end{aligned}
$$

类似的使用上述分析方法, 得:

$$
\begin{aligned}
&\sum_{i=1}^{n}\omega(i)^2\\
=&\sum_{p\in P}\sum_{i=1}^{n/p}\omega(i)+\sum_{p\in P}\left\lfloor\frac{n}{p}\right\rfloor\\
=&O\left(n\log^2\log n\right)
\end{aligned}
$$

对于 $Q$, 由于仅在 SF 中的数处求值, 有:

$$\begin{aligned}
&Q(F)_n\\
=&\sum_{d|n}\mu(n/d)F_d\\
=&\sum_{d|n}(-1)^{\omega(n/d)}F_d\\
=&\sum_{d|n}(-1)^{\omega(n)-\omega(d)}F_d\\
=&\sum_{d|n}(-1)^{\omega(n)}(-1)^{\omega(d)}F_d\\
=&(-1)^{\omega(n)}\sum_{d|n}(-1)^{\omega(d)}F_d
\end{aligned}$$

显然, 设 $R(F)_n=(-1)^{\omega(i)}F_n$, 则 $Q(F)=R(P(R(F)))$。由于 $F_i$ 的次数此时仍为 $O(\omega(i))$, 故求 $R(F)$ 的时间复杂度为 $O\left(\sum_{i=1}^{n}\omega(i)=n\log\log n\right)$, 因此求 $Q(F)$ 的时间复杂度为它和求 $P$ 的时间复杂度之和, 即 $O(n\log^2\log n)$。

综上, 求 $Q(P(F)\otimes P(G))$ 的时间复杂度为 $O(n\log^2\log n)$。

## 3 非 SF 处的 Dirichlet 卷积计算

显然对每个正整数 $n$ 存在唯一的 $n=ds$ 满足 $d\perp s, d\in PN, s\in SF$, 上述为在 $d=1$ 时计算 Dirichlet 卷积的方法, 考虑 $d>1$ 时的情况。

记 $h=f*g$, 则当 $ij=n$ 时, $f(i)g(j)$ 对 $h(n)$ 有贡献, 设 $v=\gcd(i,d)$, 则 $(i/v)(j/(d/v))=n/d=s$, 且 $s\in SF$, 因此可以设 $f_{d,v}(i)=f(iv)[i\perp d]$, $g_{d,v}(i)=g(id/v)[i\perp d]$, 然后计算 $f_{d,v}*g_{d,v}$ 在 SF 处的 Dirichlet 卷积, 并将每个 $s$ 处的值累加到对应的 $n$ 上。

因此, 总的时间复杂度为:

$$\begin{aligned}
&O\left(\sum_{d\in PN}\sum_{v|d}(n/d)\log^2\log(n/d)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2b^3\le n}\sum_{v|a^2b^3}(1/a^2b^3)\right)\quad\text{(将}d\text{ 表示为}a^2b^3\text{ 的形式)}
\end{aligned}$$

设有 $xy = v$, 则下述转化只会重复不会遗漏

$$
\begin{aligned}
&O\left(\sum_{d\in PN}\sum_{v|d}(n/d)\log^2\log(n/d)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2b^3\le n}\sum_{v|a^2b^3}(1/a^2b^3)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{x|a^2}\sum_{b^3\le n/a^2}\sum_{y|b^3}(1/a^2b^3)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{x|a^2}\sum_{b^3\le n/a^2}\sqrt{b^3}(1/a^2b^3)\right)\quad (y=2\sqrt{x}\text{ 是因数个数的上界})\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{x|a^2}\sum_{b^3\le n/a^2}(1/a^2b^{1.5})\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{x|a^2}(1/a^2)\right)\quad (\text{自然数的-1.5 次方和收敛})
\end{aligned}
$$

考虑将 $x$ 分解为 $p^2q$, 其中 $q\in SF$, 易证分解方法唯一, 则:

$$
\begin{aligned}
&O\left(\sum_{d\in PN}\sum_{v|d}(n/d)\log^2\log(n/d)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{x|a^2}(1/a^2)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{p^2q|a^2}(1/a^2)\right)\\
=&O\left(n\log^2\log n\times\sum_{a^2\le n}\sum_{p|a}\sum_{q|a/p}(1/a^2)\right)\\
=&O\left(n\log^2\log n\times\sum_{p^2\le n}\sum_{z^2\le n/p^2}\sum_{q|z}(1/p^2z^2)\right)\quad (z=a/p)\\
=&O\left(n\log^2\log n\times\sum_{p^2\le n}\sum_{q^2\le n/p^2}\sum_{t^2\le n/p^2/q^2}(1/p^2q^2t^2)\right)\quad (t=z/q)\\
=&O\left(n\log^2\log n\times\sum_{p^2\le n}\sum_{q^2\le n/p^2}(1/p^2q^2)\right)\quad (\text{自然数的-2 次方和收敛})\\
=&O\left(n\log^2\log n\times\sum_{p^2\le n}(1/p^2)\right)\quad (\text{自然数的-2 次方和收敛})\\
=&O\left(n\log^2\log n\right)\quad (\text{自然数的-2 次方和收敛})
\end{aligned}
$$

# 4 优化

注意到子集卷积中可以通过插值的方法将空间复杂度优化至线性。经过 $O(\text{poly}(n))$ 的预处理后, 最初的单项式求所有点值和最后的已知所有点值求单个系数的时间复杂度也优化到了 $O(n)$, 从而保持了时间复杂度不变。

考虑对 Dirichlet 卷积做类似的优化, 其中由于占位多项式的次数为 $O(\log n)$, 则预处理的时间复杂度为 $O(\text{poly}(\log n))$。计算 $P(F)$ 仅需做多项式的加法, 则仅需将点值相加, 但会把短的多项式加到长的上面, 因此 $F_i(x_j)$ 需要在存在 $ki \le n$ 满足 $\omega(ki) \ge j$ 时计算。注意到, 这相当于求最大的 $\omega(ki)$, 即求最大的 $\omega(k)$ 满足 $k \le n/i$。显然, 当 $k = 2^{\lfloor \log n/i \rfloor}$ 时, 这取到最大值 $\lfloor \log n/i \rfloor$, 因此 $\omega(ki) = \omega(i) + \omega(k) \le \omega(i) + \log n/i$。注意到, 和先前不同的是, 现在有 $P(F)_i$ 需计算的点值个数 $cnt(i) \ge cnt(pi) + O(1)$, 因此计算时应枚举所有 $pi$ 而非枚举所有 $i$ (特别的, 枚举所有需计算该点值的 $i \le n/p$ 的时间复杂度为 $\Theta(n \log n)$), 则计算所有所需的 $P(F)(x_j)$ 的时间复杂度为:

$$
\begin{aligned}
&\sum_{p\in P, p\le n} \sum_{i=1}^{n/p} (\omega(pi) + \log n/p/i) \\
=&O(n \log^2 \log n) + \sum_{p\in P, p\le n} \sum_{i=1}^{n/p} \log(n/p)/i \\
=&O(n \log^2 \log n) + \sum_{p\in P, p\le n} \sum_{k=0}^{\log n/p} \sum_{k\le \log(n/p)/i<k+1} k+1 \\
=&O(n \log^2 \log n) + \sum_{p\in P, p\le n} \sum_{k=0}^{\log n/p} n(k+1)/p/2^k \\
=&O(n \log^2 \log n) + \sum_{p\in P, p\le n} n/p \\
=&O\left(n \log^2 \log n\right)
\end{aligned}
$$

且计算 $\otimes$ 所需的点值的时间复杂度为:

$$
\begin{aligned}
&\sum_{i=1}^{n} (\omega(i) + \log n/i)^2 \\
=&\sum_{i=1}^{n} \omega(i)^2 + 2\sum_{i=1}^{n} \omega(i) \log n/i + \sum_{i=1}^{n} \log^2 n/i \\
=&O(n \log^2 \log n) + O(n \log \log n) + O(n) \\
=&O\left(n \log^2 \log n\right)
\end{aligned}
$$

同理于 $P$, 计算 $Q$ 的时间复杂度也为 $O(n \log^2 \log n)$。而此时空间复杂度降低至 $O(n)$。实现时, 可以用 unordered_map 等结构记录当前考虑的位置的键值对, 用 vector 等结构记录需要计算当前点值的所有位置排序后的结果, 并在做 $P$ 前预处理每个质数需要计算的位置, 否则时间复杂度容易退化。

## 5 总结

在本文中, 介绍了一种实现 Dirichlet 卷积的方法, 其时间复杂度为 $O(n \log^2 \log n)$, 空间复杂度为 $O(n)$。

虽然时间复杂度较为优秀, 但这个做法实现较为困难 (代码长度为暴力实现的几十倍), 并且在 OI 的数据范围中的表现不如暴力实现, 目前的效率只存在于理论当中。期待有更加有效的实现方法出现, 并在 OI 中推广。

## 6 致谢

感谢中国计算机协会提供学习和交流的平台。
感谢国家集训队教练任舍予的指导。
感谢南京外国语学校张超老师的关心与指导。
感谢父母对我的培育和教诲。
感谢金怀恩同学提供的帮助。
感谢所有帮助我的同学、老师。

## 7 参考文献

[1] cp-algorithms contributors. Sieve of Eratosthenes [EB/OL]. https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html, 2026.

[2] 洛谷. Dirichlet 卷积相关算法 [EB/OL]. https://www.luogu.com.cn/article/gystixdk, 2026.

[3] OI Wiki 社区. 莫比乌斯函数与莫比乌斯反演 [EB/OL]. https://oi.wiki/math/number-theory/mobius/, 2026.

# 浅谈树上邻域与链邻域的并交问题

杭州第二中学 汪苏轶

## 摘 要

本文介绍了树上邻域理论，树上邻域求并与求交方法，以及树上链邻域求交的理论与算法，并给出了这些方法在相关问题中的一些应用。

## 引言

树结构相关问题是算法竞赛中的重要研究对象，其中树上邻域类问题亦十分常见。

由树上邻域的概念出发，可以自然地类比到平面几何中的圆域问题。经过研究发现，树上邻域在进行并与交等运算时，呈现出与平面圆域相似的结构性与美感；进一步推广至树上链邻域后，这些良好的结构性质在一定程度上仍得以保留。

本文将围绕树上邻域的并、交运算以及树上链邻域的交运算展开讨论，并结合具体问题，对相关性质进行简单的的分析与探讨。

## 1 记号与约定

设 $T = (V, E)$ 是一张无向图，其中 $V$ 为非空顶点集合，$E$ 为边集合。称 $T$ 是一棵树，当且仅当 $T$ 联通且无环。

树 $T$ 上任意两点之间的简单路径与距离可按如下方式定义。

**定义 1.1** (简单路径)**.** 对于任意 $s, t \in V$，定义其简单路径为

$$\mathrm{Path}^r(s, t) = (s = v_0, v_1, \ldots, v_k = t),$$

其中 $v_i \in V$，且满足：

1. 对所有 $0 \le i < k$，$(v_i, v_{i+1}) \in E$；
2. $v_0, v_1, \ldots, v_k$ 两两不同。

在树中，任意两点间的简单路径唯一。也称 $s, t$ 之间的唯一简单路径为连接 $s, t$ 的链。

**定义 1.2** (距离)**.** 若 $\mathrm{Path}^r(s,t)=(v_0,\dots,v_k)$，则称

$$d_T^r(s,t)=k$$

为 $s$ 与 $t$ 的树上距离。

为了描述树上的边中点等结构，尝试扩展原有点集。

**定义 1.3** (广义点)**.** 定义广义点集合

$$\mathcal{V}=V\cup\{\mathcal{M}(u,v)\mid(u,v)\in E\},$$

其中 $\mathcal{M}(u,v)$ 表示边 $(u,v)$ 的中点。

**定义 1.4** (广义距离)**.** 先定义基本距离 $d_e:\mathcal{V}\times V\to\mathbb{R}\cup\{+\infty\}$：

$$d_e(u,x)=\begin{cases}0, & u=x,\\ +\infty, & \text{otherwise},\end{cases}$$

$$d_e(\mathcal{M}(u,v),x)=\begin{cases}\frac{1}{2}, & x=u\text{ 或}x=v,\\ +\infty, & \text{otherwise}.\end{cases}$$

广义距离 $dis:\mathcal{V}\times\mathcal{V}\to\mathbb{R}$ 定义为

$$dis(p,q)=\min_{x,y\in V}(d_T^r(x,y)+d_e(p,x)+d_e(q,y)).$$

**定义 1.5** (广义路径)**.** 定义广义路径为

$$\mathrm{Path}(p,q)=\{x\in\mathcal{V}\mid dis(p,x)+dis(x,q)=dis(p,q)\}.$$

表示 $p$ 与 $q$ 在树上路径中经过的所有广义点。

**定义 1.6** (行走)**.** 定义行走函数 $\curvearrowright:\mathcal{V}\times\mathcal{V}\times\mathbb{R}\to\mathcal{V}$，其中 $\curvearrowright(s,t,\theta)$ 表示从 $s$ 朝向 $t$ 行走距离 $\theta$ 后到达的唯一点（$0\le\theta\le d_T(s,t)$）。形式化为：

$$\curvearrowright(s,t,\theta)=p\quad\text{where } p\in\mathrm{Path}(s,t),\ d_T(s,p)=\theta.$$

上述关于广义点的相关定义仅为了辅助接下来树上邻域理论的描述，其基本保留了传统树上路径、距离等的性质。

**定义 1.7** (树上邻域)**.** 定义广义点 $u$ 在树上的 $r$ 邻域为所有与 $u$ 之距离 $\le r$ 的点所构成的集合，记作 $N(u,r)$。形式化的，有：

$$N(u,r)=\{x\in\mathcal{V}\mid dis(u,x)\le r\}$$

称 $u$ 为邻域 $N(u,r)$ 的中心，$r$ 为该邻域的半径。

称一个邻域 $N(u,r)$ 是真实的，当且仅当满足以下两个条件之一：

$$\begin{cases} u \in V, r \in \mathbb{Z} \\ u \notin V, r = p + \frac{1}{2}, p \in \mathbb{Z} \end{cases}$$

可理解为该邻域的所有边界点都是树上真实的顶点。

本文中所讨论的所有邻域均为真实邻域。接下来如果未经特殊说明，所有的邻域均指真实邻域。

**定义 1.8** (点到链的距离)**.** 对于树上的顶点 $u$ 和一条连接顶点 $s,t$ 的链 $\mathrm{Path}(s,t)$，定义 $u$ 到 $\mathrm{Path}(s,t)$ 的距离：

$$dis(u, \mathrm{Path}(s,t)) = \min_{x \in Path(s,t)} dis(u,x)$$

同时，若上述式子中取到最小值的 $x = x_0$，则称 $u$ 属于 $\mathrm{Path}(s,t)$ 的 $x_0$，或者 $u$ 在链 $\mathrm{Path}(s,t)$ 上属于 $x_0$。

**定义 1.9** (链到链的距离)**.** 对于树上任意两条链 $\mathrm{Path}(u_1,v_1)$ 和 $\mathrm{Path}(u_2,v_2)$，定义这两条链之间的距离：

$$dis(\mathrm{Path}(u_1,v_1), \mathrm{Path}(u_2,v_2)) = \min_{x \in \mathrm{Path}(u_1,v_1)} \min_{y \in \mathrm{Path}(u_2,v_2)} dis(x,y)$$

**定义 1.10** (树上链邻域)**.** 定义一条连接顶点 $u,v$ 的链的 $r$ 邻域为所有与链 $\mathrm{Path}(u,v)$ 距离 $\le r$ 的点所构成的集合，记作 $M(u,v,r)$。形式化的，有：

$$M(u,v,r) = \{x \in V \mid dis(x, \mathrm{Path}(u,v)) \le r\}$$

类似的，称 $\mathrm{Path}(u,v)$ 为链邻域 $M(u,v,r)$ 的中心链，$r$ 为该链邻域的半径。

本文中讨论的所有有关树上链邻域的问题均不包含广义点。

# 2 树上邻域理论

## 2.1 直径与树上邻域相关性质

本小节将首先研究一些有关树上邻域的基本性质。

由树上邻域的定义 $N(u,r)$ 不难联想到平面上的圆 $circle(o,r)$，故树上邻域在一些场合也被称之为“树上圆”。对于平面上的一个圆，连接圆内任意两点的最长直线即为直径，而所有的直径都会经过圆心一点。由此容易引导将该结论类比到树上。

对于树上的一个由顶点构成的点集 $S$，定义点集 $S$ 的直径长度 $d(S)$ 为 $\max_{x,y \in S} dis(x,y)$，而任意一条满足 $x,y \in S$ 且 $dis(x,y) = d(S)$ 的路径 $\mathrm{Path}(x,y)$ 都被称为点集 $S$ 的直径。有如下定理：

**定理 2.1.** 对于树上任意一个点集 $S$，$S$ 的所有直径中点重合（中点可能为广义点）。

证明. 反证。若存在 $S$ 的两条不同直径 $\mathrm{Path}(x_1, y_1)$ 和 $\mathrm{Path}(x_2, y_2)$ 使得两条直径的中点 $c_1 \neq c_2$，则 $\mathrm{Path}(c_1, x_1)$ 和 $\mathrm{Path}(c_1, y_1)$ 中比有一条路径与 $\mathrm{Path}(c_1, c_2)$ 只在 $c_1$ 相交，不妨设为 $\mathrm{Path}(c_1, x_1)$；同理有 $\mathrm{Path}(c_2, x_2)$。那么 $dis(x_1, x_2) = dis(x_1, c_1) + dis(x_2, c_2) + dis(c_1, c_2) = d(S) + dis(c_1, c_2)$，与直径的定义中的最大性不符，矛盾。 □

故对于一个点集 $S$，可以定义点集的中点 $m(S)$ 为一个广义点。找到中点和直径后，就可以类似平面上的圆，定义点集 $S$ 对应的树上邻域 $\mathcal{K}(S) = N(m(S), \frac{d(S)}{2})$。容易发现 $\mathcal{K}(S)$ 必然是一个真实邻域。

接下来考虑一些从平面中的圆覆盖所引导出的性质：

**定理 2.2.** 对于树上任意一个点集 $S$ 与真实邻域 $N(u, r)$，若 $S \subset N(u, r)$，则 $\mathcal{K}(S) \subset N(u, r)$。

证明. 首先证明 $dis(u, m(S)) \leq r - \frac{d(S)}{2}$。若不然，则取任意一条 $S$ 的直径 $\mathrm{Path}(x, y)$，必有 $x, y$ 中的一者在 $m(S)$ 的与 $u$ 不同的一棵子树中。不妨设 $x$ 是，那么由 $x \in S \subset N(u, r)$ 可知 $dis(x, u) = dis(x, m(S)) + dis(m(S), u) = \frac{d(S)}{2} + dis(m(S), u) \leq r$，因此 $dis(m(S), u) \leq r - \frac{d(S)}{2}$。

从而对于任意一点 $t \in CC(S) = N(m(S), \frac{d(S)}{2})$，都有 $dis(t, u) \leq dis(t, m(S)) + d(m(S), u) \leq \frac{d(S)}{2} + (r - \frac{d(S)}{2}) = r$，因此 $t \in N(u, r)$。进而可得 $\mathcal{K}(S) \subset N(u, r)$。 □

## 2.2 树上邻域求并

在讨论平面上的圆求并问题时，通常会考虑求解最小圆覆盖。即用一个最小的圆覆盖所有给定的圆。扩展到树上的情形时，也希望得到类似结论。

本节将探索的主要问题是：对于树上的两个点集 $S, T$，$\mathcal{K}(S), \mathcal{K}(T)$ 和 $\mathcal{K}(S \cup T)$ 之间的关系。换句话说，若只知道 $\mathcal{K}(S), \mathcal{K}(T)$，能否在不知道 $S, T$ 本身为何值时，快速求出 $\mathcal{K}(S \cup T)$。

考虑对 $\mathcal{K}(S)$ 和 $\mathcal{K}(T)$ 的包含关系进行讨论：

- 若 $\mathcal{K}(S) \subset \mathcal{K}(T)$，即 $dis(m(S), m(T)) + \frac{d(S)}{2} \leq \frac{d(T)}{2}$，则显然 $\mathcal{K}(S \cup T) = \mathcal{K}(T)$。
- 若 $\mathcal{K}(T) \subset \mathcal{K}(S)$，即 $dis(m(S), m(T)) + \frac{d(T)}{2} \leq \frac{d(S)}{2}$，则显然 $\mathcal{K}(S \cup T) = \mathcal{K}(S)$。
- 否则有 $\mathcal{K}(S) \not\subset \mathcal{K}(T)$ 且 $\mathcal{K}(T) \not\subset \mathcal{K}(S)$，即两者互不包含。结论为：

$$\mathcal{K}(S \cup T) = N\left(\frown\left(m(S), m(T), \frac{len - d(S)/2 + d(T)/2}{2}\right), \frac{len + d(S)/2 + d(T)/2}{2}\right)$$

其中 $len = dis(m(S), m(T))$。

尝试证明上述结论。对于点集 $S$，必然能找到直径 $\mathrm{Path}(u_1, v_1)$，不妨设 $u_1$ 不与 $m(T)$ 在 $m(S)$ 的相同子树中；同理能在 $T$ 中找到直径 $\mathrm{Path}(u_2, v_2)$，$u_2$ 不与 $m(S)$ 在 $m(T)$ 的相同

子树中。则 $dis(u_1,u_2)=dis(u_1,m(S))+dis(m(S),m(T))+dis(m(T),u_2)=len+d(S)/2+d(T)/2$，该值一定是 $S\cup T$ 的直径长度。

从而这条直径的中点就是 $\curvearrowright(u_1,u_2,\frac{len+d(S)/2+d(T)/2}{2})=\curvearrowright(m(S),m(T),\frac{len-d(S)/2+d(T)/2}{2})$。

通过上述讨论可知，在只知道 $\mathcal{K}(S)$ 和 $\mathcal{K}(T)$ 的情况下就可以直接求出 $\mathcal{K}(S\cup T)$，该结论也通常被称为树上圆理论。拥有该结论后，若需要维护点集的直径并支持合并，可以不维护整个点集 $S$，而是只维护 $\mathcal{K}(S)$。

该种维护方式相较于传统的维护点集内任意一条直径更加简洁，在下面的例题中很容易看出。

**例题 1** (Range Diameter Sum[1]). 给定一棵包含 $n$ 个点的树，定义：

$$\mathrm{diam}(l,r)=\max_{l\le u,v\le r} dis(u,v)$$

计算：

$$\sum_{1\le l\le r\le n}\mathrm{diam}(l,r)$$

数据范围：$1\le n\le 10^5$。

解法　求单个 $\mathrm{diam}(l,r)$ 即为树上邻域的标准形式。现在需要求解所有子区间的答案之和，求解类似问题的经典做法是使用分治。

对于一个分治区间 $[l,r]$，令 $mid=\lfloor\frac{l+r}{2}\rfloor$，递归分治求解 $[l,mid]$ 和 $[mid+1,r]$ 两个区间的所有子区间答案，而在当前分治过程中计算跨过 $mid$ 的区间答案。

预处理出所有形如 $\mathcal{K}([x,mid])$ 和 $\mathcal{K}([mid+1,y])$ 的树上圆，接下来即求这些树上圆两两并半径之和。

固定 $x$，则 $\mathcal{K}([x,mid])$ 被确定。$y$ 从 $mid+1$ 增加到 $y$ 的过程中，$\mathcal{K}([mid+1,y])$ 逐渐增大，因此 $\mathcal{K}([mid+1,y])$ 和 $\mathcal{K}([x,mid])$ 的关系也可以被分为三段：

- 第一段中，$\mathcal{K}([x,mid])\supset\mathcal{K}([mid+1,y])$，此时贡献即为 $\mathcal{K}([x,mid])$ 的直径。
- 第二段中，两个树上圆互不包含。根据树上圆求并的讨论，两个树上圆求并后新树上圆的直径是 $dis(m([x,mid]),m([mid+1,r]))+d([x,mid])/2+d([mid+1,y])/2$（此处分别用 $m(S)$ 和 $d(S)$ 来表示 $\mathcal{K}(S)$ 的中点和直径）。
- 第三段中，$\mathcal{K}([x,mid])\subset\mathcal{K}([mid+1,y])$，此时贡献即为 $\mathcal{K}([mid+1,y])$。

对于第一段和第三段，贡献都是容易处理的。对于第二段 $d([x,mid])+d([mid+1,y])$ 部分也容易计算。而 $dis(m[x,mid],m[mid+1,y])$ 部分可以转化为以下问题：动态维护一个可重点集 $S$，需要支持以下两种操作：

[1]题目来源：Codeforces Round 691 (Div. 1) F. Range Diameter Sum https://codeforces.com/contest/1458/problem/F.

- 向 $S$ 中插入或删除顶点。
- 给定顶点 $x$，查询 $\sum_{y\in S} dis(x,y)$。

该问题可以使用点分树或重链剖分解决。视最后一步的具体维护与实现方式，复杂度在 $O(n\log^2 n)$ 到 $O(n\log^3 n)$ 不等。 ■

通过上述例题可以发现，树上圆理论能够让点集之间的包含关系刻画更加具体。

## 2.3 树上邻域求交

本小节中将主要讨论树上真实邻域求交的问题。经过分析得到的最终结论为：树上任意两个真实邻域求交的结果必为真实邻域或空集。接下来将对该结论进行证明。

**定理 2.3.** 树上任意两个真实邻域 $N(u_1,r_1)$ 和 $N(u_2,r_2)$ 的交必为真实邻域或空集。

证明. 容易观察到两个真实邻域之交为空当且仅当 $dis(u_1,u_2) > r_1 + r_2$。否则必有：

$$N(u_1,r_1)\cap N(u_2,r_2) = N\left(\frown\left(u_1,u_2,\frac{r_1-r_2+dis(u_1,u_2)}{2}\right),\frac{r_1+r_2-dis(u_1,u_2)}{2}\right)$$

令 $len = dis(u_1,u_2)$，广义点 $c = \frown(u_1,u_2,\frac{r_1-r_2+len}{2})$。根据 $\frown$ 的性质必然有 $dis(u_1,c) = \frac{r_1-r_2+len}{2}$ 和 $dis(u_2,c) = len - \frac{r_1-r_2+len}{2} = \frac{r_2-r_1+len}{2}$。则对于任意 $t\in N(c,\frac{r_1+r_2-len}{2})$，都有：

$$\begin{cases} dis(t,u_1)\le dis(t,c)+dis(c,u_1)\le \frac{r_1+r_2-len}{2}+\frac{r_1-r_2+len}{2} = r_1 & \Longrightarrow\ t\in N(u_1,r_1)\\ dis(t,u_2)\le dis(t,c)+dis(c,u_2)\le \frac{r_1+r_2-len}{2}+\frac{r_2-r_1+len}{2} = r_2 & \Longrightarrow\ t\in N(u_2,r_2)\end{cases}$$

由此可知 $N(c,\frac{r_1+r_2-len}{2})\subset (N(u_1,r_1)\cap N(u_2,r_2))$。

再证明对于任意一点 $t$，若 $t\in N(u_1,r_1)$ 且 $t\in N(u_2,r_2)$，则必有 $t\in N(c,\frac{r_1+r_2-len}{2})$。反正，存在一个点使得 $t\notin N(c,\frac{r_1+r_2-len}{2})$，那么有 $dis(t,c) > \frac{r_1+r_2-len}{2}$。由于 $u_1,u_2$ 一定在 $c$ 的不同两棵子树中，因此以下两个条件之一必有一者成立：$dis(t,u_1) = dis(t,c)+dis(c,u_1)$ 或 $dis(t,u_2) = dis(t,c)+dis(c,u_2)$。不妨设成立前者，那么 $dis(t,u_1) = dis(t,c)+dis(c,u_1) > \frac{r_1+r_2-len}{2}+\frac{r_1-r_2+len}{2} = r_1$，这与 $t\in N(u_1,r_1)$ 矛盾。因此 $N(c,\frac{r_1+r_2-len}{2})\supset (N(u_1,r_1)\cap N(u_2,r_2))$

结合上述两个性质可知：$N(u_1,r_1)\cap N(u_2,r_2) = N(c,\frac{r_1+r_2-len}{2})$。 □

若视空集也是树上真实邻域，则由上述分析可以发现：树上真实邻域对求交运算封闭。

但该结论仍然不够优美，因为树上真实邻域的中心点可能并不是树上的节点。在下一节中，将引入树上链邻域的概念，从而让邻域求交的表示更加方便，不需要再使用广义点来作为邻域中点。

# 3 树上链邻域理论

## 3.1 链邻域与邻域的关系

在“记号与约定”一节中已经提及，本部分所有的链邻域均不包含广义点。也就是说，所有 $M(u,v,r)$ 中 $u,v$ 均为真实的树上顶点，$r$ 一定是整数。

上一节中已经讨论了树上邻域求交的相关问题。链邻域则是邻域的一种扩展形式，他能够表示所有真实的邻域。

**引理 3.1** (真实邻域的链邻域表示)**.** 任意一个真实邻域都可以用链邻域来表示。

证明. 对任意一个真实邻域 $N(u,r)$，分 $u$ 的类型进行讨论：

- 对于 $u \in V$ 的真实邻域 $N(u,r)$，可以直接用链邻域 $M(u,u,r)$ 来表示。
- 对于 $u \notin V$ 的真实邻域 $N(u=\mathcal{M}(p,q),r)$，可以用链邻域 $M(p,q,r-\frac{1}{2})$ 来表示。

综上，无论 $u$ 为何种类型，均可以用一个链邻域来表示该真实邻域。 □

本节中，将最终证明链邻域求交之后必然是链邻域或空集。

## 3.2 零半径树上链邻域求交

该问题亦可描述为树上两条链求交。

**定理 3.1** (链求交定理)**.** 对于树上任意两条链 $\mathrm{Path}(u_1,v_1),\mathrm{Path}(u_2,v_2)$，他们的交必为一条链或空集。

证明. 该定理是经典结论。首先他们的交必然是 $\mathrm{Path}(u_1,v_1)$ 的子集，故必为若干段链。若段数 $\geq 2$，则根据这些链也是 $\mathrm{Path}(u_2,v_2)$ 的子集可以推出 $T=(V,E)$ 中包含环，与树的定义矛盾。

故其交集必为一条链或零条链（即空集）。 □

## 3.3 等半径树上链邻域求交

本小节将证明对于任意两个半径相等的链邻域 $M(u_1,v_1,r)$ 和 $M(u_2,v_2,r)$，求交后仍然为链邻域或空集，并给出一个较为简单的表达形式。

对于树上的任意两条链 $\mathrm{Path}(u_1,v_1)$ 和 $\mathrm{Path}(u_2,v_2)$，他们之间的位置关系可以分为两种：相交和不相交。对于这两种情况分别进行讨论。

**引理 3.2.** 对于树上两条相交的链 $\mathrm{Path}(u_1, v_1), \mathrm{Path}(u_2, v_2)$ 和任意整数 $r \geq 0$，设 $\mathrm{Path}(x, y) = \mathrm{Path}(u_1, v_1) \cap \mathrm{Path}(u_2, v_2)$，则 $M(u_1, v_1, r) \cap M(u_2, v_2, r) = M(x, y, r)$。

证明. 不妨设 $u_1, u_2$ 在 $x$ 侧（即 $dis(u_1, x) \leq dis(u_1, y)$，其他同理），$v_1, v_2$ 在 $y$ 侧。

若 $t \in M(x, y, r)$，则根据定义必然存在一点 $c \in \mathrm{Path}(x, y)$ 满足 $dis(c, t) \leq r$。而显然 $c \in \mathrm{Path}(u_1, v_1), \mathrm{Path}(u_2, v_2)$，因此也有 $t \in M(u_1, v_1, r), M(u_2, v_2, r)$。这说明 $M(x, y, r) \subset (M(u_1, v_1, r) \cap M(u_2, v_2, r))$。

同时，如果一个点 $t$ 同时满足 $t \in M(u_1, v_1, r)$ 和 $t \in M(u_2, v_2, r)$，假设 $t$ 属于 $\mathrm{Path}(u_1, v_1)$ 中的 $c$。则分两种情况讨论：

1. 若 $c \in \mathrm{Path}(x, y)$，那么由 $t \in M(u_1, v_1, r)$ 可知 $dis(t, c) \leq r$，因此也有 $t \in M(x, y, r)$。

2. 否则必有 $c \in \mathrm{Path}(x, u_1)$ 或 $c \in \mathrm{Path}(v_1, y)$ 之一成立。不妨设前者成立。则此时有 $t$ 属于 $\mathrm{Path}(u_2, v_2)$ 中的 $x$，由 $t \in M(u_2, v_2, r)$ 可知 $dis(t, x) \leq r$，因此也有 $t \in M(x, y, r)$。

故无论 $c$ 在何处，必有 $t \in M(x, y, r)$，也即 $(M(u_1, v_1, r) \cap M(u_2, v_2, r)) \subset M(x, y, r)$。

结合两个包含关系可得：$M(u_1, v_1, r) \cap M(u_2, v_2, r) = M(x, y, r)$。 □

**引理 3.3.** 对于树上两条不相交的链 $\mathrm{Path}(u_1, v_1), \mathrm{Path}(u_2, v_2)$ 和任意整数 $r \geq 0$，$M(u_1, v_1, r) \cap M(u_2, v_2, r)$ 必为一个真实邻域或空集。

证明. 不妨设 $x$ 为 $\min\limits_{p \in \mathrm{Path}(u_1, v_1)} dis(p, \mathrm{Path}(u_2, v_2))$ 取到最小值时的 $p$，$y$ 为 $\min\limits_{q \in \mathrm{Path}(u_2, v_2)} dis(q, \mathrm{Path}(u_1, v_1))$ 取到最小值时的 $q$。容易证明 $x, y$ 都是唯一的。又令 $len = dis(x, y)$。

若 $len > 2r$，则显然两个链邻域无交，交集为空集。

否则必有 $len \leq 2r$。令广义点 $c = \frown(x, y, \frac{len}{2})$。

此时，发现对于所有 $dis(t, y) = dis(t, c) + dis(c, y)$ 的点 $t$（形象地说，即为在 $c$ 靠近 $x$ 一侧的点），都有 $dis(t, y) = dis(t, c) + dis(c, y) = dis(t, c) + dis(c, x) \geq dis(t, x)$，因此对于这一些点可以忽略 $M(u_1, v_1, r)$ 的限制，只需要满足 $dis(t, y) \leq r$ 即 $dis(t, c) + dis(c, y) \leq r$ 即 $dis(t, c) \leq r - dis(c, y) = r - \frac{len}{2}$ 即可。

同理，对于所有 $dis(t, x) = dis(t, c) + dis(c, x)$ 的点也都只需要满足 $dis(t, c) \leq r - \frac{len}{2}$ 即可。所有满足这两个条件之一的点构成了所有点的全集。故两个链邻域的交集即为 $N(c, r - \frac{len}{2})$。 □

结合引理 3.2，引理 3.3 和定理 3.1 可得到如下定理：

**定理 3.2** (等半径链邻域求交定理)**.** 对于任意两个半径相等的树上链邻域 $M(u_1, v_1, r)$ 和 $M(u_2, v_2, r)$，$M(u_1, v_1, r) \cap M(u_2, v_2, r)$ 必为空集，或可以被表示为链邻域的形式。

## 3.4 一般树上链邻域求交

上一小节中证明了等半径的两个链邻域相交仍是链邻域或空集。此小节将讨论半径不同时的情况。经过分析发现此性质仍然满足。下面将逐步对该性质进行证明。

仍然考虑两个链邻域的中心链的位置关心进行分类讨论。

**引理 3.4.** 对于树上两条相交的链 $\mathrm{Path}(u_1,v_1),\mathrm{Path}(u_2,v_2)$ 和任意两个整数 $r_1,r_2\geq 0$，有 $M(u_1,v_1,r_1)\cap M(u_2,v_2,r_2)$ 仍为链邻域。

证明. 不妨设 $r_1\leq r_2$，$u_1,v_1$ 在 $x$ 侧，$u_2,v_2$ 在 $y$ 侧。

首先对整棵树进行一些变换：如果 $dis(u_1,x)<r_2-r_1$，则在 $u_1$ 节点上接一条长度为 $r_2-r_1-dis(u_1,x)$ 的虚链，同时令 $u_1'$ 为虚链的链底；否则令 $u_1'=u_1$。对 $v_1$ 进行类似操作得到 $v_1'$。

现在必有 $dis(u_1',x),dis(v_1',y)\geq r_2-r_1$ 成立。考虑先求 $M(u_1',v_1',r_1)$ 和 $M(u_2,v_2,r_2)$ 的交。注意此时新添加的虚链并不影响两条链的交。

接下来说明下述等式成立：

$$M(u_1',v_1',r_1)\cap M(u_2,v_2,r_2)=M(u_1',v_1',r_1)\cap M(\curvearrowright(x,u_1',r_2-r_1),\curvearrowright(y,v_1',r_2-r_1),r_1)$$

令 $x'=\curvearrowright(x,u_1',r_2-r_1),y'=\curvearrowright(y,u_2',r_2-r_1)$。

上述添加虚链的过程保证了 $dis(u_1',x),dis(v_1',y)\geq r_2-r_1$，因此 $x'$ 和 $y'$ 必然是存在的。

首先说明 $\forall t\in M(x',y',r_1)$，都有 $t\in M(u_2,v_2,r_2)$。设 $t$ 在 $\mathrm{Path}(x',y')$ 链上属于点 $c$。

1. 若 $c\in\mathrm{Path}(x,y)$，则由 $\mathrm{Path}(x,y)\subset\mathrm{Path}(u_2,v_2)$ 可知 $dis(t,\mathrm{Path}(u_2,v_2))\leq dis(t,\mathrm{Path}(x,y))\leq r_1\leq r_2$，显然 $t\in M(u_2,v_2,r_2)$。

2. 若 $c\notin\mathrm{Path}(x,y)$，则 $c\in\mathrm{Path}(x',x)\setminus\{x\}$ 和 $c\in\mathrm{Path}(y',y)\setminus\{y\}$ 之一必然成立。不妨设成立前者，则有 $dis(t,x)=dis(t,c)+dis(c,x)\leq r_1+(r_2-r_1)=r_2$，因此 $t\in M(u_2,v_2,r_2)$。

由此可知 $M(x',y',r_1)\subset M(u_2,v_2,r_2)$，故有：

$$(M(u_1',v_1',r_1)\cap M(u_2,v_2,r_2))\supset(M(u_1',v_1',r_1)\cap M(x',y',r_1))$$

接下来说明对于任意顶点 $t$，若 $t\in M(u_1',v_1',r_1)$，$t\notin M(x',y',r_1)$，则必有 $t\notin M(u_2,v_2,r_2)$。

考虑反证。设存在这样顶点 $t$ 同时满足下面三个条件：$t\in M(u_1',v_1',r_1)$，$t\notin M(x',y',r_1)$，$t\in M(u_2,v_2,r_2)$。

设 $t$ 属于 $\mathrm{Path}(u_1',v_1')$ 的 $c$，由于 $t\notin M(x',y',r_1)$，因此必有 $c\notin\mathrm{Path}(x',y')$，即下列两个条件之一成立：$c\in\mathrm{Path}(u_1',x')\setminus\{x'\}$ 或 $c\in\mathrm{Path}(v_1',y')\setminus\{y'\}$。不妨设成立前者。由 $t\notin M(x',y',r_1)$ 可得 $dis(t,x')>r_1$。

当以 $x$ 为根时，由于 $t\in\mathrm{Path}(u_1',x')\setminus\{x'\}$，因此 $t$ 一定在 $x'$ 的子树内，进一步得到 $t$ 在链 $\mathrm{Path}(u_2,v_2)$ 上属于 $x$。故 $dis(t,\mathrm{Path}(u_2,v_2))=dis(t,x)=dis(t,x')+dis(x',x)>r_1+(r_2-r_1)=r_2$，与 $t\in M(u_2,v_2,r_2)$ 矛盾，假设不成立。

因此得到若 $t \in M(u_1', v_1', r_1)$，$t \notin M(x', y', r_1)$，则必有 $t \notin M(u_2, v_2, r_2)$，也就是说：

$$(M(u_1', v_1', r_1) \cap M(u_2, v_2, r_2)) \subset (M(u_1', v_1', r_1) \cap M(x', y', r_1))$$

结合两个结论就可以得到成立：

$$M(u_1', v_1', r_1) \cap M(u_2, v_2, r_2) = M(u_1', v_1', r_1) \cap M(\curvearrowright(x, u_1', r_2 - r_1), \curvearrowright(y, v_1', r_2 - r_1), r_1)$$

同时根据加虚链的方式容易知道 $M(u_1, v_1, r_1) \subset M(u_1', v_1', r_1')$，因此：

$$\begin{aligned}
&M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2) \\
=&M(u_1, v_1, r_1) \cap M(u_1', v_1', r_1) \cap M(u_2, v_2, r_2) \\
=&M(u_1, v_1, r_1) \cap M(u_1', v_1', r_1) \cap M(\curvearrowright(x, u_1', r_2 - r_1), \curvearrowright(y, v_1', r_2 - r_1), r_1) \\
=&M(u_1, v_1, r_1) \cap M(\curvearrowright(x, u_1', r_2 - r_1), \curvearrowright(y, v_1', r_2 - r_1), r_1)
\end{aligned}$$

由上式可知该交集其实是两个等半径链邻域求交的结果，由定理 3.2 可知结果必然是链邻域或空集。

又由 $\text{Path}(x, y) \subset M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2)$ 可知结果非空，故结果必然为链邻域的形式。 □

**引理 3.5.** 对于树上两条不交的链 $\text{Path}(u_1, v_1), \text{Path}(u_2, v_2)$ 和任意两个整数 $r_1, r_2 \geq 0$，有 $M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2)$ 必为一个链邻域或空集。

证明. 与等半径的证明方式类似，仍然不妨设 $x$ 为 $\min\limits_{p \in \text{Path}(u_1, v_1)} dis(p, \text{Path}(u_2, v_2))$ 取到最小值时的 $p$，$y$ 为 $\min\limits_{q \in \text{Path}(u_2, v_2)} dis(q, \text{Path}(u_1, v_1))$ 取到最小值时的 $q$。容易证明 $x, y$ 都是唯一的。又令 $len = dis(x, y)$。

若 $len > r_1 + r_2$ 则显然交集为空集。否则令 $w = \frac{len + r_1 - r_2}{2}$。

- 如果 $0 \leq w \leq len$，那么可以找到一广义点 $c = \curvearrowright(x, y, w)$。

  发现对于一个点 $c$ 在两个链邻域的交中的等价条件为 $r_1 - dis(c, \text{Path}(u_1, v_1)) \geq 0$ 且 $r_2 - dis(c, \text{Path}(u_2, v_2)) \geq 0$。

  此时，对于所有满足 $dis(t, y) = dis(t, c) + dis(c, y)$ 的点 $t$（形象的看为 $c$ 靠 $x$ 一侧的点），都有 $r_1 - dis(t, \text{Path}(u_1, v_1)) \geq r_1 - dis(t, x) \geq r_1 - dis(t, c) - dis(c, x) = r_1 - dis(t, c) - w = r_2 - len + w - dis(t, c) = r_2 - len(c, y) - dis(t, c) = r_2 - dis(t, y) = r_2 - dis(t, \text{Path}(u_2, v_2))$，即对于这一部分的点只需要考虑 $r_2 - dis(t, \text{Path}(u_2, v_2)) \geq 0$ 的限制即可，该限制又可被改写为 $r_2 - dis(t, c) - (len - w) \geq 0$ 即 $dis(t, c) \leq r_2 - len + w = r_1 - w$。

  对于另一边满足 $dis(t, x) = dis(t, c) + dis(c, x)$ 的点 $t$（即 $c$ 靠 $y$ 一侧的点）类似的也可将限制化简为 $dis(t, c) \leq r_1 - w$。发现这样两个点集的并集恰好为所有顶点全集。故此时满足限制的点即为 $N(c, r_1 - w)$。且发现 $N(c, r_1 - w)$ 必然是真实邻域，再由定理 3.1 即可得到结果能由链邻域的形式表示。

- 否则有 $w < 0$ 和 $w > len$ 两者之一成立。两者分别等价于 $r_2 > r_1 + len$ 于 $r_1 > r_2 + len$。不妨设成立前者，即有 $r_2 > r_1 + len$。

  接下来将说明可以将 $M(u_2, v_2, r_2)$ 的限制放缩为 $N(x, r_2 - len)$。

  首先证明 $N(x, r_2 - len) \subset M(u_2, v_2, r_2)$。若一个点 $t \in N(x, r_2 - len)$，则 $dis(t, x) \le r_2 - len$，则有 $dis(t, \text{Path}(u_2, v_2)) \le dis(t, y) \le dis(t, x) + dis(x, y) \le (r_2 - len) + len = r_2$，故 $t \in M(u_2, v_2, r_2)$。进而可得 $(M(u_1, v_1, r_1) \cap N(x, r_2 - len)) \subset (M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2))$。

  若一个点 $t \in M(u_1, v_1, r_1)$ 满足 $t \notin N(x, r_2 - len)$，则可以得到 $dis(t, x) > r_2 - len > r1$，但是 $dis(t, \text{Path}(u_1, v_1)) \le r_1$，因此 $t$ 一定属于 $\text{Path}(u_1, v_1)$ 链上的一个非 $x$ 的点。那么就有 $dis(t, \text{Path}(u_2, v_2)) = dis(t, y) = dis(t, x) + dis(x, y) > (r_2 - len) + len = r_2$，即 $t \notin M(u_2, v_2, r_2)$。进而有 $(M(u_1, v_1, r_1) \cap N(x, r_2 - len)) \supset (M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2))$。

  结合上述两个条件就有 $(M(u_1, v_1, r_1) \cap N(x, r_2 - len)) = (M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2))$。因此这两个链邻域求交的结果就是 $M(u_1, v_1, r_1)$ 和 $N(x, r_2 - len) = M(x, x, r_2 - len)$ 求交的结果，而 $x \in \text{Path}(u_1, v_1)$，因此将问题转化为了两个中心链有交的链邻域求交问题，由引理 3.4 可知，结果必然为链邻域的形式。

综上，无论 $w$ 为何值，均有结果为链邻域或空集。引理得证。 □

结合引理 3.4 和引理 3.5 可得到如下具有一般性的且优美的定理：

**定理 3.3** (链邻域求交定理)**.** 对于任意两个树上链邻域 $M(u_1, v_1, r_1)$ 和 $M(u_2, v_2, r_2)$，都有 $M(u_1, v_1, r_1) \cap M(u_2, v_2, r_2)$ 必为空集，或可以被表示为链邻域的形式。

也就是说，若将空集也视作链邻域，则一棵树的链邻域对求交操作封闭。

## 3.5 算法与例题

如果将空集也视为一种特殊的链邻域，那么容易使用一个结构体来存储链邻域。

当需要对两个链邻域求交时，算法的主体思想在上一小节的证明过程中已经给出。除去所有较为复杂的分类讨论以外，主要需要进行以下几种操作：

1. 给出树上的两个点 $u, v$，查询 $dis(u, v)$。
2. 给出树上的一条链 $\text{Path}(u, v)$ 和一点 $x$，查询 $dis(x, \text{Path}(u, v))$ 以及 $x$ 属于链 $\text{Path}(u, v)$ 上的哪一点。
3. 给出树上的两个点 $u, v$ 和一个非负整数 $d \le dis(u, v)$，查询 $\curvearrowright (u, v, d)$。

上述三种操作均为树上问题的经典操作，存在多种不同解法。设 $n = |V|$，下面将给出这三个问题的一种较为简单的 $O(n)$ 预处理，$O(\log n)$ 进行单次查询的办法：

1. 任选一节点作为根节点，对整棵树进行重链剖分。

2. 由重链剖分经典问题的求解方法，容易单次 $O(\log n)$ 查询任意两点的最近公共祖先，以及一个点的 $k$ 级祖先。定义两点 $u,v$ 的最近公共祖先为 $\mathrm{LCA}(u,v)$，一个点 $u$ 的 $k$ 级祖先为 $\Uparrow_k(u)$。

3. 对于操作 1，答案即为 $dep_u + dep_v - 2 \cdot dep_{\mathrm{LCA}(u,v)}$。

4. 对于操作 2，逐个求解 $A = \mathrm{LCA}(u,v), B = \mathrm{LCA}(u,x), C = \mathrm{LCA}(v,x)$，设 $p$ 为 $A,B,C$ 三点中深度最大的节点，则 $x$ 必属于链 $\mathrm{Path}(u,v)$ 上的 $p$ 点。再求 $dis(x,p)$ 即可得到 $dis(x,\mathrm{Path}(u,v))$。

5. 对于操作 3，首先求出 $w = \mathrm{LCA}(u,v)$。如果 $dis(u,w) \geq d$，则 $\curvearrowright(u,v,d) = \Uparrow_d(u)$；否则 $\curvearrowright(u,v,d) = \Uparrow_{dis(u,v)-d}(v)$。

由上一小节中讨论的过程可知，所有的链邻域求交操作都可以在进行常数次上述三种操作来完成。因此，可以做到 $O(n)$ 对树进行预处理，随后在 $O(\log n)$ 的复杂度内完成一次链邻域求交操作。

接下来将给出几道与链邻域求交有关的例题进行分析。

**例题 2** (Ald[2])**.** 给定一棵 $n$ 个顶点构成的树，顶点从 $1 \sim n$ 标号。你需要维护一个由树上路径构成的可重集 $S$，并支持以下几种操作共 $q$ 次：

1. 给定两个顶点 $u,v$，将 $\mathrm{Path}(u,v)$ 加入到可重集 $S$ 中。

2. 给定两个顶点 $u,v$，将一个 $\mathrm{Path}(u,v)$ 从可重集 $S$ 中删除。

3. 给定一个非负整数 $d$，求所有以 $S$ 中路径为中心链的 $d-$ 链邻域的交的大小，即：

$$\left| \bigcap_{\mathrm{Path}(u,v)\in S} M(u,v,d) \right|$$

数据范围：$1 \leq n,q \leq 10^5$。

解法　根据定理 3.2 的结论，$\bigcap_{\mathrm{Path}(u,v)\in S} M(u,v,d)$ 一定可以被表示为链邻域的形式。链邻域数点问题在《浅谈一类树上统计相关问题》[1] 一文中进行了详细介绍。这一部分并不是本文重点，读者可以自行思考或阅读该篇论文。

接下来将讨论如何求出每一次 3 类型询问时交集的链邻域表示，该问题也是本例题中与本文关联最大的部分。

[2]题目来源：Petrozavodsk Summer 2023. Day 7. PKU Contest G. Ald. https://qoj.ac/contest/1376/problem/7507.

根据引理 3.2 的结论，对两个中心链相交的链邻域求交，结果就是半径相等，中心链为原本两条中心链交部分的链邻域。

将此结论推广到任意有限集合中依然成立。即：若 $\bigcap_{\text{Path}(u,v)\in S} \text{Path}(u,v) \neq \varnothing$，那么下式成立：

$$\bigcap_{\text{Path}(u,v)\in S} M(u,v,d) = M(u',v',d). \quad \text{where Path}(u',v') = \bigcap_{\text{Path}(u,v)\in S} \text{Path}(u,v)$$

否则该结论并不成立。但是有如下结论成立：

**引理 3.6.** 对于由链构成的集合 $S$ 满足 $\bigcap_{\text{Path}(u,v)\in S} \text{Path}(u,v) = \varnothing$，对于任意两条 $S$ 中的链 $\text{Path}(u_1,v_1), \text{Path}(u_2,v_2) \in S$ 满足 $dis(\text{Path}(u_1,v_1), \text{Path}(u_2,v_2)) = \max_{p_1,p_2\in S} dis(p_1,p_2)$，则成立下式：

$$\bigcap_{\text{Path}(u,v)\in S} M(u,v,d) = M(u_1,v_1,d) \cap M(u_2,v_2,d)$$

证明. 由于所有中心链的交为空集，因此必然存在两条链的交为空，从而距离最长的两条中心链交为空，即有 $dis(\text{Path}(u_1,v_1), \text{Path}(u_2,v_2)) > 0$。

如果 $M(u_1,v_1,d) \cap M(u_2,v_2,d) = \varnothing$，那么显然等式左右均为空集，等式成立。

否则根据引理 3.3，设 $x \in \text{Path}(u_1,v_1), y \in \text{Path}(u_2,v_2)$ 为让两条链距离取到最小值的 $x,y$，令 $len = dis(x,y)$，$c = \frown(x,y,\frac{len}{2})$，那么 $M(u_1,v_1,d) \cap M(u_2,v_2,d) = N(\frown(x,y,\frac{len}{2}), d-\frac{len}{2})$。

由 $\text{Path}(u_1,v_1)$ 和 $\text{Path}(u_2,v_2)$ 是所有链对中距离最大的可知，对于任意 $\text{Path}(u',v') \in S$，都有 $dis(c,\text{Path}(u',v')) \leq \frac{len}{2}$。设 $c$ 属于链 $\text{Path}(u',v')$ 中的 $t$，则有 $N(\frown(x,y,\frac{len}{2}), d-\frac{len}{2}) \subset N(t,d) \subset M(u',v',d)$。引理得证。 □

利用引理 3.6 的结论，只需要维护出 $S$ 集合中距离最远的两条链即可。

得到上述结论后，使用线段树分治将问题转化为只向 $S$ 集合中添加路径。同时维护以下两个信息：

- 若 $S$ 中所有路径交集不为空，则维护所有路径的交集。
- 否则维护 $S$ 中距离最大的两条链。

不难发现上述信息在新增加一条路径时都可以在 $O(\log n)$ 的复杂度内进行更新。因此本题可以在 $O(n + q\log q\log n)$ 的复杂度内求出每一个 3 询问对应的链邻域表示。 ■

**例题 3** (这里有只毛毛虫[3])**.** 给定一张 $n$ 个点 $n$ 条边构成的无向连通图（即基环树）。

定义基环树上任意两个节点的距离 $dis(u,v)$ 为 $u$ 与 $v$ 之间最短路径的边数。

定义一个该基环树上的一个毛毛虫 Cat(u,v,c) 为：

$$\text{Cat}(u,v,c) = \{x \mid dis(x,u) + dis(x,v) \leq dis(u,v) + c\}$$

你需要维护一个由毛毛虫构成的可重集 $S$，支持以下操作 $q$ 次：

[3]本题为笔者原创题。

- 给定三个整数 $u, v, c$，将 $\mathrm{Cat}(u, v, c)$ 插入集合 $S$，并在插入后输出 $S$ 中所有毛毛虫交集的大小。

数据范围：$1 \le n, q \le 5 \times 10^5$。

解法　若本题的无向图是一棵树，则该问题即为上面所介绍的树上链邻域求交。

考虑求出基环树的环，将所有环上的边切断后，原图变成若干棵树，每棵树都以一个环上的点为根。尝试对于每一棵拆出来的树维护对应树上的交。

考虑一次操作会对每个树产生什么影响。假设操作为 $\mathrm{Cat}(u, v, c)$，则对于 $u$ 和 $v$ 所在的树，影响即为在对应的树上进行一次链邻域求交操作。

而对于非 $u$ 和 $v$ 所在的树，其在 $\mathrm{Cat}(u, v, c)$ 的部分其实是一个根的邻域。具体的，设根为 $t$，则 $t$ 树内与 $\mathrm{Cat}(u, v, c)$ 相交的部分即为 $N(t, c - dis(c, \mathrm{Path}(u, v)))$。

通过这一部分分析可以发现，对于每一棵树，每次操作本质上是进行一次链邻域求教操作（由定理 3.1，邻域求交也可以被表示为链邻域求交）。故每一棵树中在 $S$ 交中的部分即为树上的一个链邻域或空。

接下来尝试快速维护这个过程。对于一棵以 $t$ 为根的树的链邻域交，容易被表示为 $M(u_t, v_t, r_t) \cap N(t, d_t)$，初始时 $d_t$ 为对应树中最深节点的深度。一次操作中，会对 $M(u_t, v_t, r_t)$ 产生影响的只有 $u$ 和 $v$ 所在的树中，直接暴力更新即可。

而对于别的树，只会对 $N(t, d_t)$ 这一部分中的 $d_t$ 产生影响，且具体影响为将 $d_t$ 对一个值取较小值。尝试快速找到所有 $d_t$ 减小的 $t$，并对这样的 $d_t$ 重新求链邻域之交。容易发现这样的更新只会进行 $\sum dep_t = O(n)$ 次。而对 $d_t$ 取较小值的值一定形如常数段公差为 0 或 1 的等差数列，因此可以直接用线段树维护环上每个节点对应树当前的 $d_t$、$d_t + rnk_t$ 和 $d_t - rnk_t$ 即可快速找出一次操作会更新的位置。

综上，本题只需要进行 $O(n + q)$ 次链邻域求交操作。而对于每一次求交操作后，还需要进行一次链邻域数点操作，该操作可以参考《浅谈一类树上统计相关问题》[1] 一文中所给出的做法。至此，本题可以在 $O((n + q) \log n)$ 的时间复杂度内被解决。 ■

## 4 总结

本文从树上邻域出发，首先讨论了树上邻域与点集直径之间的关系，并发现树上真实邻域对求交操作封闭。进一步地，将树上邻域推广到树上链邻域，通过证明若干引理，发现若将空集也视作树上链邻域，则树上链邻域对求交操作也封闭。同时，也给出了实现树上链邻域求交的具体算法实现方法，并给出几道例题加以分析，对此操作的应用进行了初步的展示。

目前本理论与算法在信息学竞赛的具体题目中出现与应用较少。希望本文能起到抛砖引玉的作用，期待读者能够对部分理论进行进一步研究。

## 5 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢杭州第二中学李建老师的关心和指导。
感谢家人、朋友对我的支持与鼓励。
感谢陈昕阳、孙恒喆学长和章绍嘉同学对我的启发。
感谢其他给予我帮助的老师与同学。

## 参考文献

[1] 朱羿恺. (2023). 浅谈一类树上统计相关问题. 2023 年信息学奥林匹克中国国家集训队论文.

[2] 王羽立. (2020). 题解 CF1458F【Range Diameter Sum】. Luogu. https://www.luogu.com.cn/article/rb0wgud8.

[3] 吴思成. (2023). 冬のエピローグ. Luogu. https://www.luogu.com.cn/article/hop4knd8.

[4] 朱羿恺. (2023). Petrozavodsk Summer 2023. Day 7. PKU Contest Tutorials. QOJ. https://qoj.ac/download.php?type=attachments&id=1376&r=1.

# 可撤销模型在一类特定情形下的拓展

中国人民大学附属中学 王梓丞

## 摘 要

信息学竞赛中广泛存在一类支持撤销修改的模型。本文讨论了此种模型在扩展到支持在线删除时的情形，将其概括为一类栈操作问题，并通过暴力重构与势能分析的手段给出了其在一些特殊情形下的解决方法。

## 1 概述

### 1.1 可撤销模型

在信息学竞赛中存在许多模型，其能够在对数据结构进行修改的同时维护对上一次修改的撤销操作，我们称这样的模型为可撤销模型。以下是其中的两个例子。

**例 1.** 可撤销并查集[1]

给定 $n$ 个点的图 $G$，初始边集为空。需要支持如下三种操作：

1. 加入无向边 $(u, v)$。
2. 撤销上一次加边操作。
3. 询问两个点 $u, v$ 是否在同一连通块内。

使用并查集维护连通性。维护所有修改构成的栈，合并时若将某个节点 $x$ 的父亲设为了 $y$，则在栈中加入元素 $(x, y)$；撤销时弹出栈顶，断开二者之间的连边即可。

若合并时采用启发式合并，即每次将 $x$ 连到 $y$ 时保证 $x$ 的集合大小较小，则单次操作复杂度 $O(\log n)$。注意由于路径压缩的复杂度是均摊的，此处不能进行路径压缩优化。 ■

**例 2.** 可撤销 01 背包

给定 $n, m$，有一个初始为空的物品集合 $S$，需要进行 $n$ 次如下操作：

[1] 详细讨论参见 [1]

1. 加入一个价值为 $c$，重量为 $w$ 的物品。
2. 撤销上一个加入的物品。

每次操作后，对每个 $1 \le i \le m$ 输出任意选取重量和不超过 $i$ 的物品可以获得的最大价值和。

考虑维护物品构成的栈 $S$，并计算 $f_{i,j}$ 表示考虑栈底到栈顶前 $i$ 个物品，且背包容量不超过 $j$ 时的最大价值。加入物品时可由 $f_{|S|}$ 数组 $O(m)$ 计算出 $f_{|S|+1}$ 的值；撤销则只需弹出栈顶并丢弃栈顶的 $f$ 数组，复杂度 $O(nm)$。 ■

另外，所有的持久化数据结构实际上也能够通过版本回溯支持撤销。

## 1.2 扩展到带删除时的情形

如上所述的可撤销模型仅能撤销上一次修改操作，而在许多情况下我们维护的结构需要进一步支持删除此前进行过的某一次修改操作。对于一些可撤销模型，其由于自身数据结构或修改的特殊性（如修改可减），能够以与撤销相同的方法维护删除；但大多数可撤销模型，比如上一部分所述的两种模型，并不能直接维护删除。我们希望扩展一般的可撤销模型，使得其能够维护任意删除的操作。

### 1.2.1 离线情况

若操作允许离线，则上述问题容易在 $O(nC \log n)$ 的时间复杂度下做到任意删除一次修改，其中 $n$ 为操作次数，$C$ 为可撤销模型单次修改/撤销的复杂度。

**问题描述** 给定一数据结构与某种修改，要求支持 $n$ 次如下操作：

1. 修改操作：对该数据结构进行某种修改。
2. 撤销操作：撤销第 $i$ 次修改操作。

此处假设存在一个可撤销模型，能够维护该数据结构的给定修改与对上一次修改的撤销。

**解法** 首先对操作时间轴建立线段树，离线后处理出每个修改的出现时间区间，然后将该次修改加入到对应的 $O(\log n)$ 个线段树结点内。

处理询问时在线段树上 dfs，每次进入一个线段树结点时处理其上的所有修改，离开该区间时撤销所有修改。则 dfs 到叶子 $[l, l]$ 时数据结构中恰维护了该叶子到根的所有线段树结点里的修改，也即时刻 $l$ 时存在的所有修改。

上述做法中每个操作被加入到了 $O(\log n)$ 个线段树结点中，因此总的修改/撤销次数为 $O(n \log n)$ 次。 ■

### 1.2.2 在线情况

在线的情形下，我们必须实时维护所有进行过的修改，并对修改构成的集合进行更新。

注意到所有修改一定是后进先出的，即每次只能撤销最后一次进行的修改。因而修改集合的更新方法类似一个栈：一次操作可以压入一个修改，或从栈顶弹出最晚压入的修改。为了进一步刻画能处理的操作，以下引入操作栈的概念：

**定义 1** (操作栈)**.** 对一个可撤销模型，记任意时刻其操作栈 $S$ 为一个包含了所有已进行的修改的栈，其中所有修改的进行顺序自栈底到栈顶递增。

此时对于任意可撤销模型，前述的带删除问题都可以归约到为如下的栈操作问题。

**栈操作问题** 给定一个初始为空的栈 $S$，要求通过栈的 push，pop 操作在线地维护如下操作：

1. 向栈中加入一个元素 $x$。
2. 从栈中弹出某个指定的元素 $x$。

不幸的是，对于一般的情况，上述问题并不存在一个复杂度较优的做法。

**定理 1.** 栈操作问题的最坏入栈出栈次数为 $O(n^2)$，其中 $n$ 为操作总次数。

证明. 构造如下的操作序列：首先进行 $\frac{n}{2}$ 次加入操作；然后进行 $\frac{n}{2}$ 次删除操作，每次指定弹出栈底的元素。此时第 $i$ 次删除至少需要出栈 $\frac{n}{2}-i+1$ 次，总出栈次数为 $\sum_{i=1}^{\frac{n}{2}}(\frac{n}{2}-i+1)=O(n^2)$。 □

因此对于一般的带删除问题，此种方式并不能做到比暴力撤销更好。但在一些删除有特殊限制的问题下，上述问题的复杂度的确能够得到优化。在下文的讨论中，我们将给带删除问题的删除操作加以不同的特殊限制，并试图在其上设计出更为高效的算法。

## 2 单栈模拟队列问题

### 2.1 问题引入

考察如下带删除问题：

**问题描述** 给定一个初始为空的栈 $S$，要求仅通过 $S$ 的 push 和 pop 操作，在线地维护如下修改：

1. 向栈中加入一个元素 $x$。
2. 从栈中弹出加入时刻最早的元素，保证栈不为空。

注意到所有元素实际上满足先进先出的性质，因此可以认为我们用栈等效地实现了一个队列。即，两种操作分别对应于队列的 push 和 pop 操作。我们称该问题为**单栈模拟队列问题**。

## 2.2 双栈模拟队列问题

首先回顾上述问题的一个更广为人知的弱化版本。

**问题描述** 维护由两个初始为空的栈 $S_0, S_1$ 构成的数据结构，要求仅通过对两个栈的 push、pop 操作，在线地维护如下修改：

1. 向数据结构中加入元素 $x$。
2. 删除加入时刻最早的元素，保证该数据结构不为空。

上述问题要求使用两个栈等效地实现一个队列，因此该问题称为双栈模拟队列问题。

**解法** 首先给两个栈中的所有元素定序。具体来说，保证在操作的任意时刻：

1. $S_0$ 中所有元素的加入时刻自栈顶到栈底递增，$S_1$ 中所有元素的加入时刻自栈底到栈顶递增。
2. $S_0$ 中所有元素的加入时刻均早于 $S_1$ 中所有元素的加入时刻。

则此时最早和最晚加入的元素分别为 $S_0$ 和 $S_1$ 的栈顶。因而对于加入元素 $x$ 的操作，可直接将 $x$ 压入 $S_1$ 的栈顶；删除操作则可将 $S_0$ 的栈顶弹出，但此时 $S_0$ 可能为空。此时可暴力重构两个栈。假设某次删除操作前 $S_0$ 为空，而 $|S_1| = t$，则可将 $S_1$ 的所有元素取出，并全部放入 $S_0$，然后再取出 $S_0$ 的栈顶。

### 2.2.1 复杂度分析

为了分析该解法的复杂度，以下引入势能分析的概念。[2]

**定义 2** (势函数)**.** 对一初始数据结构 $D_0$ 及其依次进行 $n$ 个操作的过程中得到的数据结构 $D_1 \sim D_n$，定义势函数 $\Phi$ 为将数据结构 $D_i$ 映射到一个实数 $\Phi(D_i)$ 的函数。

为书写方便，在不引起歧义的情况下，以下将用 $\Phi_i$ 指代 $\Phi(D_i)$。

**定义 3** (摊还代价)**.** 对一初始数据结构 $D_0$ 依次进行的 $n$ 个操作，记第 $i$ 个操作的实际代价为 $c_i$，则第 $i$ 个操作的摊还代价 $\hat{c}_i$ 用势函数 $\Phi$ 定义为：

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

[2]参考自 [2]

称 $n$ 次操作的总摊还代价为 $\sum_{i=1}^{n}\hat{c}_i=\sum_{i=1}^{n}(c_i+\Phi_i-\Phi_{i-1})=\sum_{i=1}^{n}c_i+\Phi_n-\Phi_0$。通常来说，势函数的定义满足 $\Phi_0=0$ 及 $\forall 1\le i\le n,\Phi_i\ge 0$。此时 $\Phi_n-\Phi_0\ge 0$，总摊还代价即为总实际代价 $\sum_{i=1}^{n}c_i$ 的一个上界。

回到本题。以下假设栈的单次 push、pop 操作的代价均为 1。显然对栈的 pop 操作次数一定不会多于 push 操作的次数，因此下文仅分析 push 操作的代价，则实际复杂度与该结果至多差常数倍。

定义势能函数 $\Phi=|S_1|$，其中 $|S|$ 代表栈 $S$ 中的元素个数。对于加入操作和 $S_0$ 不为空时的删除操作，$|S_1|$ 至多变化 1，因而 $\hat{c}_i\le 2$。

对于 $S_0$ 为空的删除操作，记此时 $|S_1|=t$，则操作后 $|S_1|=0$，且一共 push $t$ 次，因此其摊还代价为：

$$\begin{aligned}\hat{c}_i&=c_i+\Phi_i-\Phi_{i-1}\\&=t+0-t\\&=0\end{aligned}$$

故单次操作摊还代价为 $O(1)$。显然 $\Phi_i\ge 0$，因此总时间复杂度不超过 $\sum\hat{c}_i=O(n)$。 □

## 2.3 单栈模拟队列问题的解法

回到原问题，考虑沿用双栈模拟队列的思想。我们将栈 $S$ 中的所有元素分成两类：push 元素和 pop 元素，简记为 1 元素和 0 元素，同时保证所有元素的加入时刻满足 0 元素早于 1 元素、靠近栈顶的 0 元素早于靠近栈底的 0 元素、靠近栈顶的 1 元素晚于靠近栈底的 1 元素。不难发现该模型的本质即为将前述的 $S_0,S_1$ 两个栈放到一个栈中维护。

对于加入操作，此时仅需向 $S$ 中 push 一个元素 $x$，并将其标记为 1 元素。删除操作则要求弹出最靠近栈顶的 0 元素，但暴力弹出第一个 0 元素及其上的所有 1 元素的复杂度无法保证。

考虑在查找并弹出 0 元素的过程中进行一定范围内的重构以保证复杂度。删除时若栈顶不为 0 元素，则不断弹出栈顶，直到弹出的 0 元素个数与 1 元素个数相等；然后将所有弹出的元素放回去，保证所有 0 元素比所有 1 元素更靠近栈顶。特别地，若弹出所有元素后仍无法满足前述限制，则将所有元素变为 0 元素，并按加入时刻从晚到早依次放回 $S$ 中。此时 $S$ 的栈顶为 0 元素，直接弹出即可。

## 2.4 复杂度分析

以下若无特殊说明，默认 log 函数的底数为 2。

不妨记序列 $s_1,s_2,\cdots,s_m$ 表示操作前栈 $S$ 中自栈顶向下第 $i$ 个元素的 01 类型，其中 $m=|S|$。定义势能 $\Phi=2(\sum_i[s_i=0]\log i)+4\log n(\sum_i[s_i=1])$。

加入单个 1 元素的摊还代价:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + 2(\sum_{i=1}^{m}[s_i = 0](\log(i+1) - \log i)) + 4\log n \\ &\le 1 + 2\log n + 4\log n = 6\log n + 1\end{aligned}$$

删除操作分为两部分：栈顶重构和弹出栈顶的 0 元素。

**栈顶重构** 此处需分两种情形讨论。若存在 $s$ 的一段前缀满足 01 个数相等，上述算法将取出满足条件的最短前缀，不妨假设其中有 $t$ 个 0。由于 $s_1 = 1$，根据假设，任意一段 $< 2t$ 的前缀中 1 必然比 0 多。因此对于从前往后第 $i$ 个 0（$1 \le i < t$），其初始位置至少为 $2i+1$。则：

$$\begin{aligned}\hat{c}_i &= 2t + 2(\sum_{i=1}^{t}\log i) - 2(\sum_{i=1}^{2t}[s_i = 0]\log i) \\ &\le 2t + 2(\sum_{i=1}^{t}\log i) - 2(\log 2t + \sum_{i=1}^{t-1}\log(2i+1)) \\ &\le 2t - 2t = 0\end{aligned}$$

若 $s$ 的任意前缀中 1 均比 0 多，则全栈的 1 个数至少为 $\lceil \frac{m}{2} \rceil$。此时有：

$$\begin{aligned}\hat{c}_i &\le m + 2(\sum_{i=1}^{m}\log i) - 4\lceil\frac{m}{2}\rceil\log n \\ &\le m + 2(\sum_{i=1}^{m}\log i) - 2m\log m\end{aligned}$$

为了分析上述不等式，提出如下引理：

**引理 1.** 对任意正整数 $n$，有 $n\log n - \sum_{i=1}^{n}\log i \ge n - \log n - 1$。

证明. 归纳。记 $f(n) = n\log n - \sum_{i=1}^{n}\log i$，则首先 $n = 1$ 时上式成立；$n > 1$ 时：

$$\begin{aligned} f(n) &= \lfloor\frac{n}{2}\rfloor \log n - (\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor} \log i) + \sum_{i=\lfloor\frac{n}{2}\rfloor+1}^{n} (\log n - \log i) \\ &\geq f(\lfloor\frac{n}{2}\rfloor) + \lfloor\frac{n}{2}\rfloor \\ &\geq 2\lfloor\frac{n}{2}\rfloor - \log\lfloor\frac{n}{2}\rfloor - 1 \\ &\geq n - 1 - \log\lfloor\frac{n}{2}\rfloor - 1 \geq n - \log n - 1 \end{aligned}$$

□

回到原式，则 $\hat{c_i} \leq m - 2(m - \log m - 1) \leq 2\log m + 2 \leq 2\log n + 2$。

**弹出栈顶** 此时 $\hat{c_i} = 0 + 2\sum_{i=2}^{m}[s_i = 0](\log(i-1) - \log i) \leq 0$。

综上所述，每次操作的摊还代价均不超过 $O(\log n)$，而显然 $\Phi_i \geq 0$，因此总复杂度不超过 $\sum \hat{c_i} = O(n \log n)$。 □

## 2.5 例题

**例 3.** 地铁规划[3]

给定 $n$ 个点的图 $G$ 与其上的 $m$ 条无向边 $(u_i, v_i)$。你需要使用交互库给定的可撤销并查集接口，在线地对每个 $1 \leq l \leq m$ 求出最大的 $r$，满足只保留编号在 $[l, r]$ 中的所有边后原图无环。另外，你使用并查集进行加边操作时不能加边超过 $lim$ 次。

$2 \leq n \leq 10^5$，$2 \leq m \leq 2 \times 10^5$，$lim \geq 5 \times 10^6$。

**解法** 不妨记 $f_{l,r} = 0/1$ 表示编号在 $[l, r]$ 中的所有边构成的子图是否无环。注意到 $f_{l,r} = 1 \Rightarrow f_{l+1,r} = 1$，因此对每个左端点 $i$ 的最大右端点值 $r_i$，有 $r_i \leq r_{i+1}$。由此可考虑双指针，每次 $i \to i + 1$ 时删除编号为 $i$ 的边，然后不断右移 $r$ 直到成环。则加边和删边的总次数均不超过 $m$。此处并查集只支持撤销上一次加边，而每次需要删除的边 $i$ 恰为区间中最早加入的边，直接套用上述单栈模拟队列做法，将 push 和 pop 分别改为加边和 undo，则总交互次数 $O(m \log m)$。值得注意的是，该做法在实际情况下的常数表现远优于前述分析给出的上界，能够直接在本题约 $1.5m \log m$ 的交互次数限制下通过。 ■

[3]UOJ 693，https://uoj.ac/problem/693

# 3 单栈模拟优先队列

**问题描述** 给定一个初始为空的栈 $S$，要求仅通过 $S$ 的 push 和 pop 操作，在线地维护如下修改：

1. 向栈中加入一个元素 $x$，权重为 $p$。保证 $p$ 互不相同。
2. 从栈中弹出 $p$ 最小的元素，保证栈不为空。

显然该数据结构等效于一个优先队列，因此称作**单栈模拟优先队列问题**。

## 3.1 解法

拓展前述 01 元素的想法。将栈中的所有元素标记为 0 或 1，保证任意时刻满足如下条件：

1. 所有 0 元素的权重自栈顶向栈底递增。
2. 任意一个 1 元素的权重均大于所有比其更靠近栈顶的 0 元素的权重。

实际上，单栈模拟队列中对 01 元素的限制即为上述限制的一个特例情况。

由此可以得到，任意时刻栈中最小权重的元素一定位于最靠近栈顶的 0 元素与其前面的所有 1 元素构成的集合中。考虑直接套用单栈模拟队列的做法：加入时 push 一个 1 元素，删除时进行栈顶重构，此时若共取出了 $n$ 个 0 元素与 $n$ 个 1 元素，则需将这些元素按 $p$ 从大到小插入回 $S$，并将前 $n$ 个元素重标为 0，后 $n$ 个元素标为 1；否则若重构了栈中的所有元素，则需要将所有元素重标为 0。

容易说明上述栈顶重构统计到了所有可能成为最小值的元素，因此重构后的栈顶 0 元素必然为最小值。另外，重构前第 $i$ 个 0 元素至少为前 $2n$ 个元素中的权重第 $i$ 小，因此重构后第 $i$ 个 0 对应的元素权值必然不会增大，故前述条件仍能得到满足。

该做法的入栈次数同单栈模拟队列的做法，为 $O(n \log n)$。然而重构栈顶前 $n$ 个数时需要对其按权重排序，由于重构的总个数为 $O(n \log n)$，此时最坏可能产生 $O(n \log^2 n)$ 的额外时间复杂度。

考虑仔细地分析排序的过程。首先对于所有 0 元素，其顺序已经确定，故只需将所有 1 元素排序，最后与 0 元素归并即可；归并的总复杂度同入栈次数，为 $O(n \log n)$。

对于 1 元素，注意到一次重构后会产生一个已按权重递增序排好的 1 元素连续段，这启发我们按照一些递增段的方式维护所有 1 元素。具体来说，定义 $S$ 中的一个 1 元素段为一段位置连续的 1 元素，满足其权重自栈顶方向向栈底方向递增。对于这些 1 元素段，可以直接通过归并的方式进行排序。

我们对前述算法做如下修改：加入一个 1 元素时令其自成一段；重构时取出所有涉及到的 1 元素段，进行某种形式的归并，并最终合并为单个 1 元素段。注意到若重构为全栈

重构，显然不会切开一个 1 元素段；不为全栈重构，则弹出的最后一个元素一定为 0 元素，故所有取出的 1 元素段都是完整的。

对于归并过程：记取出的 1 元素段的元素个数分别为 $a_1, \cdots, a_p$，则可将所有段按照哈夫曼树[4]的方式合并。即，每次取出长度最小的两个段 $i, j$，花费 $a_i + a_j$ 的代价将其归并为长为 $a_i + a_j$ 的 1 元素段，直至合并成单个段。最后将该段与所有 0 元素进行归并即可。

## 3.2 复杂度分析

对该部分定义势能 $\Phi' = 4(\log n + 2)(\sum_i [s_i = 1]) - 4\sum a \log a$，其中 $a$ 为每个 1 连续段的长度（元素个数）。

**加入操作** 若增加了一个长为 1 的段，则其对 $\sum a \log a$ 没有贡献。因此有 $\hat{c}_i = 4 \log n + 8$。

**栈顶重构** 分成两部分考虑：对所有 1 元素段排序；若为全栈重构，将所有 1 元素变为 0 元素。

对于排序的过程，记取出的段长为 $a_1 \sim a_p$。对每个段计算其对实际代价的贡献，则其贡献系数为该段在哈夫曼树上的深度 $h_i$（记根节点深度为 0），总实际代价即为 $\sum a_i h_i$。为了分析 $h_i$ 的上界，首先考虑对哈夫曼树的如下引理：

**引理 2.** 哈夫曼树中，记 $s_u$ 为 $u$ 子树内所有叶子的段长之和，即段 $u$ 的段长。若一个节点 $u$ 的深度至少为 2，则 $u$ 的二级祖先 $f$ 满足 $s_f \geq 2s_u$。

证明. 记 $u$ 的兄弟为 $v$，父亲为 $fa_u$，$fa_u$ 的兄弟为 $x$，考虑证明 $s_u \leq s_x$。若 $s_u > s_x$，根据哈夫曼树的合并规则，合并 $u, v$ 时 $s_u, s_v$ 必然为所有 $s$ 值中的前两小；但此时结点 $x$ 仍未出现/未被合并，则必然存在 $x$ 子树内一个点 $y$ 可供选择。此时 $s_y \leq s_x < \max(s_u, s_v)$，与 $s_u, s_v$ 为前两小矛盾。

因此 $s_u \leq s_x = s_f - s_{fa_u} \leq s_f - s_u$，则原命题得证。 □

根据上述引理，可得出如下推论：

**定理 2.** 对长为 $a_i$ 的段（哈夫曼树上的叶子），该叶子在哈夫曼树上的深度 $h_i$ 满足 $h_i \leq 2(\log A - \log a_i) + 1$，其中 $A = \sum_{i=1}^{p} a_i$。

证明. 反证法。对叶子 $i$，其初始 $s_i$ 值恰为 $a_i$；若结点 $u$ 深度至少为 2，则其向上跳两步后 $s$ 值翻倍。由归纳法，$i$ 的 $2k$ 级祖先 $f$ 必然满足 $s_f \geq 2^k a_i$。若 $i$ 存在 $2(\log A - \log a_i + 1)$ 级祖先 $f$，则 $s_f \geq 2^{\log A - \log a_i + 1} a_i > A$，而 $s$ 显然有上界 $A$，矛盾。因此原命题得证。 □

回到复杂度分析部分，则实际代价 $c = \sum a_i h_i \leq \sum a_i (2 \log A - 2 \log a_i + 1) = 2A \log A - 2\sum a_i \log a_i + \sum a_i$。另外在实现时，需要每次取出大小前两小的结点，该过程可以从小到大枚举结点 $s$ 值大小解决，因此单次合并代价变为 $2(s_u + s_v)$，则实际代价需再乘以 2。此时有：

[4] 参见 [3]

$$\begin{aligned}\hat{c}_i &= c_i + \Phi'_i - \Phi'_{i-1} \\ &\le 2(2A\log A - 2\sum a_i \log a_i + \sum a_i) - 4A\log A + 4\sum a_i \log a_i \\ &= 2\sum a_i\end{aligned}$$

不超过重构过程中的入栈次数的两倍。

全栈重构时涉及到将一个 1 元素段中的 1 元素变为 0 元素的操作。考虑单个这样的操作对势能的影响，不妨记该 1 元素所属的段长为 $x$。此时有 $\hat{c}_i = 1 + 4(x\log x - (x-1)\log(x-1)) - 4(\log n + 2)$。

考虑分析 $x\log x - (x-1)\log(x-1)$ 在 $x > 1$ 时的范围。由于 $x\log x$ 的导函数为单调增的函数 $\log x + 1$，因此 $x\log x - (x-1)\log(x-1) = \int_{x-1}^{x}(\log y + 1)\mathrm{d}y \le \log x + 1$。则前述 $\hat{c}_i \le 1 + 4(\log x + 1 - \log n - 2) \le 0$。

则单次修改的代价可忽略，因而所有操作的代价也可忽略。因此关于重构元素数线性对数的项均被均摊掉，复杂度可分析至 $O(n\log n)$。

更严谨地说，由于上述分析中任意操作的代价最多导致单次 push 的代价由 1 变为 3，取总势能为 $3\Phi + \Phi'$，即可分析至任意操作摊还代价 $O(\log n)$。 □

### 3.3 离线下任意删除问题的另解

回顾 1.2.1 中的问题，其可以较为直接地用上述单栈模拟优先队列的思想解决。考虑离线后预处理出每个修改操作的撤销时间，并将其作为每个修改操作的权重，然后进行上述单栈模拟优先队列问题。显然每次弹出的最小值即为指定删除的元素，时间复杂度 $O(n\log n)$。

## 4 单栈模拟多栈问题

**问题描述** 给定一个初始为空的栈 $S$ 与常数 $k$，要求仅通过 $S$ 的 push 和 pop 操作，在线地维护如下修改：

1. 向栈中加入一个元素 $x$，其带有 $1 \sim k$ 之一的颜色。
2. 给定 $i$，满足 $1 \le i \le k$，要求删除上一个加入的颜色为 $i$ 的元素，保证栈中存在一个颜色为 $i$ 的元素。

对每种颜色的所有元素，其加入与弹出的顺序满足后进先出的性质；而 $k$ 种颜色之间互不影响，因此上述结构不弱于 $k$ 个栈的并。我们称该问题为**单栈模拟 $k$ 栈问题**。

## 4.1 解法

首先有一个十分自然的想法：保证在操作的过程中，对于每种颜色 $i$，所有颜色为 $i$ 的元素在栈中满足加入时刻越晚的元素越靠近栈顶。

我们试图继续沿用单栈模拟队列的基本思想：给每个元素分配 01 标记、删除时进行一定范围内的栈顶重构。但此时重构后的最优形态并不显然。直觉上来说，若记栈顶到栈底第 $i$ 个元素的颜色为 $v_i$，则将 $v$ 的一个前缀调整成 $1,2,\cdots,k,1,\cdots,k,\cdots$ 可能最优。

考虑刻画上述调整的形态。以下给出 0 元素段的概念：定义栈中的一段连续元素为 0 元素段，当且仅当所有元素均标记为 0，且其颜色互不相同。此时易知任意 0 元素段的长度均不超过 $k$。

我们对所有 0 元素维护一些 0 元素段，使得每个 0 元素恰包含在一个 0 元素段内；对于 1 元素则没有类似的结构。另外，我们还要求弹出一个 0 元素段内的元素时，必须同时弹出所有该段内的元素。在此种限制下，一个 0 元素段内所有元素在栈 $S$ 中的相对顺序对操作不再造成影响，因此可以忽略之，而将一个 0 元素段视为其内所有元素构成的集合。

最后，记从栈顶到栈底第 $i$ 个 0 元素段中所有元素颜色的集合为 $T_i$，则限制维护过程中所有段满足 $T_{i+1}\subseteq T_i$；同时对于一个颜色为 $v$ 的 1 元素，考察所有比它更靠近栈顶的 0 元素段 $T_1\sim T_i$，我们要求 $v\in T_i$。此时类似单栈模拟优先队列的情形，每种颜色最靠近栈顶的元素必然位于第一个 0 元素段中，或比其更靠近栈顶的所有 1 元素中。

接下来在上述结构的基础上处理每个操作。对于加入操作，直接压入一个 1 元素即可；对于删除操作，若栈顶不为 0 元素，则进行栈顶重构：不断弹出栈顶并统计 0 元素与 1 元素的个数，直到二者相等。注意此时可能弹出了一个 0 元素段中的某些元素，根据前述规则，剩下的元素需要一并弹出。重构的过程如下：

1. 对于每种颜色，整理出被弹出的所有元素的顺序。
2. 假设栈顶重构共弹出了 $p$ 个 0 元素段。依次进行如下过程 $p$ 次：遍历每种颜色 $i$ 并维护集合 $R$，从弹出的所有颜色为 $i$ 的元素中取出加入时刻最晚的，放进 $R$ 中；若不存在则跳过。此后将 $R$ 中的所有元素标为 0，构造出共 $p$ 个 0 元素段 $R_1\sim R_p$。
3. 遍历每种颜色 $i$，若还剩下未被加入的颜色为 $i$ 的元素，将其标为 1 并按顺序压回 $S$ 中。然后将所有构造出的 0 元素段按 $p\to 1$ 的顺序压回栈中。

注意此过程可能使 0 元素的数量增加，但不会改变 0 元素段的数量。

此时栈顶成为了 0 元素。可以证明，此时栈顶的 0 元素段包含了栈中出现过的每种颜色的最晚被加入的元素。为了保持前述条件，考虑将整个元素段弹出，然后删除某个元素。此后将未被删除的元素标记为 1，依次压回 $S$ 中即可。

## 4.2 正确性证明

需要证明上述重构的过程能够保证前述的如下条件：

1. 所有段满足 $T_{i+1} \subseteq T_i$。
2. 对于一个颜色为 $v$ 的 1 元素，所有比它更靠近栈顶的 0 元素段对应的集合 $T_1 \sim T_i$ 均满足 $v \in T$。

证明. 记重构时取出的 $p$ 个 0 元素段的颜色集合依次为 $T_1 \sim T_p$，重构后的颜色集合为 $T'_1 \sim T'_p$。

**引理 3.** $T_i \subseteq T'_i$。

证明. 对于 $v \in T_i$，由于 $T_i \subseteq T_{i-1} \subseteq \cdots \subseteq T_1$，因此弹出的所有元素中至少有 $i$ 个 $v$。根据前述过程，构造 $R_1 \sim R_i$ 的时候一定均取了一个颜色为 $v$ 的元素，因此 $v \in T'_i$。 □

回到原证明。对第一个条件，由构造过程，构造出的 $p$ 个 0 元素段间必然满足该条件；而 $T_{p+1} \subseteq T_p \subseteq T'_p$，因此该条件成立。对第二个条件，由构造过程，若一个元素成为了 1 元素，则构造出的所有 0 元素段中必然包含了与其颜色相同的元素；对于未被重构的颜色为 $v$ 的 1 元素，有 $v \in T_p, T_p \subseteq T'_p \subseteq \cdots \subseteq T'_1$，故 $v \in T'_1 \sim T'_p$，命题得证。 □

## 4.3 复杂度证明

仍然定义势能 $\Phi = 2(\sum_i [s_i = 0] \log i) + 4 \log n (\sum_i [s_i = 1])$，其中 $s_i$ 为自栈顶向栈底第 $i$ 个元素的标记。

**加入操作** 此处的分析与单栈模拟队列中对加入操作代价的分析相同，即 $\hat{c}_i \le 6 \log n + 1$。

**栈顶重构** 若重构没有涉及到所有栈中的元素，不妨假设共弹出了 $t_1$ 个 1 元素与 $t_0$ 个 0 元素。由于单个 0 元素段的大小不超过 $k$，上述过程在弹出 $t_1$ 个 0 元素后至多又额外弹出了 $k$ 个元素，因此弹出的总元素个数不超过 $2t_1 + k$。

将重构的过程近似地看作两部分：所有原本为 0 的元素移动到栈顶的过程，以及 $x$ 个 1 元素变为 0 元素的过程。由前述的分析可知，$\forall 1 \le i < t_1$，第 $i$ 个 0 元素的位置必然 $\ge 2i + 1$，因此对该部分有：

$$\begin{aligned}
\Phi_i - \Phi_{i-1} &= 2\left(\sum_{i=1}^{t_0} \log i - \sum_{i=1}^{t_0+t_1} [s_i = 0] \log i\right) \\
&\le 2\left(\sum_{i=1}^{t_0} \log i - \sum_{i=1}^{t_1-1} \log(2i+1) - \sum_{i=0}^{t_0-t_1} \log(2t_1 + i)\right) \\
&\le 2\left(\sum_{i=1}^{t_1} (\log i - \log 2i) - \sum_{i=1}^{t_0-t_1} (\log(2t_1 + i) - \log i)\right) \\
&\le -2t_1
\end{aligned}$$

对若干个 1 元素变为 0 元素的过程，每个 1 元素原本对势能贡献 $2\log n$，变化后至多贡献 $\log(t_0 + t_1)$，因此该部分 $\Phi_i - \Phi_{i-1} \le 0$。该次操作摊还代价为：

$$\begin{aligned}\hat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\ &\le 2t_1 + k - 2t_1 + 0 \\ &\le k\end{aligned}$$

若栈 $S$ 中的所有元素均被重构，则 1 元素数量必然过半，且重构后所有元素都将变为 0 元素。这一部分分析同 2.4 节对应部分的分析，有 $\hat{c_i} \le 2\log n + 2$。

**弹出栈顶** 根据前述做法，此部分等价于弹出栈顶的 $t$ 个 0 元素，并加入 $t-1$ 个 1 元素，满足 $t \le k$。则有：

$$\begin{aligned}\hat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\ &= (t-1) + 4(t-1)\log n - 2(\sum_{i=1}^{t} \log i) - \sum_{i=t+1}^{m} [s_i = 0](\log(i-1) - \log i) \\ &\le (t-1)(4\log n + 1) \le 4k\log n + k\end{aligned}$$

综上所述，单次操作的摊还代价 $\hat{c_i}$ 均不超过 $O(k\log n)$。总操作次数上界即为 $\sum_{i=1}^{n} \hat{c_i} = O(nk\log n)$。

另外还需讨论栈顶重构部分的额外时间复杂度。若在前述重构中使用链表维护哪些颜色仍有元素未被取空，则构造每个 0 元素段的复杂度均为其大小，该部分总复杂度与重构元素个数相同。对每次重构建立链表有 $O(k)$ 的额外时间复杂度，而该部分的总影响为 $O(nk)$，不是复杂度瓶颈。 □

## 4.4 例题

**例 4.** *The Profiteer*[5]

给定 $n$ 个物品，第 $i$ 个物品有价值 $v_i$，初始价格 $a_i$ 和新价格 $b_i$。记 $f(V)$ 表示取出价格和不超过 $V$ 的物品，可以获得的最大价值和，其中每个物品只能取一次。求出有多少个区间 $[l, r]$，满足将编号在 $[l, r]$ 中的物品的价格由 $a_i$ 改为 $b_i$ 后，有 $\frac{\sum_{i=1}^{k} f(i)}{k} \le E$，其中 $k, E$ 为给定的常数。

$1 \le n, k \le 2 \times 10^5$，$n \times k \le 10^7$，$1 \le E \le 10^9$，对每个物品有 $1 \le a_i < b_i \le k$，$1 \le v_i \le 10^4$。

[5]QOJ 3998，https://qoj.ac/problem/3998

**解法** 这里仅讨论与本文相关的一种解法。记 $F_{l,r}$ 表示若将 $[l,r]$ 中物品的价格由 $a$ 改为 $b$，所求 $\frac{\sum_{i=1}^{k} f(i)}{k}$ 是否不大于 $E$。由于 $a_i < b_i$，改变后物品的价格变大，每个 $f(i)$ 必然不降，因此有 $F_{l,r} = 1 \Rightarrow F_{l-1,r}, F_{l,r+1} = 1$。则有如下两条推论：

1. 对每个 $l$，满足条件的 $r$ 是一段后缀。
2. 对每个 $i$，记 $R_i$ 为满足 $F_{i,r} = 1$ 的最小 $r$ 值，则 $R_i \le R_{i+1}$。

即该问题仍然可以双指针，则可以维护当前 $i$ 对应的 $R$，每次 $i \to i+1$ 时删去物品 $(b_i, v_i)$，加入物品 $(a_i, v_i)$；$R \to R+1$ 时删去物品 $(a_{R+1}, v_{R+1})$，加入物品 $(b_{R+1}, v_{R+1})$。过程中需要维护所有当前物品做 01 背包的结果，但 01 背包无法直接带删除。

考虑任何时刻所有价格取 $b$ 的物品构成一段区间，价格取 $a$ 的物品构成一个前缀与一个后缀。关于中间区间部分的修改为加入右端点、删除左端点，是一个队列的形式；而关于两侧是两个栈的形式。使用双栈模拟队列可以将前者改为 $O(n)$ 次对两个栈的入栈出栈操作，则原问题变为维护总计 $O(n)$ 次对四个栈的操作，套用单栈模拟 $k$ 栈的做法可做到 $O(n \log n)$ 次加物品/撤销操作。每次操作是 $O(k)$ 的，总时间复杂度 $O(nk \log n)$。 ■

## 4.5 拓展：单栈模拟 $k$ 优先队列问题

**题目大意** 给定初始为空的栈 $S$，在线地维护如下修改：

1. 向栈中加入元素 $x$，其包含指定的颜色 $1 \le i \le k$ 与权重 $p$。保证所有元素的 $p$ 值互不相同。
2. 给定 $1 \le i \le k$，要求弹出栈中颜色为 $i$ 的最小权重的元素，保证此时至少存在一个颜色为 $i$ 的元素。

**解法** 维护的每种颜色的部分实际上都等效于一个优先队列，因此朴素的做法是套用单栈模拟 $k$ 栈，其中每个栈再模拟一个优先队列。但若直接这样做，第 $i$ 个栈的入栈出栈次数为 $O(n_i \log n_i)$，其中 $n_i$ 为涉及到颜色 $i$ 的操作个数。记总操作次数为 $N = \sum n_i$，则总复杂度为 $O((\sum n_i \log n_i) k \log(\sum n_i \log n_i))$，最坏情况下可到 $O(Nk \log^2 N)$。

注意到单栈模拟 $k$ 栈与模拟优先队列的做法实质上有某种共通性。实际上，单栈模拟 $k$ 栈过程中每一种颜色的所有元素也可由一个自顶到底的 01 标记序列刻画。

因此有如下想法：在维护单栈模拟 $k$ 栈的基本结构时，对每种颜色的所有元素额外保证其关于权重满足 3.1 节中的性质。即，所有 0 元素的权重自栈顶向栈底递增；任意一个 1 元素的权重均大于所有比其更靠近栈顶的 0 元素的权重。

为了做到这一点，只需在重构过程中保证每种颜色的所有元素均被排序。具体而言，对每种颜色整理被取出的所有元素的顺序时，改为按照权重排序即可。使用前述哈夫曼树归并的方法可做到 $O(nk \log n)$，正确性与复杂度证明从略。 ■

## 5 总结

本文讨论了可撤销模型及维护操作栈的思想，通过暴力重构与势能分析，设计了若干栈模拟特定数据结构的问题及解法，得到了一些特殊的带删除情况下复杂度优秀的算法。希望本文上述的一些思路可以对 OI 中的相关问题产生更多的启发。

## 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢人大附中叶金毅老师、梁霄老师对我的关心和指导。
感谢家人、朋友对我的支持与鼓励。
感谢邵敏楷、王照元同学与我讨论以及给予的启发。
感谢所有为本文勘误的学长和同学们。

## 参考文献

1. 王相文. 浅谈并查集在 OI 中的特殊应用, 2023
2. OI-wiki 贡献者. 均摊复杂度—OI-wiki, 2022.
3. OI-wiki 贡献者. Huffman Tree —OI-wiki, 2022.

# 集合幂级数中的稀疏多项式乘法

北京市十一学校 武林

## 摘 要

集合幂级数近年来在许多比赛中都有出现，相关的理论知识也有所普及。本文归纳总结了其中处理稀疏多项式乘法的方法，并提出了稀疏对称差卷积不需要离散对数的一般情况做法。

## 1 引言

集合幂级数是一类以“有限集合”为自变量、以数为取值的函数。在信息学竞赛中，许多问题可以使用这一工具进行解决，而从吕凯风学长的集训队论文 [1] 开始，有关的技术便被不断开发。其中，针对稀疏多项式的乘法在许多题目中也有所应用，例如 CF1119H 当中。笔者在该问题中提出了一类不同的做法，且具有泛化性，可以拓展到更一般的情况。

## 2 定义与约定

**记号 2.1.** 记集合 $U_n$ 为 $\{0, 1, \dots, n-1\}$。

**记号 2.2.** 设 $X$ 为一集合，我们用 $2^X$ 表示 $X$ 的幂集，即所有 $X$ 的子集构成的集合。

**记号 2.3.** $[P]$ 为艾佛森括号，即在表达式 $P$ 为真时取 1，为假时取 0。

**记号 2.4.** $A \cup B$ 表示 $A$ 与 $B$ 的并集，$A \cap B$ 表示 $A$ 与 $B$ 的交集。$A \oplus B$ 表示 $A$ 与 $B$ 的对称差，即 $A \oplus B = \{x \mid [x \in A] \neq [x \in B]\}$。用 $A \backslash B$ 表示 $A$ 和 $B$ 的差集，即 $A \backslash B = \{x \mid x \in A, x \notin B\}$。

**记号 2.5.** 对于一个自然数 $m$，定义其位集合为

$$\operatorname{supp}(m) = \{x \mid x \in \mathbb{N}, \lfloor m/2^x \rfloor \equiv 1 \bmod 2\}.$$

即 $m$ 的二进制展开中为 1 的位下标集合。

**定义 2.1.** 设 $F$ 为一个域，则称函数 $f : 2^{U_n} \to F$ 为 $F$ 上的一个集合幂级数。对每个 $S \subseteq U_n$，记 $f_S$ 表示把 $S$ 带入 $f$ 后的函数值，并称 $f_S$ 为该集合幂级数 $S$ 项的系数。

**定义 2.2.** 设 $f, g$ 为集合幂级数，定义 $h = f + g$，其中 $h$ 是一个集合幂级数，当且仅当对任意 $S \subseteq U_n$ 有 $h_S = f_S + g_S$。类似地定义减法。

**定义 2.3.** 对任意 $c \in F$，$S \subseteq U_n$，定义 $f = cx^S$ 为一个 $f_S = c$，其余项都为 0 的集合幂级数。同时对任意集合幂级数都有 $f = \sum_{S \subseteq U_n} f_S x^S$。这里 $x$ 是一个变元。

**定义 2.4.** 设 $f, g$ 为集合幂级数，定义 $h = fg$，其中 $h$ 是一个集合幂级数，当且仅当

$$h = \sum_{L \subseteq U_n} \sum_{R \subseteq U_n} f_L g_R x^{L \star R}$$

其中 $\star$ 是一个定义在 $2^{U_n}$ 上的二元运算，在本文中涉及的情况只包括 $\star \in \{\cup, \cap, \oplus\}$。这三类运算都满足交换律和结合律，因此可以定义集合幂级数的连乘。

下文探讨的稀疏多项式乘法是这样的一个问题

**问题 1.** 给定 $m$ 个稀疏多项式 $A_i$，求

$$\prod_{0 \le i < m} A_i$$

其中 $m$ 可达 $O(2^n)$ 级别，而第 $i$ 个多项式的非零项数 $k_i$ 是 $o(n)$ 的。同时为了讨论的方便，下面一般认为计算在模奇质数 $p$ 的域中进行。

# 3 集合并卷积

集合并卷积就是 $\star = \cup$ 时的集合幂级数乘法。本段会分为两个部分，分别介绍传统的并卷积算法以及如何在此基础上做稀疏多项式乘法。

## 3.1 快速莫比乌斯变换

**引理 3.1.** 两个集合幂级数的并卷积可以在 $O(n2^n)$ 复杂度内计算。

具体而言，我们设 $\hat{f}_S = \sum_{T \subseteq S} f_T$，那么在 $f$ 上的卷积实际上就是 $\hat{f}$ 上的对位乘法。而我们也可以通过 $f_S = \sum_{T \subseteq S} (-1)^{|S|-|T|} \hat{f}_T$ 由 $\hat{f}$ 得到 $f$。

由 $f$ 得到 $\hat{f}$ 的过程一般称为莫比乌斯变换（Moebius Transform），反过来一般称为莫比乌斯反演（Moebius Inversion）。

下面给出这两种变换的伪代码，不再进行更详细的原理解释，如有需要可以查看参考文献 [2]。

**Algorithm 1** 快速莫比乌斯变换

**输入：** 全集的大小 $n$，集合幂级数 $f$

**输出：** $f$ 的莫比乌斯变换

**for** $i = 0$ to $n-1$ **do**
  **for** $S \subseteq U_n \backslash \{i\}$ **do**
    $f_{S\cup\{i\}} \leftarrow f_{S\cup\{i\}} + f_S$
  **end for**
**end for**
**return** $f$

**Algorithm 2** 快速莫比乌斯反演

**输入：** 全集的大小 $n$，集合幂级数 $f$

**输出：** $f$ 的莫比乌斯反演

**for** $i = 0$ to $n-1$ **do**
  **for** $S \subseteq U_n \backslash \{i\}$ **do**
    $f_{S\cup\{i\}} \leftarrow f_{S\cup\{i\}} - f_S$
  **end for**
**end for**
**return** $f$

实际上，这两种变换还可以从更高的维度来理解。

**定义 3.1.** 对于 $n$ 维向量 $\mathbf{x} \in F^n$ 和集合 $S \subseteq U_n$，定义

$$\mathbf{x}^S = \prod_{i\in S} \mathbf{x}_i$$

**定义 3.2.** 对于 $n$ 维向量 $\mathbf{x} \in F^n$ 和集合幂级数 $f$，定义

$$f(\mathbf{x}) = \sum_{S\subseteq U_n} f_S \mathbf{x}^S$$

如果向量的每一维都是一个变元，那么我们得到的就是 $f$ 用 $n$ 元多项式表示的方法。在这个意义下考虑 $\mathbf{x}^S\mathbf{x}^T = \mathbf{x}^{S\cup T}$，取 $S = T = \{i\}$ 可以得到 $\mathbf{x}_i^2 = \mathbf{x}_i$，且容易验证这个限制是充要的。于是，我们可以把并卷积理解为 $h(\mathbf{x}) = f(\mathbf{x})g(\mathbf{x}) \bmod (\mathbf{x}_0^2 - \mathbf{x}_0, \dots, \mathbf{x}_{n-1}^2 - \mathbf{x}_{n-1})$。

而莫比乌斯变换就是给每个变元带入 $x^2 - x = 0$ 的两个根 $0, 1$ 求出的点值，莫比乌斯反演则是进行插值。

## 3.2 稀疏多项式乘法

现在考虑如何做多个稀疏多项式的乘法，然而直接做乘法是困难的。具体地，$O(n)$ 个二项式相乘就能得到一个 $2^n$ 项都有值的集合幂级数，不能维持稀疏的性质。

另一种处理连乘的常见方法是取 ln 再相加，考虑能否用它优化稀疏多项式乘法。

设 $f$ 是一稀疏集合幂级数，观察 $\ln(f)$ 的泰勒展开

$$\ln(f)=\sum_{i>0}(-1)^{i-1}\frac{(f-1)^i}{i}$$

此处的 $\ln(f)$ 定义并不是严谨的，因为没有考虑收敛性的问题。不过，这个定义主要起到启发性的作用，最后我们将直接展开做法流程来验算结果。

考虑 $(f-1)^i$ 都有哪些项有值，这里的乘法是并卷积。设 $f=\sum_i b_i x^{a_i}$，$|a|=|b|=k$，集合 $T=\{X\mid \exists B\subseteq U_k, X=\bigcup_{y\in B} a_y\}$，容易发现 $(f-1)^i$ 只在 $T$ 中的集合处有值。因此 $\ln(f)$ 也只在 $T$ 中的集合处有值。

由定义，$|T|\le 2^k$，而稀疏多项式的 $k$ 是 $o(n)$ 级别。于是只要能求出 $\ln(f)$，就能得到较低复杂度的乘法。

依据无限求和的定义求 ln 是困难的，不过利用 $\ln(f)\circ\mathbf{x}=\ln(f(\mathbf{x}))$，设 $g=\ln(f)$，我们有 $\hat{g}_S=\ln(\hat{f}_S)$。因此只需要在莫比乌斯变换后对值取对数再进行逆变换即可。

观察到我们并不需要进行大小为 $n$ 的莫比乌斯变换，考虑集合幂级数 $h=\sum_i b_i x^{\{i\}}$，我们有 $f(\mathbf{x})=h(\mathbf{y})$，其中 $\mathbf{y}_i=\prod_{j\in a_i}\mathbf{x}_j$。

我们实际要进行的操作是求出 $\ln(f(\mathbf{x}))\bmod(\mathbf{x}_0^2-\mathbf{x}_0,\dots,\mathbf{x}_{n-1}^2-\mathbf{x}_{n-1})$，做法是给每个 $\mathbf{x}_i$ 代入 $0,1$ 并插值。对于代入后的 $\mathbf{x}$，可以发现对应的 $\mathbf{y}$ 也是一个 01 向量，因此 $\mathbf{y}_i^2-\mathbf{y}_i=0$ 也是成立的。

于是对于求 $\ln(f(\mathbf{x}))\bmod(\mathbf{x}_0^2-\mathbf{x}_0,\dots,\mathbf{x}_{n-1}^2-\mathbf{x}_{n-1})$，转化为求 $\ln(h(\mathbf{y}))\bmod(\mathbf{y}_0^2-\mathbf{y}_0,\dots,\mathbf{y}_{k-1}^2-\mathbf{y}_{k-1})$ 并代入与 $\mathbf{x}$ 相应的 $\mathbf{y}$ 是正确的。所以我们可以直接对 $h$ 在 $U_k$ 上取 ln，然后代入 $\mathbf{y}_i=\prod_{j\in a_i}\mathbf{x}_j$ 得到 $\ln(f)$。

对值取对数并不能方便地在模意义下进行，但是在整个算法流程中，我们只需要在点值空间执行对数域中的加减，因此可以用乘法群（加上 0 的计数处理）模拟对数值，从而避免求离散对数甚至实数意义下的对数。

定义 $G=\{(x,c)\mid 1\le x<p,x,c\in\mathbb{Z}\}$，$(x,c)+(y,d)=(xy\bmod p,c+d)$，那么 $(G,+)$ 是一个以 $(1,0)$ 为单位元，以 $-(x,c)=(x^{-1}\bmod p,-c)$ 为逆元的交换群。

再定义 $A:\{0,\dots,p-1\}\to G$，$B:G\to\{0,\dots,p-1\}$。具体地，对 $x>0$ 有 $A(x)=(x,0)$，$A(0)=(1,1)$，$B((x,c))=[c=0]x$。

群 $(G,+)$ 和函数 $A,B$ 可以完成我们对对数的所有操作，具体可以参见下面的伪代码，其中 FMT、IFMT 就是快速莫比乌斯变换和反演。

因为我们是用函数来定义的集合幂级数，下面会用 $A(f)$ 来表示一个满足 $g_S=A(f_S)$ 的集合幂级数 $g$。

**Algorithm 3** 并卷积稀疏多项式乘法

**输入：** 全集的大小 $n$，稀疏集合幂级数 $f_0, \dots, f_{m-1}$

**输出：** 这些稀疏集合幂级数的乘积 $g$

$g'_S \leftarrow (1,0)$ for all $S \subseteq U_n$ {初始化为 $(G,+)$ 上的单位元}
**for** $t = 0$ to $m-1$ **do**
  $k = |a_t|$ {$f_t = \sum_i b_{t,i} x^{a_{t,i}}$}
  $h_S \leftarrow 0$ for all $S \subseteq U_k$
  $h_{\{i\}} \leftarrow b_{t,i}$ for all $0 \le i < k$
  $h \leftarrow \mathrm{FMT}(k, h)$ {在 $F_p$ 上做莫比乌斯变换}
  $h' \leftarrow A(h)$ {从 $F_p$ 上转到 $(G,+)$ 上}
  $h' \leftarrow \mathrm{IFMT}(k, h')$ {在 $(G,+)$ 上做莫比乌斯反演}
  **for** $S \subseteq U_k$ **do**
    $T = \bigcup_{i \in S} a_{t,i}$
    $g'_T \leftarrow g'_T + h'_S$
  **end for**
**end for**
$g' \leftarrow \mathrm{FMT}(n, g')$ {在 $(G,+)$ 上做莫比乌斯变换}
$g \leftarrow B(g')$ {从 $(G,+)$ 上转到 $F_p$ 上}
$g \leftarrow \mathrm{IFMT}(n, g)$ {在 $F_p$ 上做莫比乌斯反演}
**return** $g$

下面进行验算，只需验证 $m = 1$ 的情况即可。设稀疏集合幂级数 $f = \sum_{i<k} b_i x^{a_i}$，$h = \sum_{i<k} b_i x^{\{i\}}$，我们将做法展开，确认最终的 $g = f$。

$$\begin{aligned}
\ln(g) &= \ln(f) \\
&= \sum_{S \subseteq U_k} \ln(h)_S \prod_{i \in S} x^{a_i} \\
&= \sum_{S \subseteq U_k} \sum_{T \subseteq S} (-1)^{|S|-|T|} \ln(\hat{h}_T) x^{q_S} \\
&= \sum_{T \subseteq U_k} \ln(\hat{h}_T) \sum_{T \subseteq S \subseteq U_k} (-1)^{|S|-|T|} x^{q_S}
\end{aligned}$$

此处 $q_S = \bigcup_{i \in S} a_i$。考虑 $\ln(g)$ 的莫比乌斯变换 $\ln(\hat{g})$，取其集合 $Y$ 处的系数

$$\ln(\hat{g}_Y) = \sum_{T \subseteq U_k} \ln(\hat{h}_T) \sum_{T \subseteq S \subseteq U_k} (-1)^{|S|-|T|} [q_S \subseteq Y]$$

设 $X = \{x \mid a_x \subseteq Y\}$，第二个求和号可以化简为 $\sum_{T \subseteq S \subseteq X} (-1)^{|S|-|T|} = [T = X]$。

于是我们说明了做法求得的 $\ln(\hat{g}_Y) = \ln(\hat{h}_X) = \ln(\sum_{i \in X} b_i)$，是符合需求的。

这一做法的另一种理解方式是，用一个集合 $S$ 包含 $a$ 中哪些集合的 $2^k$ 种情况划分出 $2^k$ 种 $\hat{f}_S$ 的取值，然后对它做乘除法意义下的差分，得到 $2^k$ 个超集乘操作。

做法的复杂度为 $O(n2^n + \sum k_i 2^{k_i})$，其中 $k_i$ 是第 $i$ 个稀疏多项式的项数。朴素执行 $(G, +)$ 里的运算会带来 $O(\log p)$ 的求逆元代价，可以用分数 $x/y$ 来存储运算的中间结果，只在最后求一遍离线逆元，这样复杂度是加 $O(\log p)$ 而不是乘 $O(\log p)$。

## 4 集合交卷积、子集卷积

将集合全部替换为补集，即可把交卷积转化为并卷积。

子集卷积的定义如下

**定义 4.1.** 设 $f, g$ 为集合幂级数，定义 $h$ 是 $f, g$ 的子集卷积当且仅当对任意 $S \subseteq U_n$ 有 $h_S = \sum_{T \subseteq S} f_T g_{S \setminus T}$

容易验证，子集卷积也是满足交换律和结合律的。

子集卷积常见的处理方法是使用占位多项式 [3]，我们定义集合占位幂级数如下

**定义 4.2.** 设 $f$ 为集合幂级数，称 $\tilde{f}$ 是 $f$ 的集合占位幂级数当且仅当

$$\tilde{f} = \sum_{S \subseteq U_n} f_S x^S t^{|S|}$$

其中 $t$ 是一个满足 $t^i t^j = t^{i+j}$ 的变元。

可以发现，对于 $f, g$ 的子集卷积 $h$，我们有 $h_S = [x^S t^{|S|}] \tilde{f}\tilde{g}$，这里的乘法是并卷积。对于多个 $f_i$ 做子集卷积得到 $h$，同样有 $h_S = [x^S t^{|S|}] \prod_{0 \le i < m} \tilde{f}_i$。

于是我们照搬稀疏并卷积的做法，但对 $t$ 的多项式做运算。考虑用 $(G, +)$ 维护多项式的最低次项系数，用一个计数器 $c$ 维护最低次项次数，剩余部分是一个常数项为 1 的、次数不超过 $n$ 的多项式。

由于最终运算结果的 $c$ 一定是非负的，且我们只需要提取 $t$ 的次数不超过 $n$ 的项，所以常数项为 1 的剩余部分在运算时可以对 $t^{n+1}$ 取模。

我们对剩余部分求形式幂级数 ln（只需保留次数 $\le n$ 的项）然后直接进行加减法运算即可，总复杂度可以做到 $O(n^2(2^n + \sum 2^{k_i}))$。

## 5 集合对称差卷积

集合对称差卷积就是 $\star = \oplus$ 时的集合幂级数乘法。本段会分为两个部分，分别介绍传统的对称差卷积算法以及如何在此基础上做稀疏多项式乘法。

## 5.1 快速沃尔什变换

**引理 5.1.** 两个集合幂级数的对称差卷积可以在 $O(n2^n)$ 复杂度内计算。

具体而言，我们设 $\hat{f}_S = \sum_{T\subseteq U_n}(-1)^{|T\cap S|}f_T$，那么在 $f$ 上的卷积实际上就是 $\hat{f}$ 上的对位乘法。而我们也可以通过 $f_S = 2^{-n}\sum_{T\subseteq U_n}(-1)^{|T\cap S|}\hat{f}_T$ 由 $\hat{f}$ 得到 $f$。

由 $f$ 得到 $\hat{f}$ 的过程一般称为沃尔什变换（Walsh-Hadamard Transform），反过来一般称为沃尔什逆变换。

下面给出快速沃尔什变换的伪代码，逆变换只需要做完再给每项乘 $2^{-n}$ 即可。不再进行更详细的原理解释，如有需要可以查看参考文献 [4]。

**Algorithm 4** 快速沃尔什变换

**输入：** 全集的大小 $n$，集合幂级数 $f$

**输出：** $f$ 的沃尔什变换

```
for i = 0 to n − 1 do
    for S ⊆ U_n \ {i} do
        f_S, f_{S∪{i}} ← f_S + f_{S∪{i}}, f_S − f_{S∪{i}}
    end for
end for
return f
```

类似地，我们可以用对 $\mathbf{x}_i^2 - 1$ 取模来理解对称差卷积，而沃尔什变换就是带入其两个根 $1, -1$ 来求点值。

## 5.2 特殊情况的做法

目前较为普及的是两种特殊情况的做法，一种是稀疏多项式只有两个非 0 位的情况，另一种是每个稀疏多项式的系数可重集相同的情况。

### 5.2.1 只有两个非 0 位

问题的形式化定义如下

**问题 2.** 给定 $m$ 个二项形式幂级数 $u_ix^{a_i} + v_ix^{b_i}$，求出

$$\prod_{0\le i<m}(u_ix^{a_i} + v_ix^{b_i})$$

对于一个 $ux^a + vx^b$，我们可以把它写成 $x^a(u + vx^{a\oplus b})$。

单独处理前面的 $x^a$ 部分，后面的部分按照 $a \oplus b = c$ 的 $c$ 分组，每组内乘积仍然是 $u + vx^c$ 的形式，且容易计算。

现在问题转化为 $\prod_{S \subseteq U_n} (c_S + d_S x^S)$。

考虑分治乘，分治乘的每个区间形如 $[a2^b, (a+1)2^b)$，可以发现区间内的乘积一定形如 $\sum_{S \subseteq U_b} r_S x^S + x^{a2^b} \sum_{S \subseteq U_b} q_S x^S$。于是把 $\sum_{S \subseteq U_b} r_S x^S$ 和 $\sum_{S \subseteq U_b} q_S x^S$ 当作两个 $U_b$ 上的集合幂级数维护，每次合并做 $O(1)$ 次卷积即可做到 $O(n^2 2^n)$ 的总复杂度。

更进一步，我们可以只维护这两个集合幂级数在 $U_b$ 上做沃尔什变换的结果，通过一定的讨论可以说明每个操作都能在 $O(2^b)$ 的时间复杂度内进行，总复杂度 $O(n2^n + m)$，其中 $m$ 是稀疏多项式的个数。

### 5.2.2 系数可重集相同

下面的做法参考了 EternalAlexander 的博客 [5]。

问题的形式化定义如下

**问题 3.** 给定长为 $k$ 的系数数列 $w_0, \ldots, w_{k-1}$ 和 $m$ 个 $k$ 元组 $a_0, \ldots, a_{m-1}$，求

$$\prod_{0 \le i < m} \sum_{j=0}^{k-1} w_j x^{a_{i,j}}$$

可以发现，每个稀疏多项式的沃尔什变换中只有共用的 $2^k$ 种系数：$\sum_i v_i w_i$，其中 $v_i \in \{-1, 1\}$。

设答案集合幂级数的沃尔什变换为 $\hat{f}$，其每个系数 $\hat{f}_S$ 可以写作这 $2^k$ 种数的乘积，只需求出每种数在其中的幂次即可。

设 $A_T = \sum_{i=0}^{k-1} (-1)^{[i \in T]} w_i$，我们希望把每个 $\hat{f}_S$ 写成 $\prod_T A_T^{B_{S,T}}$ 的形式。

对于一个集合 $X$，考虑

$$g = \sum_{i=0}^{m-1} \prod_{j \in X} x^{a_{i,j}}$$

这一集合幂级数做沃尔什变换的结果 $\hat{g}$，我们有

$$\hat{g}_Y = \sum_{T \subseteq U_k} (-1)^{|T \cap X|} B_{Y,T}$$

于是对每个 $X$ 求出对应的所有 $\hat{g}_Y$ 之后，我们相当于得到了每个 $B_Y$ 的沃尔什变换。于是只需要做一次逆变换即可得到所有 $B_{Y,T}$，总复杂度 $O(n2^{k+n} + m2^k)$。

## 5.3 稀疏多项式乘法（离散对数）

还是考虑取 ln 然后做加法，设 $f=\sum_i b_i x^{a_i}$，$|a|=|b|=k$，集合 $T=\{X \mid \exists B\subseteq U_k, X=\bigoplus_{y\in B} a_y\}$，同样可以分析出 $\ln(f)$ 只在 $T$ 中的集合处有值。

与之前类似地，设 $h=\sum_i b_i x^{\{i\}}$，我们仍然能通过 $\ln(h)$ 简单得到 $\ln(f)$。

仍然利用 $\ln(f)\circ \mathbf{x}=\ln(f(\mathbf{x}))$，可以得到沃尔什变换后对位取 ln 再逆变换的做法。然而沃尔什逆变换会涉及除以 2，这在离散对数域下不一定能执行。

设给定的稀疏多项式为 $f_i$，答案为 $g$，考虑在 $m=1$ 的时候完整展开这个做法的流程，$m>1$ 是类似的。为了方便起见，下面用 $f$ 表示 $f_0$。

我们知道 $g=f$，设 $f=\sum_i b_i x^{a_i}$，$h=\sum_i b_i x^{\{i\}}$，$|a|=k$，我们有

$$\ln(g)=\sum_{T\subseteq U_k}\ln(h)_T\prod_{i\in T}x^{a_i}$$

为了书写方便，下面定义 $c_T=\bigoplus_{i\in T}a_i$，考虑 $\ln(h)$ 的沃尔什变换 $\ln(\widehat{h})$，可以把上式进一步写成

$$\ln(g)=\sum_{T\subseteq U_k}\sum_{S\subseteq U_k}(-1)^{|S\cap T|}2^{-k}\ln(\widehat{h}_S)x^{c_T}$$

考虑取 $\ln(g)$ 的沃尔什变换 $\ln(\hat{g})$ 在集合 $Y$ 处的系数，于是我们有

$$\begin{aligned}\ln(\hat{g}_Y)&=\sum_{T\subseteq U_k}\sum_{S\subseteq U_k}(-1)^{|S\cap T|}2^{-k}\ln(\widehat{h}_S)(-1)^{|c_T\cap Y|}\\&=\sum_{S\subseteq U_k}2^{-k}\ln(\widehat{h}_S)\sum_{T\subseteq U_k}(-1)^{|S\cap T|}(-1)^{|c_T\cap Y|}\end{aligned}$$

根据 $c_T$ 的定义，设 $X=\{x\mid 0\le x<k, |a_x\cap Y|\equiv 1 \bmod 2\}$，可以发现第二个求和号的结果为 $2^k[X=S]$。

于是可以把式子整合为

$$\begin{aligned}\ln(\hat{g}_Y)&=\sum_{S\subseteq U_k}\ln(\widehat{h}_S)[X=S]\\&=\ln(\widehat{h}_X)\end{aligned}$$

这一结果不仅符合我们的直觉，也说明了做法是对的。同时它指出每个 $\ln(\hat{g}_Y)$ 都总是 $\ln(\widehat{h}_S)$ 的整系数线性组合。

于是显式求出离散对数的做法仍然可以实施，具体地，设给定的稀疏多项式最大大小为 $K$，我们把所有对数写成 $\frac{v}{2^K}$ 的形式，$v$ 在运算中对 $2^K(p-1)$ 取模而非对 $p-1$ 取模。根据线性组合的系数是整数，最后所有 $D_Y$ 对应的 $\frac{v}{2^K}$ 都满足 $2^K\mid v$。

这里可以再进行一步优化，考虑把集合看作 $F_2$ 上的 $n$ 维向量、异或看作加法，此时 $T$ 实际上就是 $a$ 张成的线性空间，$T$ 和 $a$ 定义同第一段。

因此我们可以先得到 $T$ 的一组基，不妨设为 $z_0,\dots,z_{t-1}$，此时每个 $a_i$ 可以写作 $z$ 的一个子集 $S_i$ 的和，即 $a_i=\sum_{j\in S_i} z_j$。我们将 $h=\sum_i b_i x^{\{i\}}$ 替换为 $h=\sum_i b_i x^{S_i}$，$c_T$ 的定义也改为 $\bigoplus_{i\in T} z_i$（此处认为 $z_i$ 是向量对应的集合），容易发现做法仍然是成立的。这样我们就把 $h$ 从 $U_k$ 上的集合幂级数转为 $U_t$ 上的集合幂级数了。

总复杂度 $O(n2^n+\sum t_i 2^{t_i}+P(\sum 2^{t_i}))$，其中 $P(x)$ 是求 $x$ 次离散对数的复杂度，$t_i$ 是第 $i$ 个集合幂级数对应的线性空间维数。

然而求离散对数的复杂度一般含有 $p$ 的多项式因子，在模数很大的时候无法进行。下面将介绍一种稍麻烦一点，但是可以避开离散对数的做法。

## 5.4 稀疏多项式乘法（二次剩余）

现在考虑一个简化版的问题，最终除的不是 $2^K$ 而是 2：

**问题 4.** 给定 $n,m,p,k$，有一个长为 $n+m$ 的值域在 $[1,p)$ 的整数序列 $x_1,\dots,x_{n+m}$。$x_1,\dots,x_n$ 已经给定，$x_{n+1},\dots,x_{n+m}$ 是由之前的两个 $x_i$ 进行乘除法得到的。现在给定一些下标 $a_1,\dots,a_k$，保证每个 $x_{a_i}$ 写成 $\prod_{1\le j\le n} x_j^{y_j}$ 的形式后 $y_j$ 是偶数。求出每个 $x_{a_i}$ 对应的 $\prod_{1\le j\le n} x_j^{y_j/2} \bmod p$，保证 $p$ 是奇质数。

解法　如果 $x_1,\dots,x_n$ 全都是二次剩余，我们可以直接求出 $z_1,\dots,z_n$ 满足 $z_i^2\equiv x_i \pmod p$，直接用 $z_i$ 代替 $x_i$ 做乘除法。由于需要求值的 $x_{a_i}$ 满足每个 $y_j$ 为偶数，$z_i$ 选取哪个根不影响答案。

考虑二次剩余的结构

**引理 5.2.** 对于任意模 $p$ 的非二次剩余 $x,y$，一定有 $xy$ 是二次剩余。

**引理 5.3.** 模 $p$ 的非零二次剩余和非二次剩余各有 $\frac{p-1}{2}$ 个。

**引理 5.4.** $x\neq 0$ 是模 $p$ 的二次剩余当且仅当 $x^{\frac{p-1}{2}}=1$。

**引理 5.5** (Cipolla)**.** 存在一个 $O(\log p)$ 时间复杂度求出模意义下平方根的算法。

证明与本文关系不大，暂且略过。

根据这些性质，对于一个非二次剩余 $z$，任意非二次剩余 $d$ 都能写作 $d=zq^2$ 的形式。于是考虑在交换群 $(G,\times)$ 中做运算，其中 $G=\{(x,y)\mid x\in\{1,\dots,p-1\},y\in\{0,1\}\}$，乘法运算为 $(a,b)\times(c,d)=(acz^{bd} \bmod p,(b+d) \bmod 2)$，逆元为 $(a,b)^{-1}=(a^{-1}z^{-b} \bmod p,b)$，单位元为 $(1,0)$。

对于二次剩余 $d=q^2$，其对应 $G$ 中的 $(q,0)$ 元素。对于非二次剩余 $d=zq^2$，其对应 $(q,1)$。

容易说明所有需要求值的 $x_{a_i}$ 都形如 $(q,0)$，且 $q$ 就是我们需要的答案。

而寻找这个非二次剩余可以做到 $O(\log p)$ 的时间复杂度，因为只需要随机 $O(1)$ 次并检查。现在我们做到了 $O(n\log p+m)$ 的总复杂度，同样用分数形式避免在线的除法。 ■

对于除 $2^K$ 的情况，同样对于一个非二次剩余 $z$，可以把所有数 $v$ 都写成 $z^a y^{2^k}$ 的形式，其中 $0\le a<2^k$，然后就能用类似上面的方法解决。具体实现可以考虑递推，如果对于 $j$ 已知 $v=z^a y^{2^j}$，将 $y$ 分解为 $z^u w^2$，我们就能得到 $v=z^{a+u2^j}w^{2^{j+1}}$。

容易发现，上一节的稀疏对称差卷积中的运算流程和这个简化版是一致的，于是我们得到了一个不需要离散对数的做法。

因为其代码逻辑基本和并卷积的相同，下面不再给出伪代码。最终的复杂度为 $O(n2^n+\sum t_i 2^{t_i}\log p)$，其中 $t_i$ 定义同上。

## 5.5 例题

**例题 1** (黎明前的巧克力[1]). 给定 $m$ 和集合序列 $a_1,a_2,\dots,a_m$，求 $\prod_{i=1}^{m}(1+2x^{a_i})$，其中乘法是对称差卷积。$m\le 10^6$，$a_i\subseteq U_n$，$n\le 20$。

解法 使用之前的任意一种做法均可，其中两种特殊情况的做法复杂度为 $O(m+n2^n)$。■

**例题 2** (Triple[2]). 给定 $m,u,v,w$ 和 $m$ 个集合三元组 $a_i,b_i,c_i$，求 $\prod_{i=1}^{m}(ux^{a_i}+vx^{b_i}+wx^{c_i})$。$a_i,b_i,c_i\subseteq U_{17}$，$1\le m\le 10^5$。

解法 使用通用做法或者系数可重集相同的做法均可。 ■

**例题 3** (Bitwise Magic[3]). 给定 $m,k,n$，以及 $m$ 个自然数 $b_i\ge k$，求

$$\left[\frac{t^k}{k!}\right]\prod_i\sum_{j=0}^{k}x^{\text{supp}(b_i-j)}\frac{t^j}{j!}$$

其中 $t$ 是形式幂级数的变元，$x$ 是集合幂级数的变元。$k,n\le 16$，$m\le 2^n$，$b_{i,j}\subseteq U_n$。

解法 可以发现，连续的 $k$ 个整数的线性基大小是 $\log k+O(1)$ 的，于是直接在那个线性基上做集合幂级数 ln 即可做到 $O(k^3m+k(k+n)2^n)$ 的复杂度。没有二次剩余/离散对数的复杂度是因为取 ln 的形式幂级数常数项总为 $\pm1$。

进一步观察可以发现取 ln 的形式幂级数形如 $\sum a_i\frac{t^i}{i!}$，其中 $a_i$ 为 $\pm1$。于是只有 $O(2^k)$ 种不同的形式幂级数，预处理它们 ln 的结果可以做到 $O(k^2(2^k+m+2^n)+kn2^n)$。

事实上，考虑 $\text{supp}(b)\oplus\text{supp}(b),\dots,\text{supp}(b)\oplus\text{supp}(b-k)$ 这一序列，可以发现 $b$ 高于 $\log k$ 的位对其影响只有最后连续的 1 个数，也就是只有 $O(n)$ 种。因此提出 $x^{\text{supp}(b)}$ 后不同集合幂级数只有 $O(kn)$ 种，预处理的复杂度也可以降低为 $O(k^4n)$。

[1] https://uoj.ac/problem/310

[2] https://codeforces.com/problemset/problem/1119/H

[3] https://codeforces.com/problemset/problem/1408/I

最后，如果不保证 $m$ 与 $2^n$ 同阶，也可以一次性处理 $b_i$ 相同的所有输入，总复杂度为 $O(k^4n+k(k+n)2^n+m)$。

解法参考了 qwaszx 的题解[4]。 ■

**例题 4** (Wonderful XOR Problem[5]加强版)**.** 求

$$\prod_{i=0}^{m-1}\sum_{j=0}^{k-1}b_{i,j}\sum_{l=0}^{a_{i,j}-1}x^{\text{supp}(l)}$$

$m\le 10^4, n\le 18, a_{i,j}\le 2^n, k\le 4$，原题保证 $m\le 2\times 10^5, k\le 2$。

解法　对于一组 $A_0,\dots,A_{k-1}$、$B_0,\dots,B_{k-1}$，设

$$f=\sum_{j=0}^{k-1}B_j\sum_{l=0}^{A_j-1}x^{\text{supp}(l)}$$

我们有

$$\hat{f}_S=\sum_{j=0}^{k-1}B_j\sum_{l=0}^{A_j-1}(-1)^{|\text{supp}(l)\cap S|}$$

观察到

$$\sum_{i=(u-1)2^v}^{u2^v-1}(-1)^{|\text{supp}(i)\cap S|}=[\text{supp}(2^v-1)\cap S=\emptyset]2^v(-1)^{|\text{supp}((u-1)2^v)\cap S|}$$

考虑把 $[0,A-1]$ 拆成这样的区间，设 $S$ 中的最小元素为 $q$（$S=\emptyset$ 时单独处理），$u_i=\lfloor A/2^i\rfloor$，可以得到

$$\begin{aligned}\sum_{i=0}^{A-1}(-1)^{|\text{supp}(i)\cap S|}&=\sum_{i\in\text{supp}(A)}\sum_{j=(u_i-1)2^i}^{u_i2^i-1}(-1)^{|\text{supp}(j)\cap S|}\\&=\sum_{i\in\text{supp}(A),i\le q}2^i(-1)^{|\text{supp}((u_i-1)2^i)\cap S|}\\&=\sum_{i\in\text{supp}(A),i\le q}2^i(-1)^{|\text{supp}(A)\cap S|-[i<q,q\in\text{supp}(A)]}\end{aligned}$$

讨论 $q$ 是否在 $\text{supp}(A)$ 中，总可以把原式写作 $B(-1)^{|\text{supp}(A)\cap S|}$ 的形式，其中 $B$ 只和 $A,q$ 有关。

于是枚举 $q$，我们就把原问题转化为了

$$\prod_{i=0}^{m-1}\sum_{j=0}^{k-1}b'_{i,j}x^{\text{supp}(a_{i,j})}$$

可以发现，每次的子问题的值域可以缩小为 $O(2^{n-q})$，于是总复杂度由等比数列求和得到 $O(n2^n+nmk2^k\log p)$，$k=2$ 的情况可以用特殊情况的做法做到 $O(n2^n+nm)$。 ■

[4]https://www.luogu.com.cn/article/t3eg92u9

[5]https://codeforces.com/problemset/problem/2096/H

## 6 总结

本文分别总结了稀疏集合并/交卷积、子集卷积、集合对称差卷积在不同情况下的做法。其中稀疏并卷积的部分之前相关研究较少，且基本没有例题。而稀疏对称差卷积此前的做法 [6] 需要离散对数，不能较好地处理模数很大的情况。例题则主要侧重特殊情况下的做法，拓展性较差。

本文优化了稀疏对称差卷积的做法，在模大质数时只需要二次剩余而非离散对数。然而所有的做法都依赖模数是质数，并没有给出在更一般的域上也能运行的做法。希望本文能起到抛砖引玉的作用，促进更多人在这一方面进行研究。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队教练任舍予的指导。
感谢父母对我的培养和教育。
感谢汪星明老师的教导和同学们的帮助。
感谢朱鹏睿、陈信允、许淇文同学与我交流讨论。
感谢时庆钰学长、赵海鲲同学对本文提供的宝贵建议。
感谢其他给予我帮助的老师与同学。

## 参考文献

[1] 吕凯风. 集合幂级数的性质与应用及其快速算法. IOI 2015 中国国家候选队论文集, 2015.

[2] F. Yates. The design and analysis of factorial experiments. Technical Communication 35, Commonwealth Bureau of Soils, Harpenden, U.K., 1937.

[3] 李白天. 信息学竞赛中的生成函数计算理论框架. IOI 2021 中国国家候选队论文集, 2021.

[4] B. J. Fino and V. R. Algazi. Unified matrix treatment of the fast Walsh–Hadamard transform. IEEE Transactions on Computers, vol. C-25, no. 11, pp. 1142–1146, 1976.

[5] EternalAlexander. CF1119H Triple. Online article, 2021. https://www.luogu.com.cn/article/xyia17hl.

[6] fast_photon. 一种不太快速计算多个少项式集合幂级数异或卷积的方法. Online article, 2025. https://qoj.ac/blog/fast_photon/blog/1934.

# 浅谈无向图的唯一分解

南京外国语学校 许淇文

## 摘 要

无向图的笛卡尔积是一类重要的图运算，在图结构分解与图同构等问题中具有基础性地位。与正整数的质因数分解的算术基本定理类似，Sabidussi-Vizing 定理指出，任何有限的无向图均能被唯一地分解为若干个素图的笛卡尔积。本文讨论了无向图笛卡尔积的性质，并介绍一种对给定连通无向图求出唯一分解的算法。

## 1 定义与记号

我们首先引入本文中使用的图论有关的基本概念。对于无向图 $G = (V, E)$，我们用 $V(G)$ 表示 $G$ 的顶点集合，$E(G) \subseteq V(G) \times V(G)$ 表示 $G$ 的边集。边集的每个元素 $(u, v) \in E(G)$ 描述图中的一条边，也记作 $uv \in E(G)$。我们定义 $d_G(u, v)$ 为图 $G$ 中两点 $u, v$ 之间的距离。我们定义 $x$ 的邻居集合 $N(x)$ 为所有与 $x$ 直接相邻的顶点集合，称 $a$ 是 $u$ 和 $v$ 的公共邻居，当且仅当 $a \in N(u)$ 且 $a \in N(v)$。在本文中，我们称两张无向图 $G, H$ 相等，当且仅当 $G$ 和 $H$ 在不区分顶点编号的意义下同构。

**定义 1** (无向图的笛卡尔积)**.** 无向图 $G = (V(G), E(G))$ 和 $H = (V(H), E(H))$ 的笛卡尔积 $I = G \square H$ 为一张无向图，其点集为 $V(I) = V(G) \times V(H)$（即，集合 $V(G)$ 与 $V(H)$ 的笛卡尔积）。两个顶点 $(a, x)$ 和 $(b, y)$ 相邻，当且仅当 $ab \in E(G)$ 且 $x = y$，或 $a = b$ 且 $xy \in E(H)$。

无向图的笛卡尔积运算有单位元 $K_1$，且满足交换律和结合律。

**定义 2** (笛卡尔积分解)**.** 对于无向图 $G$，若其等于若干无向图 $G_1, G_2, \ldots, G_n$ 的笛卡尔积，即 $G = G_1 \square G_2 \square \cdots \square G_n$，则称 $G_1, G_2, \ldots, G_n$ 为 $G$ 的一组笛卡尔积分解。

**定义 3** (笛卡尔积的坐标表示)**.** 对于无向图 $G$ 的笛卡尔积分解 $G = G_1 \square G_2 \square \cdots \square G_n$，其坐标表示是将 $V(G)$ 中的每个点用 $n$ 维向量标号的方案，使得图 $G$ 中两点 $u, v$ 之间有边连接，当且仅当：

1. $u$ 和 $v$ 的坐标向量恰有一维值不同；
2. 设其第 $i$ 维的值为 $a, b$ 且 $a \neq b$，则图 $G_i$ 中顶点 $a, b$ 有边连接。

**定义 4** (笛卡尔积的染色表示)**.** 对于无向图 $G$ 的笛卡尔积分解 $G = G_1 \square G_2 \square \cdots \square G_n$，其染色表示是将 $E(G)$ 中的每条边用 $n$ 种颜色中的某个染色的方案，使得颜色为 $i$ 的边连接的两个顶点，在坐标表示中仅有第 $i$ 维值不同。

对于一条边 $e = uv$，我们使用 $c(e)$ 或 $c(uv)$ 指代其在染色表示中的颜色。

## 2 唯一分解定理

**定义 5** (素图)**.** 设 $G$ 为有限无向简单图。若不存在两个无向简单图 $G_1, G_2$，使得

$$G \cong G_1 \square G_2,$$

其中 $|V(G_1)| \geq 2$ 且 $|V(G_2)| \geq 2$，则称 $G$ 在笛卡尔积意义下是素的（或称 $G$ 是一张素图）。

**定理 1** (Sabidussi–Vizing 定理)**.** 任何一个有限连通无向图 $G$ 都可以在同构意义下唯一分解成有限个素图的笛卡尔积 $G = G_1 \square G_2 \square \cdots \square G_n$。

在 1959 年和 1963 年，G. Sabidussi [2] 和 V. G. Vizing [3] 分别独立给出了该定理的证明，定理的证明略为复杂，超出了本文的讨论范围。其具体的证明可参考 [2][3]。在此处，我们不加证明地给出一种分解方式。在此之前，我们给出如下定义：

**定义 6.** 对于边集 $E(G)$ 中的两条边 $e = xy$ 和 $f = uv$，我们称 $e \sim f$ 当且仅当以下两个条件之一满足：

1. $d_G(x, u) + d_G(y, v) \neq d_G(x, v) + d_G(y, u)$；
2. $x = u$，且其为 $y$ 和 $v$ 的唯一共同邻居。

**定义 7.** 我们称 $e \approx f$，当且仅当存在一个边的序列 $e = e_0, e_1, \ldots, e_k = f$，满足对于 $i = 1, 2 \ldots k$，均有 $e_{i-1} \sim e_i$。即 $\approx$ 是 $\sim$ 在边集上的传递闭包。

在无向图的唯一分解的染色表示中，两条边 $e$ 和 $f$ 颜色相同，当且仅当 $e \approx f$。该构造也表明了将无向图分解为若干个素图的笛卡尔积的唯一分解是良定义的。

## 3 利用初始信息求出分解

对于一般的，不保证连通的无向图 $G$，求出它的笛卡尔积唯一分解是困难的。对于恰有两个连通块 $H_1, H_2$ 的图 $G = H_1 \cup H_2$，其笛卡尔积唯一分解中包含因子 $I_2$（由 2 个顶点和 0 条边组成的无向图）当且仅当 $H_1$ 与 $H_2$ 在不区分顶点编号的意义下同构。因此，图同构判

定问题可以在多项式时间内规约到一般无向图的笛卡尔积分解问题。故后文中我们仅讨论连通无向图的情形，并默认提到的无向图连通。

对于图 $G$ 的笛卡尔积分解 $G = G_1 \square G_2 \square \cdots \square G_n$，我们接下来给出一个算法，在已知该分解的染色表示中，所有与某顶点 $v_0$ 相邻的边的颜色时，复原所有边的颜色。在接下来的讨论中，我们假定存在一个合法的笛卡尔积分解的染色表示，对应已知的边颜色信息。

我们首先以 $v_0$ 为起点对 $G$ 进行 BFS，并按照与 $v_0$ 的距离对所有顶点分层。形式化地，我们定义 $L_i = \{v \mid d_G(v_0, v) = i\}$。首先，我们给出重要的引理：

**引理 1** (正方形引理)**.** 在无向图 $G$ 的笛卡尔积的染色表示中，若两条有公共顶点的边 $e$ 和 $f$ 颜色不同，则存在恰好一个 $C_4$ 包含边 $e, f$，且该 $C_4$ 没有对角线。设该 $C_4$ 的四条边按顺序分别为 $e, f, g, h$，则 $e$ 和 $g$ 颜色相同，$f$ 和 $h$ 颜色相同。

证明. 不妨设 $e$ 的颜色为 $i$，$f$ 的颜色为 $j$，且 $e = uv$ 和 $f = vw$ 有公共顶点 $v$。

则坐标表示中，$u$ 和 $v$ 恰有第 $i$ 维不同，$v$ 和 $w$ 恰有第 $j$ 维不同，由笛卡尔积的结构显然可知 $u, w$ 至少存在一个除 $v$ 外的公共邻居。

由于任意两个相邻的顶点的坐标恰有一个维度不同，且 $u$ 和 $w$ 恰有两个维度坐标不同，故 $u$ 和 $w$ 的公共邻居至多有两个，即 $u$ 和 $w$ 除 $v$ 外的公共邻居恰好有一个。

不妨设这个公共邻居为 $x$，则由 $x \neq v$ 可知 $x$ 和 $u$ 恰有第 $j$ 维不同，$x$ 和 $w$ 恰有第 $i$ 维度不同，直接说明了这个 $C_4$ 中相对的边的颜色相同。

且由于 $u$ 和 $w$，$v$ 和 $x$ 分别各有两维坐标不同，可知 $u$ 与 $w$ 不相邻，$v$ 与 $x$ 不相邻，即该 $C_4$ 没有对角线。引理成立。

□

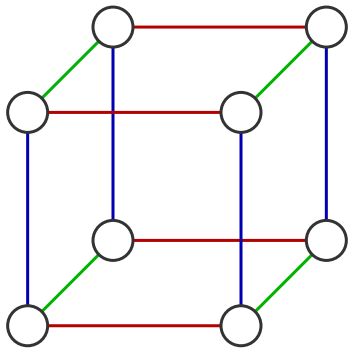


图 1: $G = C_2 \square C_2 \square C_2$

图 1 展示了立方体图的笛卡尔积分解 $G = C_2 \square C_2 \square C_2$ 对应的染色表示。其中任意一点的任意两条相邻边均颜色不同，由正方形引理可知恰好存在一个 $C_4$ 包含这两条边，其对应了立方体中包含这两条边的一个面。

由正方形引理的证明可以立即得到，对于其中涉及的边颜色不全相同的 $C_4$，设其顶点分别按顺序为 $x, y, z, w$，则对于任意顶点 $a$，有 $d_G(x,a) + d_G(z,a) = d_G(y,a) + d_G(w,a)$。

接下来，我们按顺序对每一层之间的边复原其颜色。

## 3.1 复原 $L_i$ 内部的边的颜色

对于 $L_i$ 内部的边 $uv$，我们找到 $u$ 在 BFS 树上的父亲 $x$，此时已经复原了 $ux$ 的颜色。

我们找到 $x$ 和 $v$ 在 $L_{i-1}$ 中的共同邻居 $y$。若 $y$ 不存在，则 $c(uv) = c(ux)$，否则 $c(uv) = c(xy)$。

这里利用了正方形引理的两个推论：

**推论 1.** 在无向图 $G$ 的笛卡尔积的染色表示中，若两条边 $e = uv$ 和 $f = vw$ 有公共顶点 $v$，且 $v$ 是 $u$ 和 $w$ 唯一的公共邻居，则 $e$ 和 $f$ 颜色相同。

证明. 若 $e$ 与 $f$ 颜色不同，则由正方形引理可知 $u$ 和 $w$ 必然存在除 $v$ 以外的公共邻居，与前提条件矛盾。

故推论成立。

□

**推论 2.** 在无向图 $G$ 的笛卡尔积的染色表示中，对于任意 $C_4$，设其四条边按顺序分别为 $e, f, g, h$，则 $e$ 与 $g$ 颜色相同，$f$ 与 $h$ 颜色相同。

证明. 若 $e$ 与 $g$ 颜色不同，则 $f$ 与 $e, g$ 中至少一个颜色不同。不妨设 $e$ 和 $f$ 颜色不同。

则由正方形引理可知，存在恰好一个 $C_4$ 包含 $e$ 和 $f$，则该 $C_4$ 即为 $efgh$，则可推得 $e$ 与 $g$ 颜色相同，得到矛盾。

$f$ 与 $h$ 的情况是对称的。故推论成立。

□

## 3.2 复原 $L_{i-1}$ 与 $L_i$ 之间的边的颜色

对于 $i = 1$ 的情形，此时有 $L_{i-1} = L_0 = \{v_0\}$，这些边的颜色均已知。

对于 $i > 1$ 的情形，考虑 $L_i$ 中的每个顶点 $u$。若 $u$ 仅有一条连向 $L_{i-1}$ 的边，设为 $uv$，则找到 $v$ 在 BFS 树上的父亲 $w$，有 $c(uv) = c(vw)$。

否则，考虑所有边 $uv_1, uv_2, \cdots, uv_k$。对于每个 $2 \le i \le k$，我们求出 $v_1$ 和 $v_i$ 除 $u$ 之外的公共邻居 $x_i$。若 $x_i$ 不存在，则有 $c(uv_i) = c(uv_1)$。否则，有 $c(uv_1) = c(x_iv_i)$ 和 $c(uv_i) = c(x_iv_1)$。

特别地，若对于所有 $2 \le i \le k$，$x_i$ 均不存在，则类似地找到 $v_1$ 在 BFS 树上的父亲 $w$，有 $c(uv_1) = c(v_1w)$，可以确定所有颜色。

这部分的正确性同样容易用两个推论证明。

### 3.3 时间复杂度

在上述算法中，我们对于每条边至多进行一次查找两点公共邻居的操作。每次查询两点间连边情况是 $O(1)$ 的，故每次可以在 $O(n)$ 的时间复杂度内查询公共邻居，总时间复杂度为 $O(nm)$。

## 4 利用重标号优化时间复杂度

在前文所述的算法中，查找两点公共邻居的操作是复杂度瓶颈。注意到算法中用到的查找公共邻居操作，本质是在寻找边的颜色不全相同的 $C_4$。而这些 $C_4$ 实际上对应了笛卡尔积分解中，连向某顶点的两条异色边，其所对应的形态在图中较为特殊。这启发我们利用已经求出的边的信息，来加速查找公共邻居的过程。

**定义 8** (层)**.** 在无向图 $G$ 的笛卡尔积的染色表示中，对于某种颜色 $i$，我们称该颜色的边的生成子图中的一个极大连通分量为一层。由坐标表示和颜色表示的定义可知，一层内所有顶点的坐标除第 $i$ 维外全部相同。进一步地，我们称某种颜色 $i$ 的所有层中，坐标第 $i$ 维相同的顶点为对应的顶点，类似定义两端点分别对应的边为对应的边。

当某种颜色 $i$ 的某一层包含初始顶点 $v_0$ 时，我们称其为该颜色的顶层。

我们称一条边 $e = uv$ 的标号为 $l(e)$ 或 $l(uv)$。在算法过程中，我们需要保证，对于每种颜色的边，不同层之间对应的边标号相同，不对应的边标号不同，以此来快速定位不同层之间有对应关系的边。

在给出具体的算法之前，我们先给出正方形引理有关边标号的推广：

**引理 2** (推广的正方形引理)**.** 在无向图 $G$ 的笛卡尔积的染色表示中，若两条有公共顶点的边 $e$ 和 $f$ 颜色不同，则存在恰好一个 $C_4$ 包含边 $e, f$。设该 $C_4$ 的四条边按顺序分别为 $e, f, g, h$，则有 $c(e) = c(g), c(f) = c(h), l(e) = l(g), l(f) = l(h)$。

推广的正方形引理的证明与原引理本质相同，此处略去。

接下来，我们沿用前文所述的算法思路，在复原边的颜色的同时，对边进行重标号。

### 4.1 处理 $L_i$ 内部的边

对于 $i = 1$ 的情形，此时 $L_i$ 内部的边均为顶层边，我们直接在染色后给其自由分配标号，不和已分配的标号重复即可。

对于 $i > 1$ 的情形，考虑 $L_i$ 内部的边 $uv$，我们找到 $u$ 在 BFS 树上的父亲 $x$。若 $c(uv) \neq c(ux)$，则由正方形引理，$x$ 和 $v$ 必然有除 $u$ 之外的公共邻居 $y$，且有 $c(vy) = c(ux), l(vy) = l(ux)$，故我们可以利用边的标号 $O(1)$ 找到这样的顶点 $y$，并判断其是否和 $x$ 相邻。且此时有 $l(uv) = l(xy)$。

否则有 $c(uv) = c(xu)$。此时我们找到 $u$ 连向 $L_{i-1}$ 的边中某条颜色与 $uv$ 不同的边 $uy$。如果这样的边 $uy$ 不存在，则说明 $uv$ 是一个顶层中的边，可以自由分配标号。否则利用正方形引理，找到包含 $u, v, y$ 的 $C_4$ 并给边 $uv$ 标号。

## 4.2 处理 $L_{i-1}$ 与 $L_i$ 之间的边

对于 $i = 1$ 的情形，所有涉及到的边都是颜色已经给定的顶层边，直接自由分配标号即可。

接下来考虑 $i > 1$ 的情形。首先找到所有 $u$ 连向 $L_{i-1}$ 的边，若这样的边仅有一条，则显然这条边是顶层的一条边，染色后自由分配标号即可。

对于这样的一条边 $uv$，其中 $v \in L_{i-1}$，我们找到 $v$ 在 BFS 树上的父亲 $x$，在 $O(\deg_u)$ 的复杂度内找到 $x$ 和 $u$ 除 $v$ 外的公共邻居 $y$。若 $y$ 不存在则有 $c(uv) = c(vx)$，否则有 $c(uv) = c(yx)$。

故我们可以在 $O(\deg_u)$ 的时间复杂度内确定某条 $uv$ 的颜色。我们找到 $v$ 连向 $L_{i-2}$ 的某条边 $va$，满足 $c(va) \neq c(uv)$。若这样的 $a$ 不存在，则 $v$ 位于颜色 $c(uv)$ 的顶层，故所有 $u$ 连向 $L_{i-1}$ 的边均为顶层边且颜色为 $c(uv)$，此时可以自由分配标号。

否则 $a$ 和 $u$ 一定存在除 $v$ 以外的公共邻居 $w$，我们可以在 $O(\deg_u)$ 的时间内找到 $w$。此时有 $c(uw) = c(va) \neq c(uv)$。我们称 $uvaw$ 为点 $u$ 的主正方形，此时可以利用正方形引理确定 $l(uv)$ 和 $l(aw)$。

接下来我们枚举 $u$ 的剩余连边 $ux$，由于 $c(uv) \neq c(uw)$，故要么 $c(ux) \neq c(uv)$，要么 $c(ux) \neq c(uw)$，故我们只需要检查两种情况，并使用其中一种情况的正方形引理即可。具体地，我们找到和 $x$ 相邻的，标号为 $l(uv)$ 的边，如果其和 $u$ 相邻，则我们可以利用和 $uvx$ 相关的正方形引理，另一种情形是对称的。

于是我们在 $O(\deg_u)$ 的时间复杂度内给所有 $u$ 连向 $L_{i-1}$ 的边复原了颜色并求出了标号。

## 4.3 时间复杂度

处理每条 $L_i$ 内部的边的时间复杂度是 $O(1)$ 的，而处理每个点 $u$ 连向上一层的边的时间复杂度是 $O(\deg_u)$ 的，故整个算法的总时间复杂度是 $O(m + \sum_{u \in V(G)} \deg_u) = O(m)$ 的。

# 5 在无初始信息的情况下求出分解

前文描述的算法已经可以在有已知信息的情况下，关于边数线性的时间复杂度内求出无向图的笛卡尔积分解了。对于某个合法笛卡尔积分解的染色表示，若我们将其中的两种颜色合并为一种，容易说明得到的染色表示仍然对应了一个合法的笛卡尔积分解，等价于将原分解中的两个因子图替换为它们的笛卡尔积。

这启发我们，可以初始认为所有和 $v_0$ 相连的边颜色均不同，然后运行前述算法，并在过程中时刻检查当前求得的染色是否满足笛卡尔积的性质，若不满足则选择性合并两种颜色，直到性质重新被满足。

但前述算法的前提条件为，已知的信息对应合法的染色方案。为此，我们首先需要修改前述算法，使得能在运行过程中检查笛卡尔积的性质是否满足。

## 5.1 合法性检查

在算法按照 BFS 序分层运行的过程中，我们同步按照分层的顺序进行合法性检查。对于 $L_1$ 内部的边 $uv$，其限制条件仅有 $c(uv_0) = c(vv_0)$，故对 $L_1$ 进行合法性检查是简单的。下面假设对于 $i > 1$，我们已经确认了 $L_0, L_1, \ldots, L_{i-1}$ 内部的染色符合笛卡尔积的性质，我们需要检查 $L_i$ 是否同样符合。

对于每个 $u \in L_i$，我们找到两条 $u$ 连向 $L_{i-1}$ 的异色边，由于我们在重标号的过程中已经求出了主正方形，这步操作是 $O(1)$ 的。对于每条 $u$ 连向 $L_{i-1}$ 的边，以及所有与 $u$ 相连的 $L_i$ 内部的边，我们找到主正方形中与其颜色不同的边，并检查正方形引理是否满足。

总的检查次数是线性于边数的，且由于我们已经求出了标号，每次检查的时间复杂度可以做到 $O(1)$，故合法性检查的部分总时间复杂度仍然是 $O(m)$ 的。

通过分类讨论可以证明，若被检查的 $C_4$ 均满足正方形引理，则已完成检查的部分中任意的 $C_4$ 均同样满足正方形引理。

## 5.2 最终算法

在遇到算法执行过程中，当前染色与标号不符合正方形引理时，我们需要考虑合并颜色。而由于笛卡尔积分解每一层之间的对称性，且算法是按照 BFS 序进行的，故我们第一次遇到某个顶点不符合正方形引理时，其一定是对应颜色的顶层顶点，故我们直接将与其相邻的所有已处理的边合并成同一种颜色。且由于所有涉及的边均为顶层边，我们可以直接自由分配其标号。

在整个算法执行的过程中，颜色只会不断合并，不会出现新的颜色。在我们选择 $v_0$ 为度数最小的顶点时，有 $\deg_{v_0} \le n$ 且 $\deg_{v_0} n \le \sum_{u \in V(G)} \deg_u = 2m$，即 $\deg_{v_0}^2 \le 2m$，故可以使用 $O(\deg_{v_0})$ 单次合并，$O(1)$ 单次查询当前颜色的算法维护合并之后的颜色情况，这部分的复杂度仍然是 $O(m)$ 的，不影响整体复杂度。

最终算法的流程如下：

1. 找到度数最小的顶点 $v_0$，初始化与 $v_0$ 相连的边的颜色与标号，初始化邻接表信息。

2. 从 $v_0$ 开始 BFS，并按照与 $v_0$ 的距离对图分层，得到 $L_0, L_1, \ldots, L_k$。

3. 对于 $i$ 从 1 到 $k$，依次执行：

(a) 将 $L_{i-1}$ 与 $L_i$ 之间的边染色并标号。

(b) 将 $L_i$ 内部的边染色并标号。

(c) 将标号过程中违反正方形引理的点的相邻边的颜色合并。

(d) 对于 $L_i$ 进行合法性检查。

(e) 将合法性检查过程中违反正方形引理的点的相邻边的颜色合并。

4. 用染色结果构造方案并输出。

## 6 总结

本文围绕无向图在笛卡尔积意义下的唯一分解问题展开讨论。通过引入坐标表示与染色表示，我们将笛卡尔积分解转化为边染色与层结构的问题，并利用正方形引理刻画了不同颜色边之间的局部结构特征。

在此基础上，本文给出了一种基于 BFS 的染色算法，在已知初始染色信息的条件下，可以逐层复原整个图的染色表示，从而得到对应的笛卡尔积分解。进一步地，通过对边进行重标号并利用层与对应边的结构性质，成功将算法的时间复杂度从朴素实现的 $O(nm)$ 优化至关于边数线性。

在此基础上，对于无初始信息的情况，通过引入合法性检查并结合合并颜色的思想，给出了解决一般连通无向图的笛卡尔积分解的算法，并同样达到了关于边数线性的时间复杂度。

总体而言，本文给出了一种结构清晰、实现直接且时间复杂度最优的连通无向图笛卡尔积分解算法，为相关问题的算法研究与实现提供了有价值的参考。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢南京外国语学校张超老师、李曙老师的关心和指导。
感谢时庆钰学长为本文审稿，提供宝贵的修改意见。
感谢李秉霈同学、武林同学与我讨论问题。
感谢杜瑜皓学长对我的指导和帮助。
感谢家人、朋友对我的支持与鼓励。

# 8 参考文献

[1] Wilfried Imrich and Iztok Peterin. “Recognizing Cartesian products in linear time”. In: Discrete Mathematics 307.3 (2007), pp. 472–483.

[2] Gert Sabidussi. “Graph multiplication”. In: Mathematische Zeitschrift 72.1 (1959), pp. 446–457.

[3] Vladim G Vizing. “The Cartesian product of graphs”. In: Vycisl. Sistemy 9.30-43 (1963), p. 33.

# 浅谈信息熵在信息学竞赛中的应用

华南师范大学附属中学 叶隽霖

## 摘 要

在信息学竞赛之中，交互题、通信题等非传统题出现的频率越来越高。了解信息熵，对这一类问题的分析有很大帮助。

本文简要介绍了信息熵相关知识、ID3 算法及其局部化改良。并且利用改良的 ID3 算法得到一些交互题更优秀的解法。

## 1 定义

生活中会有各种各样的信息，让我们先考虑这两种信息：

1. 明天彩票的中奖号码**不是** $a$；
2. 明天彩票的中奖号码**是** $a$。

其中如果我们得知了信息 1，可能并不惊讶。因为随便选一组号码，能中的概率实在是太低了。这也表明了信息 1 给我们带来的信息量很小。

但是信息 2 是让我们惊讶的，因为 2 发生的概率实在是太低了。这表明信息 2 对我们带来的信息量很大。

于是我们使用**信息量**来描述一个信息传递的信息的量。

**定义 1** (信息量)**.** 对于一个发生概率为 $Pr(E)$ 的事件 $E$，我们定义其**信息量** $I(E) = -\log_b(Pr(E))$。

这里对数的底数可以是任意大于 1 的实数，常见的如 $2, e, 10$，单位分别是 bit、nat、hartley。

在接下来的讨论中，我们默认使用 2 为底数。

这么定义，主要是基于以下两个原因：

**定理 1.** 对于发生概率越低的事件，其信息量越大。反之对于发生概率越高的事件，其信息量越小。

**定理 2.** 对于独立事件 $A, B$。$AB$ 同时发生的信息量 $I(AB) = I(A) + I(B)$。

类似地，观察这两个概率分布：

| $x$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $Pr(X=x)$ | 0.97 | 0.01 | 0.01 | 0.01 |

| $y$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $Pr(Y=y)$ | 0.25 | 0.25 | 0.25 | 0.25 |

如果我们要预测 $X,Y$ 随机的结果，$X$ 是很好预测的，其取值大概率是 0，也有小概率是 $1,2,3$。而 $Y$ 在 $0,1,2,3$ 中均匀随机，会较为难以预测。

于是我们使用**信息熵**来描述一个随机变量的不确定性。

**定义 2** (信息熵)**.** 对于随机变量 $X$，其取值范围为 $\mathbb{X}$。则其信息熵为：

$$H(X)=\sum_{x\in\mathbb{X}}-Pr(X=x)\log_b(Pr(X=x))$$

一个随机事件的信息熵，反映了这个随机事件期望传达的信息量，也反映了这个事件的不确定性。一般而言一个事件不确定性越大，信息熵越大。

当 $Pr(X=x)=0$ 时，$0\times\log_2(0)$ 可以看作 0。这是因为 $\lim_{x\to0^+}x\log_2(x)=0$。

**定义 3** (确定性函数)**.** 随机变量 $Y$ 是随机变量 $X$ 的**确定性函数**，当且仅当每个 $x\in\mathbb{X}$ 存在恰好一个 $y\in\mathbb{Y}$ 满足 $Pr(X=x\wedge Y=y)>0$。

在信息学竞赛的交互题中，我们常常通过询问某些关于 $X$ 的确定性函数的取值来减低 $H(X)$，从而唯一确定 $X$。

**定理 3.** 对于取值范围有 $n$ 个元素的随机变量 $X$，其信息熵最大值是 $\log_2(n)$，且在 $X$ 均匀分布时取到。

证明. 令 $f(x)=-x\log_2(x)$。假设 $X$ 的分布概率分别为 $p_1,p_2,\ldots,p_n$。则：

$$\begin{aligned}H(X)&=\sum_{i=1}^{n}f(p_i)\\&\le n\times f\left(\frac{\sum p_i}{n}\right)\\&=n\times f\left(\frac{1}{n}\right)\\&=\log_2(n)\end{aligned}$$

第二步可以通过利用 $f(x)$ 在 $[0,1]$ 上上凸，并利用 Jensen 不等式得到。

□

**定义 4** (联合熵)**.** 对于两个随机变量 $X,Y$，定义**联合熵**为：

$$H(X,Y)=\sum_{x\in\mathbb{X}}\sum_{y\in\mathbb{Y}}p(X=x\wedge Y=y)\times I(X=x\wedge Y=y)$$

其描述了两个变量共同传达的信息量。

**推论 1** (确定性函数的联合熵)**.** $Y$ 是 $X$ 的一个确定性函数当且仅当 $H(X,Y)=H(X)$。

**定义 5** (条件熵)**.** 对于两个随机变量 $X,Y$，定义**条件熵**为：

$$H(X|Y)=\sum_{y\in\mathbb{Y}}p(Y=y)H(X|Y=y)$$

其描述了 $Y$ 确定后 $X$ 的不确定性。

**推论 2** (确定性函数的条件熵)**.** $Y$ 是 $X$ 的确定性函数当且仅当 $H(Y|X)=0$。

**推论 3** (联合熵、条件熵、信息熵之间的关系)**.** $H(X,Y)=H(X|Y)+H(Y)$。

证明.

$$\begin{aligned}H(X,Y)&=-\sum_x\sum_y P(x,y)\log P(x,y)\\&=-\sum_x\sum_y P(x|y)P(y)\log\left(P(x|y)P(y)\right)\\&=-\sum_x\sum_y P(x|y)P(y)\left(\log P(x|y)+\log P(y)\right)\\&=-\sum_x\sum_y P(x|y)P(y)\log P(x|y)-\sum_x\sum_y P(x|y)P(y)\log P(y)\\&=H(X|Y)+H(Y)\end{aligned}$$

□

**定理 4** (熵的链式法则)**.** 令 $X_1,X_2,\ldots,X_n$ 是随机变量，则：

$$H(X_1,X_2,\ldots,X_n)=H(X_1)+H(X_2|X_1)+H(X_3|X_2,X_1)+\cdots+H(X_n|X_{n-1},X_{n-2},\ldots,X_1)$$

证明.

$$\begin{aligned}H(X_1,X_2,\ldots,X_n)&=H(X_1)+H(X_2,X_3,\ldots,X_n\mid X_1)\\&=H(X_1)+H(X_2\mid X_1)+H(X_3,\ldots,X_n\mid X_1,X_2)\\&=\ldots\\&=H(X_1)+H(X_2\mid X_1)+H(X_3\mid X_1,X_2)+\cdots+H(X_n\mid X_1,X_2,\ldots,X_{n-1}),\end{aligned}$$

□

**定义 6** (互信息)**.** 对于两个随机变量 $X,Y$。定义其**互信息** $I(X;Y)=\sum_{x\in\mathbb{X},y\in\mathbb{Y}}Pr(X=x\wedge Y=y)\log_2\left(\dfrac{Pr(X=x\wedge Y=y)}{Pr(X=x)Pr(Y=y)}\right)$。其描述了一个随机变量包含多少另一个变量的信息。

**推论 4.** $I(X;Y)=H(X)-H(X|Y)=H(Y)-H(Y|X)=H(X)+H(Y)-H(X,Y)$

我们可以发现，互信息描述了，当我们知道变量 $Y$ 时，$X$ 的信息熵期望的下降数。

**推论 5.** $Y$ 是 $X$ 的确定性函数当且仅当 $I(X;Y) = H(Y)$。

即知道 $Y$ 之后 $X$ 的信息熵期望会下降 $H(Y)$。

根据上面的定义，我们可以得到以下定理。

**定理 5** (互信息的链式法则)**.** 令 $X_1, X_2, \ldots, X_n, Y$ 是随机变量，则：

$$I(X_1, X_2, \ldots, X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_{i-1}, X_{i-2}, \ldots, X_1)$$

证明.

$$\begin{aligned} I(X_1, X_2, \ldots, X_n; Y) &= H(X_1, X_2, \ldots, X_n) - H(X_1, X_2, \ldots, X_n \mid Y) \\ &= \sum_{i=1}^{n} H(X_i \mid X_1, \ldots, X_{i-1}) - \sum_{i=1}^{n} H(X_i \mid X_1, \ldots, X_{i-1}, Y) \\ &= \sum_{i=1}^{n} \Big[ H(X_i \mid X_1, \ldots, X_{i-1}) - H(X_i \mid X_1, \ldots, X_{i-1}, Y) \Big] \\ &= \sum_{i=1}^{n} I(X_i; Y \mid X_1, \ldots, X_{i-1}) \end{aligned}$$

□

**定理 6.** 设 $X, Z$ 是两个随机变量，$Y$ 是一个关于 $X$ 的确定性函数。则 $I(X;Z) \geq I(Y;Z)$。

证明. 考虑 $I(X, Y; Z)$，根据互信息的链式法则有：

$$I(X, Y; Z) = I(X; Z) + I(Y; Z|X) = I(Y; Z) + I(X; Z|Y)$$

而由于 $Y$ 是 $X$ 的确定性函数，所以 $I(Y;Z|X) = 0$。而又 $I(X;Z|Y) \geq 0$。

于是：$I(X;Z) \geq I(Y;Z)$。 □

根据这个定理，我们可以通过 $X$ 的一个确定性函数 $Y$ 的信息熵变化量来推测 $X$ 的信息熵变化量。

利用信息熵，我们可以对一些交互题作简要的交互次数**估算**。

**例题 1** (经典问题)**.** 交互库有一个 $[1, n]$ 之间的整数 $x$。每次可以询问一个 $y$，交互库会返回 $[y \leq x]$。在尽可能少的次数内确定 $x$。

这个问题中，可将 $x$ 视作随机分布的，于是 $H(x) = \log_2 n$。每次交互库返回信息只有两种，根据定理 3，其信息熵不超过 1。

根据推论 5，我们知道我们至少需要 $\log_2 n$ 的信息才能保证推断出 $x$。

**例题 2** (洛谷 P9477 猜数)**.** 交互库有一个 $[1, 10^{18}]$ 之间的整数 $x$。每次可以询问一个 $y$，如果 $x = y$ 交互结束。否则会返回 $[y < x]$，但是有 $p$（$p < 0.5$）的概率出错。在尽可能少的次数内结束。

假设询问 $y$，交互库返回 $R$，先忽略 $x = y$ 的情况，设 $Z = [x < y]$。为了最大化询问的信息收益，我们需要最大化 $I(X;R)$，需要调整 $y$ 使得 $Z = 0, 1$ 的概率都接近 $\frac{1}{2}$，此时 $H(Z) = 1$。

不妨设交互库返回了 $R = 0$，根据贝叶斯公式，此时 $Z$ 为 $0, 1$ 的概率就分别变成了 $1-p, p$，则 $H(Z|R) = -p\log_2 p - (1-p)\log_2(1-p)$。

所以 $I(Z;R) = 1 + p\log_2 p + (1-p)\log_2(1-p) \le I(X;R)$（根据定理 6）。

所以交互次数期望大约是 $\frac{H(X)}{I(X;R)} \le \frac{\log_2 10^{18}}{1 + p\log_2 p + (1-p)\log_2(1-p)}$，在 $p = 0.49$ 的时候是 207219。我们可以使用动态开点线段树维护每个 $x$ 的概率，在线段树上二分 $y$。最终实现的期望应当比上面算出来的值略低，最终代码的表现也和我们的预测相符。

虽然我们上面估算的过程不太严谨，但终究是为我们交互次数的上界提供了一些参考。

# 2 ID3 算法

## 2.1 引入

考虑这么一类交互问题：有 $n$ 个人、$m$ 个属性，你知道每个人对应的属性。有一个隐藏的人，每次可以询问这个人的一个属性，要求在较少的询问中确定这个人。

| 编号 | 身高 | 体重 | 性别 | 年龄 |
|---|---|---|---|---|
| 1 | 高 | 胖 | 男 | 16 |
| 2 | 低 | 胖 | 男 | 16 |
| 3 | 低 | 瘦 | 男 | 16 |
| 4 | 高 | 瘦 | 男 | 17 |
| 5 | 高 | 胖 | 男 | 18 |
| 6 | 高 | 胖 | 女 | 18 |

交互的过程可以形式化的描述为：维护可能的答案集合，每次选定一个属性。按照这个属性，把可能成为答案的人分成若干类。每次询问的结果会指向其中的一个类。我们称这个选择的属性为**决策类型**。

把所有可能的情况画出来，把分出来的集合作原集合的儿子，这就构成了一棵**决策树**。

一般来说，一个决策树叶子的平均深度越小，或者最大深度越小，我们可以认为其越优秀。

一般而言，可能的决策树非常多。我们需要一些一些算法找到一个足够优秀的决策树，这样能大大减少我们在交互中的期望询问次数，或者最劣询问次数。

下图展示了一颗比较优秀的决策树。

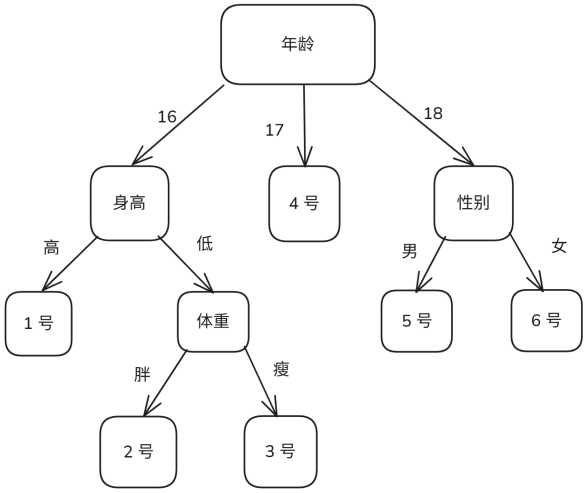



图 1: 一个较优的决策树

## 2.2 流程

观察上面的决策树，为什么我们第一次询问不去问身高，体重，或者是性别。而选择去询问年龄。

首先，我们可以把每个人看成一个数，而这个需要确定的人就是一个随机变量 $X$。我们的任务是确定这个变量，也就是我们需要降低这个变量的信息熵。

根据推论 5，降低 $X$ 的信息熵，即最大化这个询问的信息熵。

于是，构建一颗决策树。可以每次选择一个信息熵最大的询问作为决策类型。

这种算法称之为 **ID3 算法**。

**例题 3** (WC2022 猜词)**.** 交互题，给定一个词库（约 9000 个，且每个词语长度为 5），交互库有一个隐藏的词。每次可以猜一个词库的单词，如果猜测正确，则按照当前交互次数得到对应的分数，否则会告诉你反馈哪些位置字母正确，哪些位置字母存在但位置错误。要求期望得分尽可能大。多次询问，$T = 1000$。

按照 ID3 算法构建决策树，可以每个结点取信息熵前几大的询问分别计算最优期望。如果当前节点集合很小优先选择可能成为答案的。

## 2.3 局部化改良

### 2.3.1 原算法的不足之处

ID3 算法在交互的次数上比较优秀，但是时间复杂度却较大。在大部分交互题都无法实现，甚至无法获得任何部分分。

对于询问，我们可以只去随机若干个询问，取信息熵最大者。这样子我们构建出的决策树仍然较为优秀。

但是准确地计算、甚至估算询问的信息熵大概率都是困难的，因为可能的答案很多，而我们得到的信息通常比较复杂，难以整体维护。

为此，我们总结了一类在 ID3 算法基础上的改良算法，通过对目标变量的分解，能在保留部分 ID3 算法优秀性的同时，有效地优化时间复杂度。

### 2.3.2 问题概述

有 $n$ 个变量，每个变量取值范围都很小（设为 $V$）。你可以询问一些这些变量的自由组合，交互库会返回这个自由组合的一个确定性函数结果。

### 2.3.3 改良

我们不去枚举 $V^n$ 种可能的情况。而是每次只去维护一个小集合内元素所有可能情况，去随机 $Q$ 个针对这个小集合的询问，计算对于这个小集合带来的信息熵最大的询问作为当前决策。

每次做完决策，目前可能的情况可能比较少，继续做询问可能带来的信息熵就比较小了。那么我们每次可以补充还没有确定的元素进来，直到目前的可能的情况又到达了**阈值** $B$ 为止。

如果一个点已经确定（在所有可能的答案中都一致），我们可以直接将其删去。在后续针对性的答案中不去枚举它。这样子目前在答案中需要枚举的点也会很少，进一步优化了复杂度。

通过上面把所有可能情况压缩的方式，需要枚举的可能的答案类就大大减少了，在维护 ID3 算法构建决策树的优秀性的同时，也大大减少了复杂度，极大的拓展了其推广性。我们暂且称这种算法为**局部 ID3 算法**。

接下来我们将展示几个例题来显示这一算法很强的推广性。

**例题 4** (QOJ8469 Comedy's Not Omnipotent)**.** 给定长度为 $n = 10^5$ 的 01 序列 $a$。每次可以询问 $S \subset [n]$，交互库会返回 $\sum_{i\in S} a_i$。

在 $\frac{n}{2}$ 次询问内，$\sum |S|$ 不超过 $3n$ 的情况下返回 $a$。

套用上面的局部 ID3 算法即可。需要注意的是，在这个问题下如果阈值设的过大可能会导致超时，或者 $\sum |S|$ 过大，我们答案阈值要适当放小一点（大概只需要开到 40 左右）。

**例题 5** (洛谷 P14033 子集乘积，有更改)**.** 给定长度为 $n$ 的 01 序列 $a$，保证 $a_1 = 0$。每次可以询问 $S \subset [n]$，令 $x = \sum_{i\in S} a_i$，交互库会返回 $\min(x, |S| - x)$。

在 385 次询问内返回 $a$，$1 \le n \le 1000$，$1 \le T \le 10$。

在套用局部 ID3 算法时，我们能感受到这题的询问比上题略弱。单纯套用上面做法进行询问得到的信息熵会略小。

考虑到我们局部 ID3 算法一大特点是每次询问的集合比较小。如果我们有多于目前询问集合、且已经确定的 0（或者 1），我们把其全部加入这个询问集合之中，我们就把这个询问改造成了例题 4 的形式，即获取了原集合中 1 的数量。

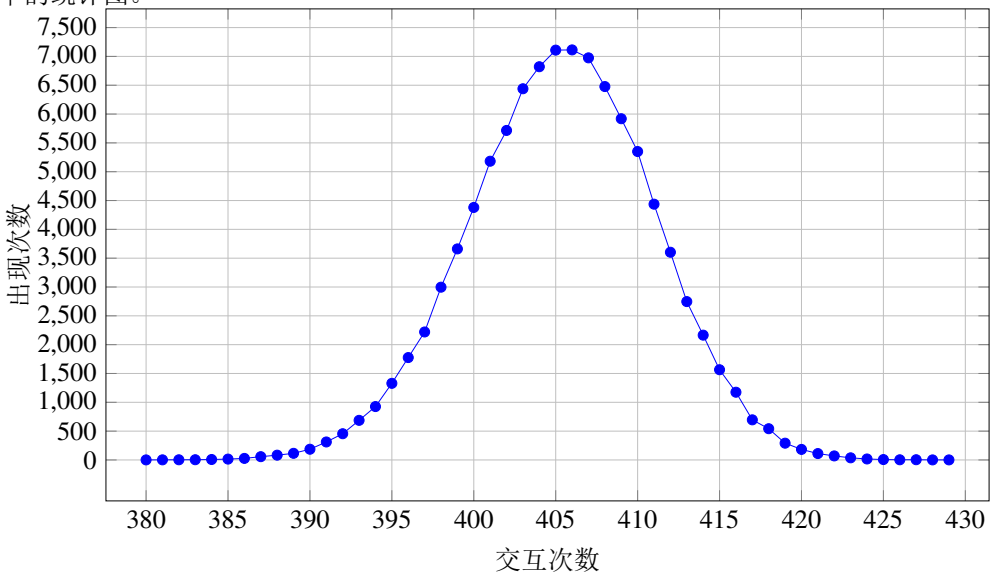
本题由于有多测，而且时限较紧，需要注意代码常数、调整阈值和随机的询问数量。必要时还可以尝试不同的随机种子。

笔者采用了阈值为 1500，每轮随机询问数量为 150 的算法能稳定通过本题。

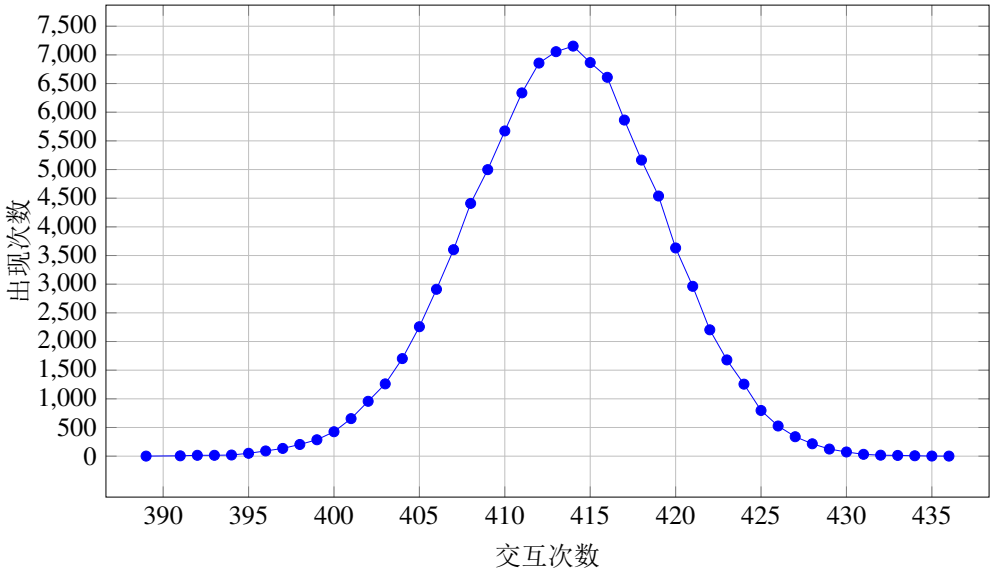
这启示我们，单纯依赖待确定的元素进行询问可能询问不是很丰富，信息熵可能较小。有些时候可以引入一些已经确定的元素来丰富我们的询问集，从而提升每个询问的信息熵。

### 2.3.4 次数分析

以例题 4 为例，$n = 1000, B = 100, Q = 20$、测试 $10^5$ 组，可以得到下面的交互次数的出现频率的统计图。




使用不同的最优询问的评判标准可能导致最终交互次数分布的不同。如：最劣情况 $X$ 剩余信息熵最小的询问作为最优询问可以得到：




可以看出使用其他的最优询问判断标准，无论是期望上还是最大值上都不如使用询问的信息熵。

接下来我们会简要分析交互次数的分布，不妨设 $n$ 远大于 $B, Q$。

除去一小段交互开始，和结束前一小段，中间维护的答案集合是比较稳定的，可以看成类似在有限图上作随机游走。尽管每次信息熵分布并不是独立随机的，我们仍可以近似地看作其独立服从于某个概率密度函数 $f(x)$。

**引理 1.** 假设 $f(x)$ 是只在 $[0, V]$ 上大于 0 的一个概率密度函数，实数 $S$ 远大于 $V$。若每次给 $S$ 减去一个服从 $f(x)$ 随机实数，并记录 $S$ 第一次 $< 0$ 的操作次数 $t$。则 $t \sim N\left(\frac{S}{\mu}, \sqrt{\frac{S\sigma^2}{\mu^3}}\right)$，其中 $\mu, \sigma$ 分别是 $f(x)$ 的均值和标准差。

证明. 对于 $n \in \mathbb{N}_+$，且 $n \approx \frac{S}{\mu}$。令 $\Phi(t)$ 为正态分布 $N(0,1)$ 的概率累积函数，我们有：

$$\begin{aligned} Pr(t \le n) &= Pr\left(S_n \ge S\right) \\ &= Pr\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \ge \frac{S - n\mu}{\sigma\sqrt{n}}\right) \\ &= \Phi\left(\frac{n\mu - S}{\sigma\sqrt{n}}\right) \\ &\approx \Phi\left(\frac{n - \frac{S}{\mu}}{\sqrt{\frac{S\sigma^2}{\mu^3}}}\right) \end{aligned}$$

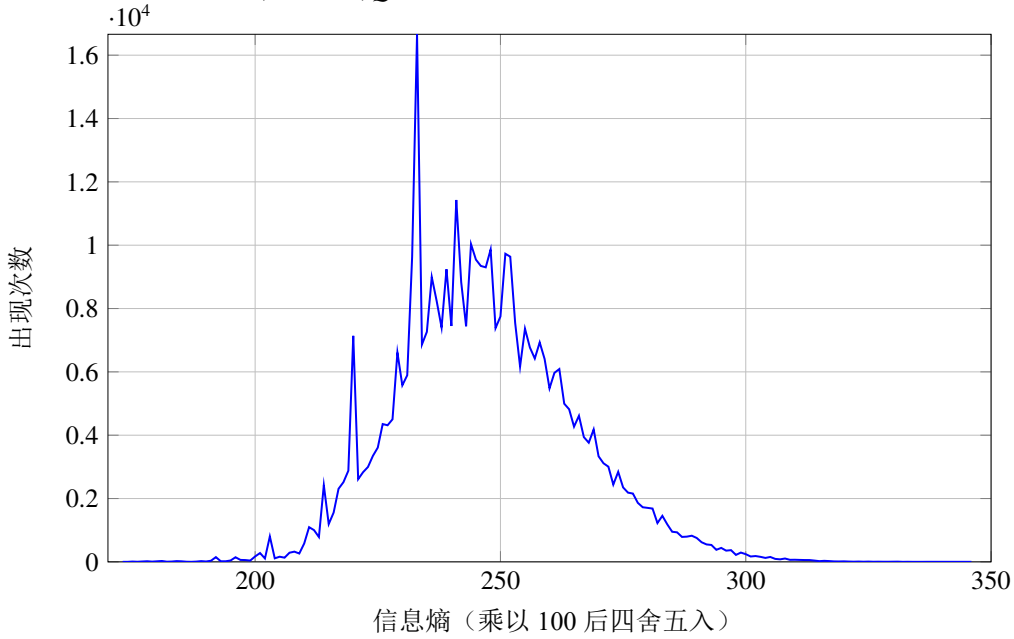
其中第三步用到中心极限定理，第四步把分母的 $n$ 估为 $\frac{S}{\mu}$。

这表明 $Pr(t \le n) \approx Pr\left(x \le \frac{n - \frac{S}{\mu}}{\sqrt{\frac{S\sigma^2}{\mu^3}}}\right)$，其中 $x \sim N(0,1)$。

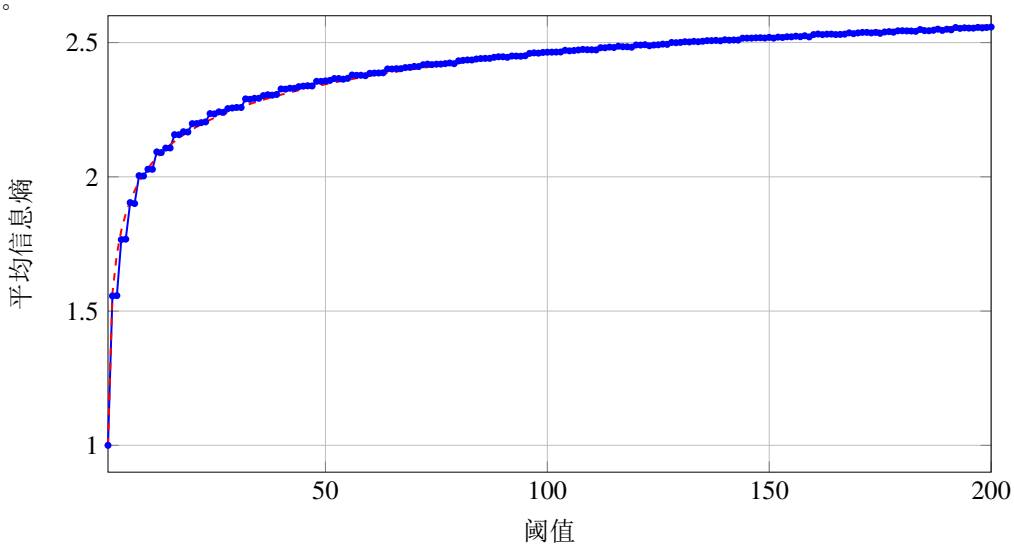
于是 $t$ 的概率概率密度函数是 $N(0,1)$ 的经过拉伸得到的，于是 $t$ 也服从于一个正态分布。解得：$t \sim N\left(\frac{S}{\mu}, \sqrt{\frac{S\sigma^2}{\mu^3}}\right)$。 □

该引理解释了为什么我们上面两个统计都呈现类似正态分布的形状。

下图展示了在 $n = 1000, B = 100, Q = 20$ 测试 $10^4$ 组，每次询问的信息熵的分布：



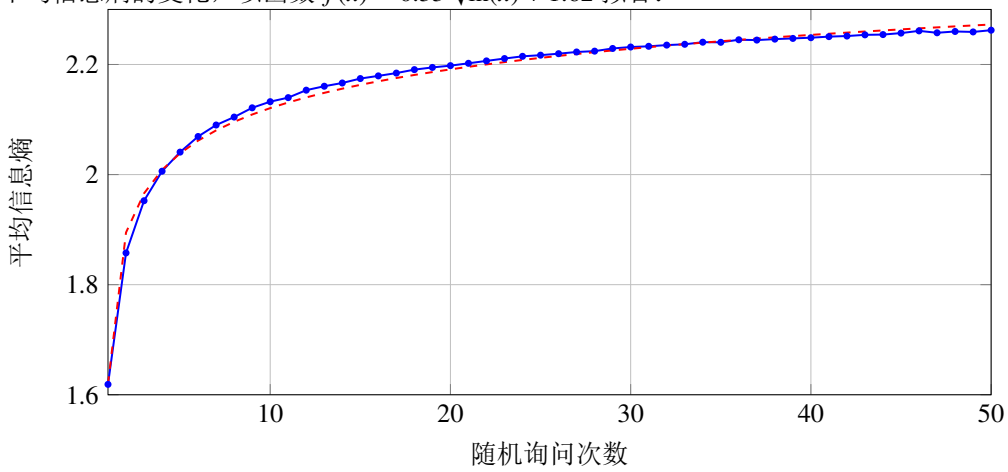

下图展示了在 $Q = 20$ 时，不同的 $B$ 平均信息熵的变化趋势，并对 $f(x) = 0.68\sqrt{\ln(x)} + 1$ 拟合。




据此我们在实际中也可以粗略地估计平均信息熵和 $\sqrt{\ln B}$ 成一次函数。

**引理 2.** 如果随机变量 $x$ 服从于 $N(0,1)$。随机选 $t$ 次，$x$ 的最大值期望记为 $M_t$。则：$\lim_{t\to\infty^+} \dfrac{M_t}{\sqrt{2\ln(t)}} = 1$。

由于随机 $Q$ 次选某个询问，在例题中所有可能询问的信息熵可以猜测为某个正态分布。于是，我们可以猜测其关于 $Q$ 的变化量也和 $\sqrt{\ln(Q)}$ 线性相关。下图是 $B = 100$ 时，随着 $Q$ 变化平均信息熵的变化，以函数 $f(x) = 0.33\sqrt{\ln(x)} + 1.62$ 拟合：




据此我们在实际中也可以粗略地估计平均信息熵和 $\sqrt{\ln Q}$ 成一次函数。

于是，在例题 4 中，我们可以粗略地估计交互次数和 $N$ 成正比，和 $\sqrt{\ln B}$, $\sqrt{\ln Q}$ 的一次函数成反比。在其他问题中使用类似的做法可能也有类似的关系。

## 3 总结

本文主要介绍了利用信息熵相关知识，简略估计算法期望交互次数的方法。并以前序知识为基础介绍了 ID3 算法以及局部性改良。

但由于篇幅原因，我们并没有介绍更多信息熵相关概念和定理。且上面提出的局部 ID3 算法实际上还有很大的优化空间，如能否把随机询问转成通过某种方法计算一个信息熵较高的询问，又或者能否够通过别的技巧维护更大范围的可能答案集合。

限于笔者的水平，无法给出局部 ID3 算法期望获得信息熵的严格计算。

希望本文能让读者对信息熵、及其延伸算法作初步了解。在未来能有更加深入的研究，在未来的信息学竞赛中有更广泛的用途。

## 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢学校的培养，教练梁泽贤对本人的指导。
感谢父母一直以来的支持。
感谢王俊霖同学为笔者提供数学理论的支持。
感谢郑煦翔、包泽韬、周康阳、陈诺同学为本文提供修改建议。
感谢各位在百忙之中阅读本文。

## 参考文献

[1] C. E. SHANNON, A Mathematical Theory of Communication, 1948

[2] Wikipedia, Entropy (information theory), https://en.wikipedia.org/wiki/Entropy_(information_theory)

[3] Milmon, 题解 P10850 [EGOI2024] Light Bulbs / 灯泡安装, https://www.luogu.com.cn/article/52f5ndpf

[4] 侯启明, 信息论在信息学竞赛中的简单应用, 国家集训队 2003 论文

[5] Wikipedia, Gaussian random walk, https://en.wikipedia.org/wiki/Random_walk

# 浅谈一类动态图上查询问题

浙江金华第一中学 赵晟昊

## 摘 要

动态图是 OI 中的一类常见问题模型，本文将讨论若干在动态图上维护信息的问题，并给出相应的在线和离线算法。

## 1 前言

动态图相关算法在信息学竞赛中应用广泛，其核心在于如何在图结构动态变化的情形下，高效维护图的各种性质并回答查询。当前 OI 领域关于动态图算法的系统性介绍仍较为缺乏，尤其是针对完全在线情形的动态图维护算法，相关资料较为零散。本文尝试对此进行梳理与总结。

本文第二节中会规定一些约定和记号。第三节中将介绍动态连通性问题的分治离线算法与基于 ET-Tree 的在线算法。第四节则会讨论动态最小生成森林的维护方法，包括分治算法与多层结构设计。第五节将会讨论动态边双连通分量的高效算法。

本文所述算法均具有较好的时间复杂度与可实现性，希望能为 OI 中的动态图问题提供有价值的参考。

## 2 定义与约定

**定义 2.1** (图)**.** 若无特殊说明，本文中讨论的所有图均为简单无向图。一个简单无向图定义为一个二元组 $G = (V, E)$，其中，$V$ 是一个非空有限集合，其元素称为顶点，$E \subseteq \{(u, v) \mid u, v \in V,\ u \neq v\}$，其元素称为边。

**定义 2.2** (生成子图)**.** 称图 $G' = (V, E')$ 是图 $G = (V, E)$ 的生成子图，当且仅当 $E' \subseteq E$。

**定义 2.3** (连通，路径)**.** 设 $G = (V, E)$ 为一个简单无向图，$u, v \in V$。若存在一个顶点序列 $u = v_0, v_1, \ldots, v_k = v$ 使得对任意 $i = 0, 1, \ldots, k-1$，都有 $(v_i, v_{i+1}) \in E$，则称顶点 $u$ 与 $v$ 在图 $G$ 连通，该序列称为一条从 $u$ 到 $v$ 的路径。

**定义 2.4** (连通块)**.** 对于图 $G = (V, E)$，若对于 $V$ 中任意两个不同的点 $u, v$，都有 $u, v$ 连通，则 $G$ 被称为连通图。一个图 $G$ 的极大连通生成子图被称为连通块。

**定义 2.5** (生成森林)**.** 称森林 $G' = (V, E')$ 是图 $G = (V, E)$ 的生成森林，当且仅当 $G'$ 是 $G$ 的生成子图，且对于任意 $(u, v) \in E$，满足 $u, v$ 在 $G'$ 上连通。

**定义 2.6** (最小生成森林)**.** 对于图 $G = (V, E)$ 和权值函数 $w : E \to \mathbb{R}$，$G$ 关于 $w$ 的最小生成森林定义为 $G$ 的一个生成森林 $G' = (V, E')$ 使得 $\sum_{e \in E'} w(e)$ 最小。

**定义 2.7** (边双连通分量)**.** 设 $G = (V, E)$ 为一个简单无向图，$u, v \in V$，$u \neq v$，若对于任意一条边 $e \in E$，删除 $e$ 后 $u$ 与 $v$ 仍然连通，则称 $u, v$ 是边双连通的。图 $G$ 的边双连通分量是其极大边双连通生成子图。

**定义 2.8** (图的运算)**.** 对于点集 $V$ 以及图 $F = (V, E_1), G = (V, E_2)$：

- $F$ 和 $G$ 的交定义为 $(V, E_1 \cap E_2)$，记作 $F \cap G$。
- $F$ 和 $G$ 的并定义为 $(V, E_1 \cup E_2)$，记作 $F \cup G$。
- $F$ 和 $G$ 的差定义为 $(V, E_1 \setminus E_2)$，记作 $F \setminus G$。

若无特殊说明，对于图 $G = (V, E)$，我们约定 $n = |V|$。

# 3 动态连通性

## 3.1 问题描述

给定图 $G = (V, E)$，初始 $E$ 为空。有 $m$ 个操作，每个操作形如以下三种操作之一：

**Insert**$(u, v)$ 对于 $V$ 中的不同节点 $u, v$，将边 $(u, v)$ 加入 $E$ 中。不妨假设操作前 $(u, v) \notin E$。

**Delete**$(u, v)$ 将边 $(u, v)$ 从 $E$ 中删除。

**Query**$(u, v)$ 查询 $u$ 和 $v$ 在 $G$ 中是否连通。

上述问题可分为两种情形：

**离线情形** 算法预先获得所有操作，可以一次性处理整个操作序列。

**在线情形** 操作逐个到达，算法必须立即处理当前操作并返回结果，无法预知后续操作。

若无特殊说明，接下来的问题也将讨论上述两种情形的算法。

## 3.2 离线情形

由于只有操作序列涉及的点有用，不妨假设 $|V| = O(m)$。

为了方便叙述，不妨假设所有操作结束后 $E$ 为空。可以视为在 $m$ 次操作后对所有 $e \in E$ 执行了 **Delete**($e$)，此时总操作次数 $m' = \Theta(m)$。

记第 $i$ 次操作的编号为 $i$。对于操作中出现过的所有边 $e$，设 $l(e)$ 为加入 $e$ 对应操作的编号，$r(e)$ 为删除 $e$ 对应操作的编号，则可以看成 $e$ 会对编号区间 $[l(e), r(e)]$ 中的所有询问产生贡献。

考虑分治。设 **Solve**($l$, $r$, $V$, $E$, $Q$) 表示当前问题中点集为 $V$，边集为 $E$，**Query** 操作构成的集合为 $Q$，其中 $Q$ 中的每个元素为三元组 $(p, u, v)$，表示其操作编号为 $p$，询问的点为 $u, v$，需要满足 $p \in [l, r]$。

若执行 **Solve**($l$, $r$, $V$, $E$, $Q$)：

建立一个图 $G' = (V, E')$，其中 $E' = \{e \mid e \in E, [l, r] \subset [l(e), r(e)]\}$。为 $V$ 中所有点按其在 $G'$ 中的连通块重标号，即对于所有 $p \in V$，取 $f(p)$ 满足 $f(p_1) = f(p_2)$ 当且仅当 $p_1, p_2$ 连通。

- 若 $l = r$：点 $u, v$ 连通当且仅当 $f(u) = f(v)$，回答 $Q$ 中询问。
- 否则：设 $mid = \left\lfloor \frac{l+r}{2} \right\rfloor$。设 $E_2 = \{(f(u), f(v)) \mid (u, v) \in E \setminus E', f(u) \neq f(v)\}$，$Q' = \{(p, f(u), f(v)) \mid (p, u, v) \in Q\}$，$V'$ 为 $E_2$ 和 $Q'$ 中所有端点构成的集合。令 $E_l = \{e \mid e \in E \setminus E', [l(e), r(e)] \cap [l, mid] \neq \varnothing\}$，$E_r = \{e \mid e \in E \setminus E', [l(e), r(e)] \cap [mid + 1, r] \neq \varnothing\}$，$Q_l = \{(p, u, v) \mid (p, u, v) \in Q', p \leqslant mid\}$，$Q_r = \{(p, u, v) \mid (p, u, v) \in Q', mid < p\}$，执行 **Solve**($l$, $mid$, $V'$, $E_l$, $Q_l$)，**Solve**($mid + 1$, $r$, $V'$, $E_r$, $Q_r$)，递归子问题。

执行 **Solve**($1$, $m$, $V$, $E_0$, $Q$) 即可获得所有 **Query** 操作对应的结果，其中 $E_0$ 为操作中出现过的所有边构成的集合，$Q$ 为所有询问构成的集合。

**引理 3.1.** $|E \setminus E'| \leqslant r - l + 1$。

证明. 对于所有 $e \in E \setminus E'$，$e$ 必然在 $[l, r]$ 中被操作过，得证。 □

**引理 3.2.** 不考虑向下递归的部分，执行 ***Solve***(*l, r, V, E, Q*) 的时间复杂度为 $O(r - l + 1)$。

证明. 除递归外，**Solve**($l$, $r$, $V$, $E$, $Q$) 的时间复杂度为 $O(r - l + 1 + |V| + |E| + |Q|)$，其中 $|Q| \leqslant r - l + 1$。由引理 3.1，$|E \setminus E'| = O(r - l + 1)$，故 $|E_l|$，$|E_r|$ 均为 $O(r - l + 1)$，可得 $|E| = O(r - l + 1)$。$|V| = O(|E| + |Q|) = O(r - l + 1)$。总时间复杂度为 $O(r - l + 1)$。 □

**引理 3.3.** 执行 ***Solve***(*1, m, V, E, Q*) 的时间复杂度为 $O(m \log m)$。

证明. 由引理 3.2，总时间复杂度为 $O(T(m))$，其中 $T(m) = 2T(\frac{m}{2}) + O(m)$，可得 $T(m) = O(m \log m)$。 □

由引理 3.3，我们得到了一个时间复杂度为 $O(m \log m)$ 的算法。

## 3.3 在线情形

### 3.3.1 算法介绍

接下来，我们将介绍一个算法 [1]，其 **Insert** 和 **Delete** 操作的时间复杂度为均摊 $O(\log^2 n)$，而 **Query** 操作的时间复杂度为最坏 $O(\log n)$。

对于所有操作中出现过的边 $e$，为 $e$ 设置一个自然数等级 $\ell(e)$，其中 $\ell_{\max}$ 表示过程中所有 $\ell(e)$ 的最大值。

设 $F$ 为 $G$ 以 $\ell$ 为权值的最大生成森林。

对于 $0 \leqslant i \leqslant \ell_{\max}$，设 $G_i = (V, \{e \mid e \in E, \ell(e) \geqslant i\})$，$F_i = F \cap G_i$。

称 $e \in E$ 为树边，当且仅当 $e \in F$。$e$ 为非树边当且仅当 $e$ 不是树边。

**引理 3.4.** 若 $e = (u, v) \in E$ 是非树边，则对于 $F$ 上 $u$ 到 $v$ 树上路径的所有边 $e'$，有 $\ell(e) \leqslant \ell(e')$。

证明. 由最大生成森林的性质可得。 □

**引理 3.5.** 对于 $0 \leqslant i \leqslant \ell_{\max}$，$F_i$ 是 $G_i$ 的一个生成森林。

证明. 对于非树边 $(u, v) \in G_i$，由引理 3.4，$F$ 上 $u$ 到 $v$ 树上路径的所有边 $e'$ 都在 $F_i$ 中，故 $u, v$ 在 $F_i$ 连通。由生成森林的定义得证。 □

对于每个操作：

**Insert**$(u, v)$ 将 $e = (u, v)$ 加入 $E$ 中，并设置 $\ell(e) = 0$。若 $u, v$ 在 $G$ 中不连通，则将 $e$ 加入 $F$。

**Query**$(u, v)$ 等价于查询 $u, v$ 是否在 $F$ 连通。

**Delete**$(e)$ 将 $e$ 从 $E$ 中删去。若 $e$ 是非树边，则什么也不做。否则，将 $e$ 从 $F$ 中删去，设 $u, v$ 所在连通块分别为 $T_u, T_v$，为了维护 $F$，需要找到一条连接 $T_u, T_v$ 的 $\ell$ 最大的边。记 **Find**$(u, v, i)$ 表示找到一条 $\ell(e) \leqslant i$ 且 $\ell(e)$ 最大的非树边 $e$ 加入 $F$，满足加入后 $u, v$ 连通。由引理 3.4 ，调用 **Find**$(u, v, \ell(e))$ 即可完成维护。

**Find**$(u, v, i)$ 记 $F_i$ 上 $u, v$ 所在连通块分别为 $T_u, T_v$，不妨假设 $|T_u| \leqslant |T_v|$。先枚举 $T_u$ 上所有 $\ell$ 为 $i$ 的树边 $e$，令 $\ell(e) \leftarrow \ell(e) + 1$。再枚举有一个端点在 $T_u$ 的 $\ell$ 为 $i$ 的非树边 $e$，由引理 3.5 得 $e$ 另一个端点在 $T_u \cup T_v$ 中。若 $e$ 另一个端点在 $T_v$，则将 $e$ 加入 $F$ 并终止该次 **Find** 过程，否则令 $\ell(e) \leftarrow \ell(e) + 1$。若没有找到需要的边，且 $i > 0$，执行 **Find**$(u, v, i - 1)$。

对于 $0 \leqslant i \leqslant \ell_{\max}$，需要在 $F_i$ 上支持加边，删边，以及在连通块内找到一个连有 $\ell$ 为 $i$ 的边的点。使用 ET-Tree 维护 [2] ，可以做到单次操作时间复杂度 $O(\log n)$。

### 3.3.2 时间复杂度分析

上文所述的 **Insert** 和 **Query** 只会进行 $O(1)$ 次对 ET-Tree 的操作，而 **Delete** 会进行 $O(c+\log n)$ 次 ET-Tree 操作，其中 $c$ 为所有边 $\ell$ 的改变次数之和。

**引理 3.6.** 对于 $0 \leqslant i \leqslant \ell_{\max}$，$F_i$ 中每个连通块的大小不超过 $\left\lfloor \frac{n}{2^i} \right\rfloor$。

证明. 考虑归纳。

- 当 $i = 0$ 时，$F_0$ 的每个连通块大小不会超过 $n$。
- 当 $i > 0$ 时，假设任意时刻 $F_{i-1}$ 中连通块大小不超过 $\left\lfloor \frac{n}{2^{i-1}} \right\rfloor$，只有 **Find**$(u, v, i-1)$ 操作会往 $F_i$ 中加边，此时 $|T_u| \leqslant \frac{|T_u|+|T_v|}{2} \leqslant \left\lfloor \frac{n}{2^i} \right\rfloor$。操作过程中只会添加两个端点都在 $T_u$ 内的边，故 $F_i$ 中新增的连通块大小均不超过 $|T_u|$，则任意时刻 $F_i$ 中连通块大小不超过 $\left\lfloor \frac{n}{2^i} \right\rfloor$。

□

由引理 3.6 可得 $\ell_{\max} \leqslant \log n$。

注意到对于任意一条边 $e \in E$，$\ell(e)$ 在操作过程中单调不降，故 $\ell(e)$ 改变次数不超过 $\ell_{\max} = O(\log n)$，则 **Delete** 操作的时间复杂度为均摊 $O(\log^2 n)$。

事实上，由于 **Query** 操作只关心 $F$ 上两点是否连通，我们可以为 $F$ 额外建立一个数据结构，在这上面只有均摊 $O(1)$ 次加边或删边和 $O(1)$ 次查询连通性。使用 $\log n$ 叉平衡树维护 ET-Tree，可以在 $O(\frac{\log^2 n}{\log\log n})$ 的时间内维护修改，而在 $O(\frac{\log n}{\log\log n})$ 的时间内维护查询。

经过这样的优化后，**Insert** 操作和 **Delete** 操作的时间复杂度仍然为均摊 $O(\log^2 n)$，但 **Query** 操作的时间复杂度降为严格 $O(\frac{\log n}{\log\log n})$。

## 4 动态最小生成森林

### 4.1 问题描述

给定图 $G = (V, E)$ 和权值函数 $w : V \times V \to \mathbb{R}$，初始 $E$ 为空。为了方便描述，不妨假设 $w$ 为单射，此时 $G$ 最小生成森林唯一。有 $m$ 个操作，每个操作形如以下三种操作之一：

**Insert**$(u, v)$ 对于 $V$ 中的不同节点 $u, v$，将边 $(u, v)$ 加入 $E$ 中。不妨假设操作前 $(u, v) \notin E$。

**Delete**$(u, v)$ 将边 $(u, v)$ 从 $E$ 中删除。

**Query**$(u, v)$ 设 $G'$ 为 $G$ 的最小生成森林，查询是否满足 $(u, v) \in G'$。

事实上，**Query** 可以换成任意对 $G'$ 的查询，如查询最小生成森林的边权之和，或着 $u$ 到 $v$ 所有简单路径中可能经过的权值最小的边，等等。

## 4.2 离线情形

考虑使用 3.2 的思路，仅对 **Solve** 函数进行修改。若执行 **Solve**$(l, r, V, E, Q)$：

令 $E_0 = \{e \mid e \in E, [l, r] \subset [l(e), r(e)]\}, E_1 = E \setminus E_0$，则 $E_0$ 中的边会对 $[l, r]$ 中的所有询问产生贡献。

对于 $E_0$ 中的所有边 $e = (u, v)$，记 $G_e = (V, \{e' | e' \in E_0, w(e') < w(e)\})$，若 $u, v$ 在 $G_e$ 中连通，则 $e$ 可以直接删去，记这样的边构成的集合为 $E_2$。记 $G'_e = (V, E_1 \cup \{e' | e' \in E_0, w(e') < w(e)\})$，若 $u, v$ 在 $G'_e$ 上不连通，则 $e$ 总是在最小生成树中，记这样的边构成的集合为 $E_3$。这两部分都可以使用并查集维护，时间复杂度 $O(|E_0|\alpha(n))$。

记 $G' = (V, E_3)$，可以对 $V$ 按照 $G'$ 中连通块重标号得到 $f(p)$。对于所有 $(p, u, v) \in Q$ 满足 $(u, v) \in E_3$，可得 $(u, v)$ 在最小生成森林中，回答询问并将 $(p, u, v)$ 从 $Q$ 中删除。

若 $l \neq r$：令 $E \leftarrow E \setminus (E_2 \cup E_3)$。使用 3.2 中的方法来得到 $mid, V', E_l, E_r, Q_l, Q_r$，执行 **Solve**$(l, mid, V', E_l, Q_l)$，**Solve**$(mid + 1, r, V', E_r, Q_r)$，递归子问题。

**引理 4.1.** $|E \setminus (E_2 \cup E_3)| = O(r - l + 1)$。

证明. $G(V, E_0 \setminus E_2)$ 形成一个森林，而 $E_0 \setminus (E_2 \cup E_3)$ 中所有边的端点都在 $E_1$ 的端点集合中，此时有 $|E_0 \setminus (E_2 \cup E_3)| < |E_1|$，由引理 3.1 的结论，$|E_0 \setminus (E_2 \cup E_3)| < |E_1| = O(r - l + 1)$，$|E| = |E_0 \setminus (E_2 \cup E_3)| + |E_1| = O(r - l + 1)$。 □

由引理 3.3 和 4.1，可得总时间复杂度为 $O(m \log m\alpha(n))$。

## 4.3 只考虑删除操作的在线情形

即给定初始的 $G$，且不执行 **Insert** 操作。

使用 3.3.1 所述结构，保持 $\ell$ 以及其他定义和操作不变，只对 **Find** 做一定的修改。对于初始的 $G$，令其中所有边 $e$ 的 $\ell(e) = 0$，$F$ 为 $G$ 关于 $w$ 的最小生成森林。

**Find**$(u, v, i)$ 记 $F_i$ 上 $u, v$ 所在连通块分别为 $T_u, T_v$，不妨假设 $|T_u| \leqslant |T_v|$。枚举 $T_u$ 上所有 $\ell$ 为 $i$ 的树边 $e$，令 $\ell(e) \leftarrow \ell(e) + 1$。每次取出有一个端点在 $T_u$ 的 $\ell$ 为 $i$ 且 $w$ 最小的非树边 $e$，若 $e$ 另一个端点在 $T_v$，则将 $e$ 加入 $F$ 并退出，否则令 $\ell(e) \leftarrow \ell(e) + 1$。若没有找到需要的边，且 $i > 0$，执行 **Find**$(u, v, i - 1)$。

**引理 4.2.** 对于两条边 $e_1, e_2$ 满足存在 $G$ 中的一个环同时包含 $e_1, e_2$，若 $w(e_1) < w(e_2)$，则 $\ell(e_1) \geqslant \ell(e_2)$。

**引理 4.3.** 在每个操作之后，$F$ 是 $G$ 以 $w$ 为权值的的最小生成森林。

证明. 我们将同时证明引理 4.2 和 4.3 。所有操作开始前引理 4.2 和 4.3 显然成立，只需要证明若删除一条边之前满足这两个引理，则删除后仍然满足。

只有 **Find** 操作会修改 $\ell$。对于其中对树边的 $\ell$ 修改，操作前 $F$ 为最小生成树，故不会对引理 4.2 产生影响。对于非树边 $e$ 的 $\ell$ 的修改，其为 $F_{\ell(e)}$ 中所在连通块的最小边，故也不会对引理 4.2 产生影响。

**Find** 实际上找到的是 $T_u$ 和 $T_v$ 之间 $\ell$ 最大的边中 $w$ 最小的边，由操作前引理 4.2 成立，可得其为所有 $\ell$ 中 $w$ 最小的边，故操作后 $F$ 最小生成森林。 □

由此，任何对最小生成森林的查询即为对 $F$ 的查询，可以使用 ET-Tree 直接维护。3.3.2 中所述分析仍然适用，故时间复杂度为 $O(m\log^2 n)$。

## 4.4 完全在线情形

### 4.4.1 算法介绍

记 $L = \log m$，维护 $A_0, A_1, \ldots, A_L$，其中 $A_i$ 为一个支持删除操作的数据结构。对于第 $i(0 \leqslant i \leqslant L)$ 个结构，记其维护的图为 $G_i$，$G_i$ 的最小生成森林为 $F_i$。记对该数据结构单次操作的时间复杂度为 $D(n)$。

将全局的图记为 $G$，其最小生成森林记为 $F$，$F \setminus G$ 中的边称为全局非树边。维护使得所有 $G_i$ 均为 $G$ 的生成子图，且 $F \subseteq \bigcup_{i=0}^{L} F_i$。更进一步，对于每条全局非树边 $e$，维护使得 $e$ 在恰好一个结构中作为非树边。

对于每个操作：

**Insert**$(u, v)$ 记 $e = (u, v)$。若 $u, v$ 在 $F$ 中不连通，则将 $e$ 加入 $F$。否则，求出 $F$ 中 $u$ 到 $v$ 路径上 $w$ 最大的边 $e'$，若 $w(e) < w(e')$ 则从 $F$ 中删除 $e'$ 并加入 $e$，然后执行 **Update**$(\{e'\})$，否则执行 **Update**$(\{e\})$。

**Delete**$(u, v)$ 从 $G_0, G_1, \ldots, G_L$ 中删除 $(u, v)$，将得到的替换边的集合记为 $S$。若 $(u, v) \in F$ 则将 $e(u, v)$ 从 $F$ 删除，此时若 $S \neq \varnothing$，将 $S$ 中 $w$ 最小的边加入 $F$。执行 **Update**$(S)$。

**Query** 对 $F$ 进行对应的查询即可。

**Update**$(S)$ 含义为将 $S$ 中的边作为非树边加入 $G_0, G_1, \ldots, G_L$，并保持其性质。设 $i$ 初始为 0，若 $|S \cup (G_i \setminus F_i)| > 2^i$，则令 $S \leftarrow S \cup (G_i \setminus F_i)$，然后令 $i \leftarrow i + 1$ 并重复上述判断；否则，用 $S \cup (G_i \setminus F_i) \cup F$ 作为边集初始化 $A_i$。

可以看到，**Delete** 操作中将所有成为树边的边进行 **Update**，故每条全局非树边仍然在至少一个 $A_i$ 中作为非树边。

**引理 4.4.** ***Delete*** 操作中的 $S$ 包含了跨过 $u, v$ 的 $w$ 最小的全局非树边。

证明. 记执行的操作为 **Delete**$(e)$，$e=(u,v)$，跨过 $u,v$ 的 $w$ 最小的全局非树边为 $e'=(u',v')$。设 $e'$ 在 $A_i$ 中作为非树边，若 $e\notin G_i$，考虑 $F_i$ 中 $u'$ 到 $v'$ 的路径，其中必然包含跨过 $u,v$ 的边，则这条边的 $w$ 一定比 $w(e')$ 小，与 $w(e')$ 最小矛盾。故 $e\in G_i$，则 $e$ 为非树边，可以正确找到 $e'$。 □

由引理 4.4 ，可得 $F$ 被正确维护。由此可得该算法的正确性。

在 **Update** 操作中，直接使用 $S\cup(G_i\setminus F_i)\cup F$ 作为边集来初始化只删除结构，会使单次操作时间复杂度达到 $\Omega(n)$。由于我们只关注删除一条树边后可能成为替换边的边，只需考虑 $S\cup(G_i\setminus F_i)$ 中所有端点在 $F$ 上的虚树。该虚树的点数和边数均为 $O(2^i)$，因此初始化 $A_i$ 的时间复杂度为 $O(2^iD(n))$。

为了求出一个点集 $S$ 在 $F$ 上的虚树，需要在 $F$ 上查询 DFS 序和最近公共祖先，可以分别用 ET-Tree 和 Link/Cut Tree 维护。

### 4.4.2 时间复杂度分析

对于每条全局非树边 $e$，设其势能为 $\Phi(e)$ 为 $L$ 减去其作为非树边所在的 $A$ 的编号。在 **Update**$(S)$ 中，会使 $S$ 中的边成为非树边，至少会使势能总和增加 $O(|S|L)$。每次操作中 $|S|\leqslant L$，故单次操作势能总和增加量不超过 $L^2$。

设 **Update** 操作带来的势能总和减少量为 $d$，可得 $2d>2^i$，故 **Update** 的时间复杂度为 $O(dD(n))$。将这部分的时间复杂度视为由势能支付，可得单次操作的时间复杂度为 $O(L^2D(n))$。由 4.3 ，取 $D(n)=O(\log^2 n)$，得单次操作的均摊时间复杂度为 $O(\log^2 n\log^2 m)$。

## 5 动态边双连通性

### 5.1 问题描述

给定图 $G=(V,E)$，初始 $E$ 为空。有 $m$ 个操作，每个操作形如以下三种操作之一：

**Insert**$(u,v)$ 将边 $(u,v)$ 加入 $E$ 中。

**Delete**$(u,v)$ 将边 $(u,v)$ 从 $E$ 中删除。

**Query**$(u,v)$ 查询 $u,v$ 是否边双连通。

和 4.1 一样，该结构也可以查询割边数量。

## 5.2 离线情形

使用和 3.2 类似的思路，若执行 **Solve**$(l, r, V, E, Q)$：

设 $E_0 = \{e \mid e \in E, [l, r] \subset [l(e), r(e)]\}$，$G_0 = (V, E_0)$，由边双连通性是等价关系，可以对所有点按照边双连通性重标号，记为 $f(p)$。

设 $V_1 = \{f(p) \mid p \in V\}$，$E_1 = \{(f(u), f(v)) \mid (u, v) \in E_0, f(u) \neq f(v)\}$，$G_1 = (V_1, E_1)$，则 $G_1$ 是一个森林。

- 若 $l = r$：$u, v$ 边双连通当且仅当 $f(u) = f(v)$，以此回答 $Q$ 中询问。
- 否则：使用 3.2 中的过程来得到 $mid, V', E_l, E_r, Q_l, Q_r$，求出 $V$ 在 $G_0$ 中的虚树，记其边集为 $E_2$。执行 **Solve**$(l, mid, V', E_l \cup E_2, Q_l)$，**Solve**$(mid + 1, r, V', E_r \cup E_2, Q_r)$，递归子问题。

由虚树的性质，$|E_2| \leqslant 2|V'|$，故 $|E_2| = O(r - l + 1)$，则 $|E| = O(r - l + 1)$。由引理 3.3，可得总时间复杂度为 $O(m \log m)$。

## 5.3 在线情形

记 $F$ 为 $G$ 的其中一个生成森林。对于所有非树边 $e$，设其等级 $\ell(e)$ 为一个自然数。对于树边，设 $c(e)$ 为所有覆盖 $e$ 的非树边的 $\ell$ 的最大值，特别的，若不存在这样的非树边，令 $c(e) = -1$。对于路径 $P$，$c(P)$ 定义为 $P$ 中所有边的 $c$ 的最小值。

记 $G_i$ 为 $(V, \{e \mid \ell(e) \geqslant i\} \cup F)$。为了保证复杂度，需要维护 $F$ 和 $\ell$ 使得 $G_i$ 中每个边双连通分量的大小不超过 $\left\lceil \frac{n}{2^i} \right\rceil$，因此所有 $\ell(e) \leqslant \ell_{\max} = \lceil \log n \rceil$。

**Query**$(u, v)$ 设 $u$ 到 $v$ 的路径为 $P$，则 $u, v$ 边双连通当且仅当 $c(P) \geqslant 0$。

**Cover**$(u, v, i)$ 对于 $u$ 到 $v$ 在 $F$ 路径上的所有边 $e$，令 $\ell(e) \leftarrow \max(\ell(e), i)$。

**Insert**$(u, v)$ 记 $e = (u, v)$，先将 $e$ 加入 $G$。若 $F$ 中 $u, v$ 不连通，则将 $e$ 加入 $F$ 并令 $c(e) = -1$；否则，令 $\ell(e) = 0$，并执行 **Cover**$(u, v, 0)$。

**Delete**$(e)$ 设 $e = (u, v)$。若 $e$ 为树边，记覆盖 $e$ 的 $\ell$ 最大的非树边为 $e'$，将 $e$ 设为非树边并将 $e'$ 设为树边，令 $\ell(e) = \ell(e'), c(e') = -1$，问题转化为删除非树边。删除 $(u, v)$，执行 **Uncover**$(u, v, i)$，然后执行 **Recover**$(u, v, i)$。

**Uncover**$(u, v, i)$ 对于 $u, v$ 路径上的所有边 $e$，若 $c(e) \leqslant i$，令 $c(e) \leftarrow -1$。

**Recover**$(u, v, i)$ 含义为对于 $u, v$ 路径上所有边 $e_1$ 满足 $c(e_1) = -1$，尝试将 $c(e_1)$ 调整为正确的值，已知 $c(e_1) \leqslant i$。只需要计算 $\ell$ 为 $i$ 的非树边对 $u, v$ 路径的贡献，然后执行

**Recover**$(u, v, i-1)$。对于与 $u, v$ 路径有交的 $\ell$ 为 $i$ 的非树边 $(x, y)$，记 $x$ 到 $u, v$ 路径上最近的点为 $w$，则 $x$ 到 $w$ 路径上的树边 $e_1$ 都满足 $c(e_1) \geqslant i$。从 $u$ 到 $v$ 依次遍历 $w$，找到所有 $\ell$ 为 $i$ 的非树边 $e = (x, y)$，满足 $x$ 到 $w$ 只经过 $c(e_1) \geqslant i$ 的树边 $e_1$。若将 $\ell(e)$ 提升到 $i+1$ 仍能满足 $G_{i+1}$ 的所有边双连通分量大小不超过 $\left\lceil \frac{n}{2^{i+1}} \right\rceil$，就令 $\ell(e) \leftarrow i+1$ 并执行 **Cover**$(x, y, i+1)$，否则执行 **Cover**$(x, y, i)$ 并退出。若第一次遍历中途退出，则改为从 $v$ 到 $u$ 的方向再次遍历 $w$ 并执行相同过程。

在 **Delete** 操作中的 **Uncover** 操作之前，若 $e$ 为树边，记 $r = \ell(e')$，可以发现在 $G_{r+1}, G_{r+2}, \ldots, G_{\ell_{\max}}$ 中，$e$ 均为割边，用 $e'$ 替换 $e$ 作为树边后，$e'$ 也为割边，而在 $G_0, G_1, \ldots, G_r$ 中，$e, e'$ 都出现了，故替换操作不会对 $G_0, G_1, \ldots, G_{\ell_{\max}}$ 的双连通性产生任何影响。由此，无论 $e$ 是不是树边，只需要修复 $u, v$ 路径上所有边的 $c$ 即可，这将由 **Recover** 实现。

**引理 5.1.** ***Recover(u, v, i)*** 操作正确维护了所有 $u, v$ 路径上所有边的 $c$。

证明. 若有一边枚举 $w$ 的过程没有在中途退出，显然 $c$ 将会被正确维护。如果两边都在中途退出，设两条非树边分别为 $(x_1, y_1), (x_2, y_2)$，若将两条边的 $\ell$ 提升到 $i+1$ 时对应的边双连通分量不交，由于 $c(x_1 \cdots w) \geqslant i$，故 $x_1, u, v$ 在 **Recover** 操作前在 $G_i$ 边双连通，同理可得 $x_1, y_1, x_2, y_2$ 在 **Recover** 操作前在 $G_i$ 边双连通，则该边双连通分量的大小超过 $2\left\lceil \frac{n}{2^{i+1}} \right\rceil \geqslant \left\lceil \frac{n}{2^i} \right\rceil$，矛盾。故 $x_1, y_1, x_2, y_2$ 操作后能通过 $c(e_1) \geqslant i$ 的树边 $e_1$ 连通，此时两边对应的 $w$ 之间的边的 $c$ 已被设置为正确的值，即已完整处理整条 $u$ 到 $v$ 的路径。 □

由此可得该算法的正确性。

使用 Toptree [3] 维护 $F$。对于每个簇 $C$，记其界点为 $a, b$，$a$ 到 $b$ 路径记为 $\pi(C)$。记 $I_{C,v,i,j}$ 为 $C$ 内的点构成的集合，其中对于每个点 $x$，设 $x$ 到 $v$ 路径为 $P$，要求满足 $c(P \cap \pi(C)) \geqslant i, c(P \setminus \pi(C)) \geqslant j$。维护 $size_{C,v,i,j} = |I_{C,v,i,j}|$，$incident_{C,v,i,j}$ 为以 $I_{C,v,i,j}$ 中的点为端点且 $\ell$ 为 $j$ 的边的个数。

对于 **Cover** 操作和 **Uncover** 操作，对边维护懒标记即可维护。对于 **Recover** 操作，由于不需要找到的非树边与 $u, v$ 路径相交，取出以 $u, v$ 路径对应的簇并在上面二分即可。

在 Toptree 中，每个簇需维护的信息量为 $\ell_{\max}^2 = O(\log^2 n)$，因此单次 Toptree 操作的时间复杂度为 $O(\log^3 n)$。每次 **Insert** 或 **Delete** 操作会引发 $O(\log n)$ 次 Toptree 操作；此外，一条边的等级 $\ell$ 每提升一次，也会触发一次 Toptree 操作，而该等级最多提升 $O(\log n)$ 次。因此，单次修改操作的均摊总时间复杂度为 $O(\log^4 n)$。

## 6 总结

本文围绕动态图，介绍了动态图上查询连通性，最小生成森林，以及边双连通性等问题。对于每个问题，分别介绍了离线情形和在线情形的算法，所有算法时间复杂度均为 $\widetilde{O}(m)$。其

中离线算法中均采用分治思想，通过时间分治，构建递归层次结构；在线算法中均采用均摊思想，对于每条边定义等级来限制访问边的次数，并用连通块大小控制等级的上界。

关于动态图上在线查询，学术界还有许多时间复杂度更优但也更加复杂的算法。受篇幅所限，本文未能逐一介绍。希望本文能起到抛砖引玉的作用，激发读者对动态图算法的兴趣，并期待未来在该领域出现更多深入的研究与成果。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢许庭强学长、张君游同学、黄建恒同学与我讨论以及给予的启发。
感谢黄蔚尧学长为本文刊误。
感谢父母对我的陪伴与支持。

## 参考文献

[1] Jacob Holm, Kristian de Lichtenberg, Mikkel Thorup, Poly-logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity, 2001

[2] M. R. Henzinger, V. King , T. Warnow, Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology, 1999

[3] Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, Mikkel Thorup, Maintaining Information in Fully-Dynamic Trees with Top Trees, 2003

[4] Christian Wulff-Nilsen, Faster Deterministic Fully-Dynamic Graph Connectivity, 2013

[5] Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, Mikkel Thorup, Fully Dynamic Connectivity in $O(\log n(\log\log n)^2)$ Amortized Expected Time, 2016

[6] David Eppstein, Zvi Galil, Giuseppe F. Italiano, Amnon Nissenzweig, Sparsification—a technique for speeding up dynamic graph algorithms, 1997

[7] Bruce M. Kapron, Valerie King, Ben Mountjoy, Dynamic Graph Connectivity in Polylogarithmic Worst Case Time, 2013

[8] David Gibb, Bruce Kapron, Valerie King, Nolan Thorn, Dynamic graph connectivity with improved worst case update time and sublinear space, 2015

# 关于信息学竞赛中的一类非传统问题及其解题方法

北京师范大学附属实验中学 周裕杭

## 摘 要

在信息学竞赛语境下，“非传统问题”通常被理解为一些特定形式的题目，如交互题、提交答案题、通信题等等。而本文所讨论的这类非传统问题不拘泥于这一理解。许多题目在形式上与一般的算法竞赛题并没有什么差异，但是用于解决问题的方法十分新颖，难得一见。因此，笔者在这篇论文中将这些题目及其解法收集起来，以供读者欣赏。

## 1 概述

相比于普通的问题，本文所选的题目思维链条往往简短但跳跃，这导致许多选手在面对这些题目时会出现无从下手的困境。为了避免本文变成题目清单，笔者将解决这些问题的方法归纳为以下几类：

- 通过建立映射转化问题
- 使用数学手段解决问题
- 发掘并分析题目隐含的性质
- 使用异于常规的方法解题

接下来，我们将每种方法与其例题相结合进行研究分析。

## 2 通过建立映射转化问题

对于许多计数类问题，我们可以通过建立映射来转化问题。具体而言，对于集合 $A$ 的计数问题，可以通过建立 $A$ 到 $B$ 的一一映射将其转化为对集合 $B$ 的计数。

**例 2.1** (成熟时追随原神[1])**.** 给定一棵初始大小为 $n$ 的树，要求支持加叶子，删叶子，换根三种修改共 $m$ 次，并在每次修改后回答如下询问：每个点从其所有儿子中随机选择一个作为重儿子，进行轻重链剖分后的重链数的期望。$1 \le n, m \le 2 \times 10^5$.

[1] https://www.luogu.com.cn/problem/P8882

观察到如下事实：

**引理 1.** 无论如何选择重儿子，对有根树进行轻重链剖分后的重链数总是等于叶子数。

证明. 根据轻重链剖分的定义，一条重链上深度最深的点一定是叶子节点。同时，每个叶子节点也一定是某条重链上最深的点。因此重链与叶子节点构成了一一对应的关系，这自然就证明了上述引理。 □

根据引理 1,我们只需知道每次修改后的叶子数,这容易使用数据结构在 $\Theta(n)$ 或 $\Theta(n \log n)$ 的复杂度内实现。

作为算法竞赛中常用的算法，轻重链剖分拥有的良好性质容易让选手在本题的思考上走上歧路，但实际上只需简单地对每条重链和叶子节点建立一一对应即可解决问题。

## 3 使用数学手段解决问题

对于一些有浓厚数学背景的题目，可以考虑使用一些数学中的经典结论来解决问题。

**例 3.1** (幻想中成为原神[2])**.** 给定 $n$，计算 $[1,n]$ 中有大于 1 的平方因子数的个数。$n$ 在范围内随机生成，允许 $2 \times 10^4$ 的绝对误差。$1 \le n \le 10^{18}$.

实际上本题即为允许一定误差的求 $n$ 以内无平方因子数的密度 [1].

记 $g(i)$ 为 $i$ 最大的平方因子，所求即为

$$\begin{aligned} & n - \sum_{i=1}^{n} [g(i) = 1] \\ = & n - \sum_{i=1}^{n} \sum_{d \mid g(i)} \mu(d) \\ = & n - \sum_{d=1}^{\sqrt{n}} \mu(d) \left\lfloor \frac{n}{d^2} \right\rfloor \end{aligned}$$

注意到，题目中允许了 $2 \times 10^4$ 的绝对误差，因此接下来我们将利用数学方法进行一些估计。

**定理 1** (巴塞尔问题)**.** $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$.

证明. 可以参考 [2] 中的证明。 □

**定理 2.** $\sum_{k=1}^{\infty} \frac{\mu(k)}{k^2} = \frac{6}{\pi^2}$.

[2]https://www.luogu.com.cn/problem/P8883

证明. 注意到

$$\begin{aligned}
&\sum_{x=1}^{\infty}\frac{1}{x^2}\sum_{y=1}^{\infty}\frac{\mu(y)}{y^2}\\
&=\sum_{x,y}\frac{\mu(y)}{(xy)^2}\\
&=\sum_{k}\frac{1}{k^2}\sum_{xy=k}\mu(y)\\
&=\sum_{k}\frac{1}{k^2}[k=1]\\
&=1
\end{aligned}$$

因此根据定理 1，$\sum_{k=1}^{\infty}\frac{\mu(k)}{k^2}=\frac{6}{\pi^2}$，证毕。 □

根据定理 2，不难想到将 $\sum_{d=1}^{\sqrt{n}}\mu(d)\left\lfloor\frac{n}{d^2}\right\rfloor$ 估计为 $\frac{6}{\pi^2}n$，从而答案即为 $\left(1-\frac{6}{\pi^2}\right)n$. 这一估计的误差项上界在 $O(\sqrt{n})$ 级别，但是实际测试的结果表明，在范围内的随机数据下，绝对误差不会超过 $2\times10^4$.

题目中 $n\le10^{18}$ 的数据范围可能会吓到选手，不过不同寻常的精度限制成为了本题的突破口，让选手可以利用数论中的经典估算解决问题。

**例 3.2** (再谈矩阵乘法[3])**.** $T$ 次给定 $n\times n$ 的矩阵 $\boldsymbol{A}$，要求判断是否存在 $p$ 使得 $\boldsymbol{A}^p=\boldsymbol{0}$. $T=30,1\le n\le100,|A_{i,j}|\le4$.

实际上，有如下事实：

**定理 3.** 若存在 $p$ 使得 $\boldsymbol{A}^p=\boldsymbol{0}$，则一定有 $\boldsymbol{A}^n=\boldsymbol{0}$.

证明. 令 $f$ 为 $\mathbf{A}$ 的特征多项式，根据 Cayley-Hamilton 定理 [3]，我们可以知道若 $R_p(\lambda)=\lambda^p \bmod f(\lambda)$，则 $\mathbf{A}^p=R_p(\mathbf{A})$. 这实际上是一个线性递推的形式，而这个线性递推的特征多项式即为 $f(\lambda)$. 而线性递推可以被写成 $\frac{P(x)}{Q(x)}$ 的形式，其中 $\deg P(x),\deg Q(x)$ 均不超过线性递推的阶数。而存在 $p$ 使得 $\mathbf{A}^p=\mathbf{0}$ 即要求 $\frac{P(x)}{Q(x)}$ 项数有限，此时必然有 $Q(x)\mid P(x)$，又因为 $\deg P(x)<n$，因此这一递推一定在第 $n$ 项处就已经开始取值为 0. □

根据定理 3，我们只需判断 $\mathbf{A}^n=\mathbf{0}$ 是否成立即可。这可以通过选取大质数 $P$，检查 $\mathbf{A}^n$ 在模 $P$ 意义下是否全 0 来实现。我们来简单分析这一做法的正确率：$\mathbf{A}^n$ 中所有元素的绝对值显然有上限 $an^n$，因而其中所有非 0 元素质因子的交不超过 $\omega((an)^n)\le n\log_2 an$ 个，因此在 $k$ 个不同的质数随机选择 $P$ 的正确率就高达 $1-\frac{n\log_2 an}{k}$. 进一步，可以考虑随机向量 $\mathbf{v}$，检查是否有 $\mathbf{v}\cdot\mathbf{A}^n=\mathbf{0}$. 由于向量乘矩阵是 $\Theta(n^2)$ 的，因此总复杂度为 $\Theta(Tn^3)$，正确率即为在原

[3]https://www.luogu.com.cn/problem/P14950

来的结果乘上 $(1-\frac{1}{P})$. 具体来说，在 $[10^8, 10^9]$ 范围内的所有质数中随机选取 $P$，就可以获得高于 99.998% 的正确率。

本题通过直接引入线性代数中的相关定理来解决问题，最终的结论也十分简明。

## 4 发掘并分析题目性质

对于一类最优化问题，其最优解可能具有简单的结构。因此，我们可以先对其进行一定分析，并利用这样的结构解决问题。

**例 4.1** (******D[4]). 定义一个序列 $p_1, p_2, \cdots, p_m$ 是乱的，当且仅当其每一项 $p_i$ 均有 $p_i = 1$ 或 $p_i$ 为质数。有 $T$ 次询问，每次询问给定 $n, k$，需要求出所有 $\sum_{i=1}^{m} p_i \le n$ 的乱的序列的 $\sum_{i=1}^{m}(p_i-k)^2$ 的最大值。$1 \le T \le 10^3, 1 \le n \le 10^9, 1 \le k \le 5 \times 10^4$.

我们不妨先对最优解的结构进行一定的分析：

**引理 2.** 最优序列中，一定不存在 $p_i$ 满足 $1 < p_i \le k$.

证明. 反证：若存在 $1 < p_i \le k$，则将 $p_i$ 拆分为 $p_i$ 个 1 一定更优。 □

**引理 3.** 若 $p_1, p_2, \cdots, p_n > k$，则 $\sum_{i=1}^{n}(p_i-k)^2 < (\sum_{i=1}^{n} p_i - k)^2$.

证明. 展开即证。 □

**引理 4.** 若序列恰有 $m$ 个 1，且满足引理 *2* 的结论，则这些序列中的答案最大值不会超过 $f(m) = m(1-k)^2 + (n-m-k)^2$.

证明. 由于序列满足引理 2 的结论，因此一定对于任意 $p_i > 1$，一定有 $p_i > k$. 因此根据引理，答案不会超过 $m(1-k)^2 + \sum_{p_i>k}(p_i-k)^2 < m(1-k)^2 + \max(n-m-k, 0)^2$. □

**定理 4.** 记 $P$ 为 $[1, n]$ 内的最大质数。当 $P > (n-P)^2 + 2(\sqrt{P}+1)$ 时，要么 $p_i$ 为全 1 序列，要么存在 $p_i = P$.

证明. 我们取两个序列:$(P, 1, 1, \cdots, 1)$ 与 $(1, 1, \cdots, 1)$,对应的答案分别为 $(P-k)^2+(n-P)(1-k)^2$ 与 $n(1-k)^2$，也即 $f(n-P)$ 与 $f(n)$. 注意到，当 $(k-1)^2 \ge P$ 时，由于 $(P-k)^2 \le P(1-k)^2$，最优解一定为 $f(n)$，因此下列证明默认 $(k-1)^2 < P$.

接下来，使用反证法：设最优序列中的最大值 $Q$ 满足 $1 < Q < P$，其中包含 $m$ 个 1. 接下来我们将对 $m$ 的大小分类讨论：

- 若 $m > n - P$，则根据引理 4，其答案不会超过 $f(m)$. 而 $f(x)$ 是二次项系数为正的二次函数，因此一定有 $f(m) < \max(f(n-P), f(n))$，从而导出矛盾。

[4] https://www.luogu.com.cn/problem/P9817

- 若 $m \le n-P$，则除了最大值 $Q$ 以外，一定存在至少一个 $p_i$ 满足 $p_i > 1$. 若 $Q < P-1$，则可以通过 $Q \leftarrow Q+1, p_i \leftarrow p_i - 1$ 的调整来让答案更大，因此一定有 $Q = P-1$. 此时根据引理 2，所有的 $p_i > 1$ 一定有 $p_i > k$，再结合引理 3，可以得到答案不超过 $(P-1-k)^2 + \max(n-m-P+1-k, 0)^2 + m(1-k)^2 \le (P-1-k)^2 + \max(n-P+1-k, 0)^2 + (n-P)(1-k)^2$. 用 $f(n-P)$ 与其作差，得到 $2P - 2k - 1 - (n-P)^2 - (k-1)^2 + 2(k-1)(n-P) \ge P - 2(\sqrt{P}+1) - (n-P)^2 > 0$，矛盾。

□

根据定理 4，当 $n-P$ 相较 $P$ 较小时，我们可以将问题转化到规模更小的问题 $(n-P, k)$ 上。实际上，不难验证当 $127 \le n \le 10^9$，总有 $P > (n-P)^2 + 2(\sqrt{P}+1)$. 因此，只需预处理出 $n$ 较小时的答案，并在每次询问时不断暴力找到 $P$，即可解决本题。记 $n$ 以内质数的最大间距为 $f(n)$，直接实现复杂度为 $\Theta(\sqrt{n}) - O\left(T\left(\frac{B^2}{\log B} + \frac{\sqrt{n} f(n)}{\log n}\right)\right)$，其中 $B = 126$. 进一步，可以通过枚举 $\sqrt{n}$ 以内的所有素数 $p$，并将 $(n-f(n), n]$ 内所有 $p$ 的倍数筛掉来找 $P$，这样复杂度为 $\Theta(\sqrt{n}) - O\left(T\left(\frac{B^2}{\log B} + \frac{\sqrt{n}}{\log n} + f(n)\log\log n\right)\right)$.

本题的结论出乎意料的简单，但其证明需要反复使用调整法、反证法，并结合对质数间距的观察解决问题。这启示我们，在设计算法前，应该先对题目的性质进行冷静地分析。

**例 4.2** (合并香蕉[5])**.** 有 $n$ 条香蕉，第 $i$ 条香蕉有初始大小 $a_i$ 与参数 $k_i$。初始时，每一条香蕉即为一堆。每当第 $i$ 条香蕉参与一次合并（即合并的两堆香蕉中包含第 $i$ 条香蕉），其大小会变成原来的 $\frac{1}{k_i}$。显然，在进行 $n-1$ 次合并后，一定只会剩下一堆香蕉，请最大化此时所有香蕉的大小之和。$3 \le n \le 10^5, 1 \le a_i \le 10^5, 2 \le k_i \le 10$.

记第 $i$ 个香蕉参与合并的次数为 $c_i$，我们所需做的就是最大化 $\sum_{i=1}^{n} a_i k_i^{-c_i}$.

**定理 5.** 最优的合并策略中，必然有 $\{c_1, c_2, \cdots, c_n\} = \{1, 2, 3, \cdots, n-1, n-1\}$.

证明. 使用调整法。考虑合并形成的最优二叉树，设其不满足上述性质，则必然存在至少两个节点，满足其两个儿子均为叶子节点。则这样的节点中深度次深的 $x$，记其两个儿子为 $a, b$；再取与 $x$ 同深度的另一节点 $y$，记其两个儿子为 $c, d$。由于 $x$ 是次深的，因此 $y$ 必然存在，且 $c, d$ 中至少有一个是叶子节点，不妨设其为 $c$.

接下来，记 $v_x = a_x k_x^{-c_x}$，则在原树中，$a, b, c$ 三点对答案的贡献即为 $v_a + v_b + v_c$。注意到，这里的 $a, b, c$ 可以随意进行调换，因此我们不妨设 $(1 - \frac{1}{k_c})v_c \ge \max((1-\frac{1}{k_a})v_a, (1-\frac{1}{k_b})v_b)$ 最大。我们作形如这样的调整：将 $d$ 的整个子树插入到上方，使其成为 $x$ 的兄弟。注意到，此时 $d$ 子树的深度完全没有发生变化，而 $c$ 的深度减少 1，$a, b$ 的深度增加 1，从而 $a, b, c$ 对答案的贡献变为 $\frac{v_a}{k_a} + \frac{v_b}{k_b} + v_c k_c$. 与原来的贡献相减，得到 $\Delta = (k_c - 1)v_c - ((1-\frac{1}{k_a})v_a + (1-\frac{1}{k_b})v_b)$. 而注意到 $(k_c - 1)v_c = k_c(1-\frac{1}{k_c})v_c \ge 2(1-\frac{1}{k_c})v_c \ge (1-\frac{1}{k_a})v_a + (1-\frac{1}{k_b})v_b$，也即 $\Delta \ge 0$. 因此，这样的调整总会让答案不劣。同时，注意到每次调整后，所有叶子节点的深度和总会 +1，这说明我们的调整过程是单调的，最终总能使其到达引理中所描述的状态，因此证明完毕。 □

[5] https://www.luogu.com.cn/problem/P11085

根据定理 5，现在我们已经知道可重集 $\{c_1, c_2, \cdots, c_n\}$，只需对每个 $i$ 分配对应的 $c_i$. 这是二分图最大权匹配的形式。如果直接进行二分图最大权匹配，复杂度难以接受。注意到，当 $c_i \geq 50$ 时，有 $na_i k_i^{-c_i} \leq 10^5 \cdot 10^5 \cdot 2^{-50} < \epsilon = 10^{-5}$，因此我们只需保留右部的 $\Theta(\log V)$ 个点。进一步，我们发现在 $k_i$ 相同时，$a_i$ 较大的点被分配到的 $c_i$ 一定较小，因此我们对于同一个 $k_i$ 也只需保留前 50 大的 $a_i$，这样左部点的数量也被压缩到了 $\Theta(k \log V)$. 而在这张图上跑费用流的复杂度即为 $O(k \log V \cdot k \log^2 V \cdot \log V) = O(k^2 \log^4 V)$.

本题需要选手通过调整法等手段分析出最优解的简单结构，从而将问题转化为匹配问题，再利用精度限制缩减问题规模。与上述题目类似，本题同样属于结论简单，但得到结论的过程却有一定难度的题目。

## 5 使用异于常规的解题方法

对于某些题目，我们可能可以使用一些非常规做法来解决问题，比如打表、搜索与随机调整，等等。而这样的算法通常并不在出题人与选手的考虑范围内，因此这些题目可以看作是一些在算法竞赛外的有趣尝试。

**例 5.1** (Fermat II[6])**.** 有 $T$ 次询问，每次询问给定 $p, n$，要求判断 $p$ 是否是 $n! + 1$ 的最小质因子。$1 \leq T, p, n \leq 2.5 \times 10^5$，有 ***50 KB*** 的代码长度限制。

注意到，满足 $n! + 1$ 的最小质因子 $p \leq 2.5 \times 10^5$ 的 $n$ 不太多，因此一个自然的想法是记录所有这样的 $(n, p)$. 具体而言，从小到大枚举 $1 \times 2.5 \times 10^5$ 的所有质数 $p$，再检查 $n! + 1$ 模 $p$ 意义下的余数，即可在 $\Theta(\frac{n^2}{\log n})$ 的复杂度内找到所有的 $(n, p)$.

显然，这样的复杂度在本题的数据范围下远远不够，但是我们可以将 $(n, p)$ 压缩成表传入程序。直接进行压缩显然会超过 50 KB 的代码长度限制。下面将从压缩方法和 $(n, p)$ 的数量两个方面来优化。

压缩方法上，可以将 $(n, p)$ 按照 $p$ 从小到大排序，对于每个 $(n, p)$，设排序后序列中的上一个元素为 $(n', p')$，若 $p' = p$ 则记录 $n + 2.5 \times 10^5$，否则记录 $n$；再额外记录每个质数 $p$ 是否在 $(n, p)$ 中出现过。这样我们将原来 $2l$ 个值域为 $[1, 2.5 \times 10^5]$ 规模的信息缩减到了 $l$ 个值域为 $[1, 5 \times 10^5]$ 和 $\pi(2.5 \times 10^5)$ 个值域为 $[0, 1]$ 的信息。使用 ASCII 码中所有无需转义，也不会导致转义的可见字符，可将每个 $[1, 5 \times 10^5]$ 的信息压缩至 3 Byte，13 个 $[0, 1]$ 信息压缩至 2 Byte.

接下来，考虑缩减 $(n, p)$ 的数量。

**定理 6** (Wilson 定理)**.** 对于质数 $p$，有 $(p-1)! \equiv -1 \pmod{p}$.

证明. 可以参考 [4] 中的证明。 □

[6] https://www.luogu.com.cn/problem/T340361

因此，对于所有 $n = p - 1$ 的 $(n, p)$，可以将其从表中删去。

**引理 5.** 对于奇质数 $p$，若 $\left(\frac{p-1}{2}\right)! + 1 \equiv 0 \pmod{p}$，则一定有 $p \equiv 3 \pmod 4$.

证明. 当 $p \equiv 1 \pmod 4$ 时，根据 Wilson 定理，$-1 \equiv (p-1)! \equiv \left(\left(\frac{p-1}{2}\right)!\right)^2 \pmod{p}$，从而一定有 $\left(\frac{p-1}{2}\right)! \not\equiv -1 \pmod{p}$. □

因此，我们可以再对所有 $p \equiv 3 \pmod 4$ 记录 $(\frac{p-1}{2}, p)$ 是否在原表中出现，若出现则将其从原表中删去。

最后，我们可以在程序中直接执行最开始的算法得到 $p$ 较小时的所有 $(n, p)$，设 $p$ 的阈值为 $B$，则这部分的复杂度为 $\Theta\left(\frac{B^2}{\log B}\right)$. 再结合上述所有优化，可以在 45 KB 的码长内压缩所有信息，并用 5 KB 的码长进行解码。解码的过程是线性的，因此最终复杂度 $\Theta\left(\frac{B^2}{\log B} + n\right)$ – $\Theta(1)$.

本题虽然形式上属于“传统题”，但是其解法却需要在本地得到查询所需的所有信息并打表，再通过题目的性质压缩信息。这样的题目在算法竞赛中并不多见，可以说十分新颖。

**例 5.2** (火大[7])**.** $T$ 次给定两个正整数 $n, m$，构造一个满足如下要求的长为 $n$ 的整数序列 $a_1, a_2, \cdots, a_n$ 或报告无解：

- 对于任意 $1 \le i \le n$，满足 $0 \le a_i < m$.
- 对于任意 $1 \le i \le j \le n$，存在 $1 \le k \le j$，满足 $a_k \in [\frac{m(i-1)}{j}, \frac{mi}{j})$，其中 $i, j, k$ 均为整数。

$1 \le T \le 4.4 \times 10^4, 1 \le \sum n < 10^6, 1 \le m \le 10^8$.

不妨先允许构造出的 $a$ 是实数的情况，此时 $m$ 对于问题而言并不重要。称集合 $S_n = \{x \mid \exists 0 \le i < j \le n, x = \frac{i}{j}\} \cap \{1\}$ 构成的集合为 $n$ 处的所有关键点，那么条件中所有对于 $a_i$ 的约束都形如 $a_i$ 必须落在关键点构成的某个区间中。

现在，对于一组解 $a_1, a_2, \cdots, a_n$，我们考虑所有对 $a_i$ 限制的交，即形如 $a_i \in [l_i, r_i)$，其中 $l_i, r_i \in S_n$. 显然，所有 $[l_1, r_1], [l_2, r_2], \cdots, [l_n, r_n]$ 应该两两不交，因此可以将其从小到大排序。

接下来，我们使用增量构造，即考察 $n \to n+1$ 时这些区间的变化：注意到，$j = n+1$ 时，将给每个 $a_i$ 分配一个 $[\frac{k}{j}, \frac{k+1}{j})$ 的限制，不妨枚举 $a_{n+1}$ 分配到的是哪一个。这样，剩余的限制与 $a_i$ 的匹配关系一定是从小到大一一匹配。于是我们很容易在 $\Theta(n^2)$ 的复杂度内完成增量。

记 $f(n)$ 为所有本质不同的限制序列 $[l_1, r_1), [l_2, r_2), \cdots, [l_n, r_n)$ 的个数，一个非常宽松的上界是 $f(n) = O(n!)$. 但是，简单实现后可以发现，$f(18) = 0$，且 $\max_{n=1}^{17} f(n) = 2716$，因此这一算法的复杂度完全可以接受。

题目中还需回答 $T$ 组含有 $m$ 的询问。只需注意到以下引理：

[7] https://www.luogu.com.cn/problem/P11074

**引理 6.** $\forall 0 \le a < b \le n, 0 \le c < d \le n$，若 $\frac{a}{b} \ne \frac{c}{d}$，且 $a, b, c, d \in \mathbb{Z}$，则 $\left|\frac{a}{b} - \frac{c}{d}\right| \ge \frac{1}{n^2}$.

证明. 通分即证。 □

于是根据引理 6，当 $m \ge n^2$ 时，任意一组限制序列均可以构造出符合要求的解。而对于 $m < n^2$ 的情况，只需 $\Theta(nf(n))$ 遍历所有限制序列进行检查即可。对询问进行记忆化，总复杂度为 $\Theta(\sum_{i=1}^{k} i^3 f(i)) - \Theta(k)$，其中 $k = 17$.

本题的解法实际上来自于对普通搜索的改进，通过记录对每个位置的限制而非具体数值，并将搜索修改为增量，使得最终的有效状态数在可接受范围内。搜索算法的重要性在算法竞赛中往往容易被忽视，本题的解法也因此显得一反常规。

**例 5.3** (过河卒[8])**.** 给定非负整数 $L$，构造一个 $n \times n$ 的棋盘，每个位置为普通点或禁止点，使得从 $(1,1)$ 走到 $(n,n)$，只能向右或者向下走，且不能经过禁止点的方案数恰为 $L$，共 $T$ 组询问。你需要保证构造的 $n \le 30$. $T = 100, 0 \le L \le 10^{16}$，$L$ 在范围内均匀随机生成。

以下均认为 $n = 30$. 考虑如下算法：

1. 随机选择一个 $n \times n$ 的棋盘，每个位置有 $p$ 的概率为禁止点，并计算出当前路径数量 $F$.
2. 对每个普通点 $(i, j)$ 计算出将其设为禁止点后从 $(1,1)$ 到 $(n,n)$ 的路径数量 $f_{i,j}$.
3. 在所有 $L \le f_{i,j} \le F$ 中，取最小的 $f_{i,j}$，将 $(i, j)$ 设为禁止点。若此时 $F = L$，则我们已经找到了一组解；若 $F > L$，则继续回到第 2 步；若不存在这样的 $f_{i,j}$，则重新回到第 1 步随机棋盘。

下表展示了取 $p = \frac{1}{300}$ 时，在 $T = 10^3$ 次随机询问下，不同范围内的 $L$ 回到第 1 步的次数 $c_1$ 与第 2 步的次数 $c_2$：

| $L \le$ | $10^9$ | $10^{12}$ | $10^{14}$ | $10^{16}$ |
|---|---|---|---|---|
| $\overline{c}_1$ | 15.56 | 27.85 | 48.00 | 60.71 |
| $\max c_1$ | 177 | 187 | 259 | 557 |
| $\overline{c}_2$ | 401.04 | 610.62 | 786.32 | 841.23 |
| $\max c_2$ | 4561 | 4020 | 4529 | 7743 |

而一次调整的复杂度为 $\Theta(n^2)$，因此上述算法在数据范围内容易通过。

我们使用了一个表现十分优秀的随机化调整算法来解决问题。不过，尽管上述算法可以轻松通过本题范围内的数据，但是笔者暂时无力对这一算法的正确率进行严格说明，也欢迎大家对此进行讨论。

[8] https://www.luogu.com.cn/problem/P14952

## 6 总结

本文针对几类非传统的解题方法，选取并讲解了许多不同的题目。笔者深知这些题目游离在信息学竞赛考试内容之外，只是想向大家分享这些灵活而又有趣的解法。本文作为抛砖引玉，希望能够引起大家对于这些题目的兴趣，也希望大家能够在未来创造出更多有趣的题目。

## 致谢

感谢中国计算机协会提供学习和交流的平台，感谢国家集训队教练的指导。
感谢一路上家人对我的支持与陪伴。
感谢李青阳同学在论文上为我提供的帮助。
感谢李青阳同学、张程皓同学、黄锦扬同学与李宁远同学提供论文中所涉及的题目。
感谢北师大实验中学胡伟栋等老师对我的栽培，以及其他所有帮助过我的老师和同学。

## 参考文献

[1] Wikipedia, Square-free integer

[2] Wikipedia, Basel problem

[3] Wikipedia, Cayley–Hamilton theorem

[4] Wikipeida, Wilson’s theorem

# 短排列模式匹配相关问题

天津英华实验学校 朱鹏睿

## 摘 要

本文围绕排列模式匹配展开，总结了若干短排列模式匹配相关的问题及其解法。阐述了信息学竞赛中的一些问题与短排列模式匹配的联系，并使用相关思路为这些问题给出了高效解法。

## 1 引言

排列匹配问题在组合问题方面有着广泛的应用，许多关于排列的性质可以被表示为避免出现某些排列的形式，例如排列是否可以用栈排序、排列是否是可分离排列。许多信息学竞赛题目也可以被表示为询问一些短排列模式匹配的相关问题。笔者发现这类问题在信息学竞赛中还没有系统的介绍，因此在这篇论文中对一些相关成果进行总结。

## 2 记号与约定

**定义 2.1.** 一个长为 $n$ 的排列是一个双射 $\pi : [n] \to [n]$，其中 $[n] = \{1, 2, \cdots, n\}$。

**定义 2.2.** 定义 $\pi(l : r)$ 为序列 $[\pi(l), \pi(l+1), \cdots, \pi(r)]$。

**定义 2.3.** 定义一个排列 $\pi$ 的点集表示 $S(\pi)$ 为：若 $|\pi| = n$，则 $S(\pi) = \{(i, \pi(i)) : 1 \leq i \leq n\}$。

**定义 2.4.** 对于一个长为 $k$ 的数列 $a$，满足 $a$ 中元素两两不同，定义其序排列 $\mathrm{ord}(a)$ 为唯一的一个排列 $\sigma$，使得 $|\sigma| = k$ 且 $\forall 1 \leq i < j \leq k, \sigma(i) < \sigma(j) \iff a_i < a_j$。

**定义 2.5.** 对于一个长为 $n$ 的排列 $\pi$，称其的一个子序列 $i_1, i_2, \cdots, i_k$ 是 $\sigma$ 的一次出现/嵌入，当且仅当 $\mathrm{ord}([\pi(i_1), \pi(i_2), \cdots, \pi(i_k)]) = \sigma$。

**定义 2.6.** 对于两个排列 $\pi, \sigma$，称 $\sigma$ 在 $\pi$ 中出现，当且仅当存在一个 $\pi$ 的子序列是 $\sigma$ 的一次出现。

# 3 排列模式匹配问题

**问题 3.1** (排列模式匹配问题)**.** 给定一个文本排列 $\pi$，以及一个模式排列 $\sigma$，判断 $\sigma$ 是否在 $\pi$ 中出现。

我们设 $|\pi| = n$，$|\sigma| = k$。该问题在学术界已经有非常广泛的研究。事实上，一般情况下该问题关于 $n, k$ 是 NP-hard 的。因此研究主要考虑 $\pi, \sigma$ 有特殊性质，或者 $k$ 较小的情况。

对于 $k \leq 4$ 的情况，这个问题有简单且高效的做法，事实上，我们在 $k \leq 4$ 的时候可以解决更强的问题，这部分将在后续章节中介绍。

而对于 $k$ 任意的情况，当我们将 $k$ 视作参数时，存在关于 $n$ 线性的做法，这一节我们将着重介绍这一算法的大致思想。

**定理 3.1.** 排列模式匹配问题可以在 $f(k) \cdot n$ 的复杂度内解决，其中 $f(k)$ 为一个关于模式长度 $k$ 的函数。

## 3.1 排列的分解

为了解决这个问题，我们引入一种对排列的分解：

**定义 3.1.** 定义一个矩形为一个横坐标区间 $I$ 和纵坐标区间 $J$ 的笛卡尔积 $I \times J$。定义一个排列 $\pi$ 中的元素 $i$ 在一个矩形的内部当且仅当 $(i, \pi(i))$ 在这个矩形的范围内部。

**定义 3.2.** 定义一个排列 $\pi$ 矩形分解的过程为：

维护一个矩形集合 $R$，初始 $R = [i, i] \times [\pi(i), \pi(i)] : (i, \pi(i)) \in S(\pi)$。接下来我们执行 $n-1$ 步操作，每次从 $R$ 中找出两个矩形 $A, B$ 从 $R$ 中删除，然后将它们的边界框（最小的包含 $A, B$ 的矩形）加入 $R$。

对于 $i = 0, 1, \cdots, n-1$，令 $R_i$ 为执行 $i$ 步操作之后矩形集合 $R$ 的状态。根据定义，$R_0$ 为初始的若干单点矩形的集合，$R_{n-1}$ 只包含一个 $[1, n] \times [1, n]$ 的矩形。

**定义 3.3.** 对于一个矩形集合 $R$ 以及其中一个矩形 $A$，定义 $\text{view}_\alpha(R, A)$ 表示满足 $B \in R$，且 $A$ 和 $B$ 的 $\alpha$ 坐标区间有交的 $B$ 数量，其中 $\alpha$ 为 $x$ 或 $y$。称这样的两个矩形可以互相看到，根据我们的定义，所有矩形都能看到自己。定义 $\text{view}(R, A)$ 表示 $\max(\text{view}_x(R, A), \text{view}_y(R, A))$。

**定义 3.4.** 定义一个矩形分解的宽度为 $\max_i \max_{A \in R_i} \text{view}(R_i, A)$。定义一个排列 $\pi$ 的宽度 $w(\pi)$ 为所有分解方式的宽度最小值。

我们在分解的过程中允许出现矩形相交的情况，矩形的合并会形成一个类似二叉树的结构。容易发现一个排列 $\pi$ 的宽度大于等于任意被其包含的排列 $\sigma$ 的宽度，我们只要将 $\pi$ 的合并过程中涉及不在 $\sigma$ 中的元素的操作全部忽略，这样得到的分解宽度一定不超过原分解宽度。

我们声称，如果我们求解了 $\pi$ 的一个宽度较小（$f(k)$ 级别）的分解，那么我们可以在优秀的时间复杂度内解决模式匹配问题。

## 3.2 格状排列

宽度较小的分解不一定存在。接下来我们引入格状排列的定义，可以说明，如果我们找不到宽度较小的分解方式，那么 $\pi$ 一定包含一个格状排列。

**定义 3.5.** 定义一个排列 $\sigma$ 是 $r \times s$ 格状排列，当且仅当 $|\sigma| = rs$，且可以将 $x$ 轴 $[1, rs]$ 划分为 $r$ 个区间 $I_1, I_2, \cdots, I_r$，将 $y$ 轴 $[1, rs]$ 划分为 $s$ 个区间 $J_1, J_2, \cdots, J_s$，使得每个 $I_a \times J_b$ 的矩形内都存在恰好一个 $(i, \sigma(i))$ 的点。

接下来我们为格状排列的宽度给出下界。

**定理 3.2.** 一个 $(2r+2) \times (2r+2)$ 格状排列的宽度 $\geq r$。

证明. 考虑我们第一次合并出矩形 $A$ 使得 $A$ 在格状排列中的连续至少三行（或三列）内各包含至少一个点。不妨假设是连续三行，且这三行分别为 $x-1, x, x+1$，那么 $A$ 在行的范围上一定完全包含第 $x$ 行。

而第 $x$ 行中任意矩形最多只能跨过两列，那么合并前该行的点至少形成 $\frac{1}{2}(2r+2) = r+1$ 个矩形，而合并之后最多会导致该行矩形数减少 1，因此合并之后该行至少还有 $r$ 个矩形，那么此时 $A$ 的可见范围 $\geq r$，分解的宽度 $\geq r$。 □

因此，当一个排列中包含足够大的格状排列时，我们无法得到一个 $f(k)$ 级别宽度的分解。但注意到如果排列中包含了一个 $k \times k$ 的格状排列，那么其一定包含所有长为 $k$ 的排列。接下来我们设计一个算法，这个算法在 $\pi$ 上运行时要么找出一个 $k \times k$ 的格状排列，要么找出一个 $\pi$ 的宽度较小的分解。

为了达成这件事情，我们再引入一个定理。

**定理 3.3.** 令 $h(r) = r^4\binom{r^2}{r}$。对于任意 $p, q, r \in \mathbb{N}$，如果 $M$ 是一个 $[1, p] \times [1, q]$ 的子集且 $|M| > h(r)(p+q-2)$。那么 $M$ 包含一个 $r \times r$ 格状点集，并且存在算法在 $O(|M|)$ 的时间复杂度内找到这样的格状点集。

证明略为繁琐，且与本文算法关系不大，这里不做过多赘述。

## 3.3 分解算法

接下来我们描述算法过程：

我们时刻维护划分 $x$ 轴与 $y$ 轴的方式，设其划分出的网格叫 $G$。

维护一个矩形集合 $R$，初始 $R$ 为所有 $[i, i] \times [\pi(i), \pi(i)]$ 形成的集合。

令 $d = 4h(k)$，合并的时候时刻满足如下性质：

- 每个 $R$ 中的矩形都被 $G$ 的某个格子完全包含，且每次合并的两个矩形属于同一个格子。

- $G$ 的每行每列的矩形数量 $\leq d$。
- $G$ 的任意相邻两行或相邻两列矩形数量 $> d$。

算法的流程如下：

初始令 $G$ 的每一行每一列（除最后一行最后一列）都恰好有 $d$ 个点，这样得到的划分显然满足上述条件。

如果 $G$ 的每一个格子内都至多有一个矩形，设 $M$ 为 $G$ 所有有矩形的格子，且此时 $G$ 有 $p$ 行 $q$ 列，那么 $|M| > d\lfloor\frac{p}{2}\rfloor$，且 $|M| > d\lfloor\frac{q}{2}\rfloor$，因此 $|M| > d \cdot \frac{1}{2}(\frac{p-1}{2} + \frac{q-1}{2}) = h(k)(p+q-2)$。那么此时可以构造出一个 $k \times k$ 的格状子排列，跳出算法。

否则，找到任意一个有至少两个矩形的格子，从这个格子中选择任意两个矩形 $B, C$，将它们合并起来得到矩形 $A$。合并完之后每行每列的矩形数一定会更小。如果产生了相邻两行的矩形数之和 $\leq d$，那么我们合并这两行，对于列同理。

实现细节：

使用链表分别维护 $G$ 每行每列的所有矩形，以及每个 $G$ 的格子中的所有矩形，并再开一个链表维护所有矩形数 $\geq 2$ 的块。

那么除了合并两行或者两列之外的所有操作都是可以做到 $O(1)$ 的，而合并两行两列的时候我们暴力扫一遍所有涉及到的矩形，复杂度 $O(d)$，这样的过程最多会进行 $O(\frac{n}{d})$ 次，因此总复杂度依然是线性的。

我们就获得了一个 $O(n)$ 的算法，要么返回一个 $k \times k$ 的格状排列，要么给出一个宽度 $\leq d$ 的分解。

## 3.4 借助分解求排列模式匹配

定义对一个矩形集合 $R$，定义其可见关系图为一张无向图，其中每个点对应 $R$ 中的一个矩形，两个点有边相连当且仅当它们互相可见。

我们在分解树上自顶向下试图将排列 $\sigma$ 嵌入排列 $\pi$，使用动态规划求解这个问题。具体地，设状态 $(i, K, F)$，其中：

- $0 \leq i \leq n-1$。
- $K$ 是 $R_i$（执行完 $i$ 次操作的矩形集合）可见关系图上的一个连通块。
- $F$ 是一个 $S(\sigma)$ 到 $K$ 的一个对应关系，满足 $S(\sigma)$ 中每个点都至多对应一个 $K$ 中矩形（可以不对应任何矩形），且 $K$ 中每个矩形至少被一个 $S(\sigma)$ 中的点对应。

令 $S'$ 为所有在 $F$ 中有对应的 $S(\sigma)$ 中的点组成的集合。我们想找出一个将 $S'$ 中的点嵌入 $\pi$ 的方式，使得对于所有 $x \in S'$，$x$ 嵌入的点在分解树上 $F(x)$ 矩形的子树中，即合并

出 $F(x)$ 矩形的合并过程中合并了 $x$ 嵌入的点（注意 $F(x)$ 矩形在几何意义上包含 $x$ 嵌入的点并不一定意味着这个条件成立）。

最终我们想要求的就是 $(n-1, R_n, F)$（其中 $F$ 将所有 $S(\sigma)$ 中的点对应到 $R_n$ 中唯一一个矩形）是否可行。

考虑如何对一个 $(i, K, F)$ 进行转移，从 $R_i$ 到 $R_{i-1}$ 进行的操作就是选择一个矩形 $A$，将其分裂为 $B, C$。如果 $A \notin K$，那么可以直接从 $(i-1, K, F)$ 转移。

而如果 $A \in K$，将 $A$ 裂为 $B, C$ 后，暴力枚举将原来每个对应到 $A$ 中的点此时是对应到 $B$ 还是 $C$ 中。

分裂矩形还有可能导致 $K$ 裂为若干个连通块 $K'_1, K'_2, \cdots, K'_p$，由于连通块的定义是可见意义下连通，因此在每个 $K'_i$ 内部具体选什么点不影响不同 $K'$ 集合之间的点的大小关系，因此这一步可以直接检验是否合法，之后从所有 $(i-1, K'_i, F'_i)$ 的与转移过来。

当 $i = 0$ 时，每个状态是否合法是平凡的。

由于可见关系图中每个点的度数 $\leq d$，并且所有在这个图上大小 $\leq k$ 的连通块可以一个从某点开始，走一条长度不超过 $2k-2$ 且最终返回起点的路径覆盖，而这样的路径条数是 $d^{O(k)}$ 的，因此不同的 $K$ 的数量不超过 $d^{O(k)}$。

一共有 $d^{O(k)} \cdot n$ 种可能的 $K$，且有 $k^{O(k)}$ 种可能的 $F$，因此总状态数是 $n^2 \cdot (dk)^{O(k)}$。但对于一个 $i-1$ 转移到 $i$ 的时候，我们只需要更新 $A \in K$ 的状态，这样的状态只有 $(dk)^{O(k)}$ 个，而更新一个状态的复杂度是 $O(2^k \cdot \text{poly}(k))$ 的。因此我们的总复杂度是 $(dk)^{O(k)} \cdot n$ 的，由于 $d = k^{O(k)}$，因此复杂度也可以表示为 $2^{O(k^2 \log k)} \cdot n$。

因此我们证明了，排列模式匹配问题可以在 $2^{O(k^2 \log k)} \cdot n$ 的时间内解决。这意味着该问题是对于固定 $k$ 是固定参数可解（Fixed-Parameter Tractable）的。

## 4 区间排列模式匹配问题

**问题 4.1 (**区间排列模式匹配问题**).** 给定一个文本排列 $\pi$，以及一个模式排列 $\sigma$，判断 $\sigma$ 是否在 $\pi$ 的每个子区间中出现。

若 $\sigma$ 在 $[l, r]$ 中出现，则 $\sigma$ 一定在所有满足 $l' \leq l \leq r \leq r'$ 的 $[l', r']$ 中出现，即我们需要找到所有的 $[l, r]$，满足 $[l, r]$ 包含 $\sigma$，且不存在任何 $[l', r'] \subset [l, r]$，使得 $[l', r']$ 包含 $\sigma$，称这样的区间是左右-极小的。

这个问题不弱于上述排列模式匹配问题，因此也是 NP-hard 的。并且该问题可以用双指针结合上文的算法做到 $O(f(k) \cdot n^2)$，因此该问题在 $k$ 固定时也是固定参数可解的。

由于笔者水平有限，暂未对一般情况下的问题发现高效的解法，但对于 $|\sigma| \leq 4$ 的问题，均存在高效的 $O(n \log n)$ 做法，下文将对这一部分展开介绍。

## 4.1 $|\sigma| \leq 4$ 的问题

对一个排列 $\sigma$，定义 rev($\sigma$) 为唯一满足 $\forall 1 \leq i \leq |\sigma|, \phi(|\sigma| - i + 1) = \sigma(i)$ 的排列 $\phi$，flip($\sigma$) 为唯一满足 $\forall 1 \leq i \leq |\sigma|, \phi(i) = |\sigma| - \sigma(i) + 1$ 的排列。那么我们可以将匹配 $\text{rev}(\sigma), \text{flip}(\sigma), \text{rev}(\text{flip}(\sigma))$ 的问题规约到匹配 $\sigma$ 的问题。

因此，我们用这种方式划分问题等价类后，当 $|\sigma| \leq 4$ 的时候，剩余的问题有 $\sigma \in \{1, 12, 123, 132, 1234, 1243, 1324, 1342, 1423, 1432, 2143, 2413\}$。

接下来，我们将这些问题分若干类解决。

下文的记号中，$pos_i$ 表示相对大小为 $i$ 的元素的位置，$val_i$ 表示相对大小为 $i$ 的元素的值。例如一个 2341 子序列的限制可以被表示为 $pos_2 < pos_3 < pos_4 < pos_1$ 且 $val_1 < val_2 < val_3 < val_4$。

### 4.1.1 $\sigma = 12\cdots k$ 的问题

使用类似动态规划的手段求解，令 $mnr_{i,j}$ 以 $j$ 位置为起点，选出一个 $12\cdots i$ 的子序列，子序列的最后一个元素最靠左能落在哪个位置，无法选出则为 $n+1$，转移式即为：

$$mnr_{i,j} = \min_{x>j, \pi(x)>\pi(j)} mnr_{i-1,x}$$

可以使用树状数组或类似数据结构维护，求出所有 $mnr_{k,j}$ 之后容易得到所有极小区间的位置，复杂度 $O(kn \log n)$。

### 4.1.2 $\sigma = 132$ 的问题

我们需要找到左右-极小的 132 出现。由于限制只有左右-极小，因此当固定了 2 的位置之后，3 的选取一定是在 $pos_2$ 前面，第一个满足 $\pi(i) > \pi(pos_2)$ 的 $i$，因此可以唯一确定每个 2 对应的最优的 3。

进一步地，1 的选取一定是 $pos_3$ 前，第一个满足 $\pi(i) < val_2$ 的位置，因此当固定 2 的位置之后，整个子序列唯一确定。最后我们只需要将 $O(n)$ 个可能的子区间放在一起取极小即可。

这一过程容易使用线段树维护，复杂度 $O(n \log n)$。

### 4.1.3 $|\sigma| = 4$，且 $\sigma(1) = 2$ 的问题

固定 2 的位置 $pos_2$ 后，我们把所有 $\pi(i) > val_2$ 的 $i$ 称为大位置，所有 $\pi(i) < val_2$ 的 $i$ 称为小位置。

对于这一类的六种情况，我们均要解决 134 以某种顺序排列的问题，由于 1 的选取只有位置重要而值无关紧要，唯一需要考虑的值偏序关系只有 $val_3 < val_4$，因此我们可以使用一个线段树，维护如下信息：

- 1 - 区间内是否存在小位置
- 3,4 - 区间内值最小/最大的大位置
- 13,31 - 区间内能选出的 13/31 子序列中，3 的值的最小是多少
- 14,41 - 区间内能选出的 14/41 子序列中，4 的值的最小是多少
- 34,43 - 区间内是否能选出的 34/43 子序列
- 134（以及其余五种重排）- 区间内是否能选出对应子序列

该信息可以快速合并，可以使用线段树维护。我们对 2 的值从大到小扫描线，每次的修改是将一个小位置改为大位置，然后对某个 $l$ 找到最小的 $r$ 使得 $[l, r]$ 的信息中对应的 134 的重排的信息为真，可以使用线段树上二分实现，复杂度 $O(n \log n)$。

### 4.1.4 $\sigma = 1324$ 的问题

采取更直接一些的做法，固定 1 的位置之后，我们需要拼上一段 324 子序列，这段 324 子序列需要满足的限制是 $pos_3 > pos_1$，且 $val_2 > val_1$，并且由于我们在求所有左右-极小子序列，我们额外希望 $pos_4$ 尽可能小。

因此，我们需要找出所有左右下-极小的 213 子序列。严谨来说，我们希望找到所有三元组对 $(l, r, v)$，使得：

- $1 \le l \le r \le n, 1 \le v \le n$
- 所有满足 $l \le i \le r$ 且 $\pi(i) \ge v$ 的元素组成的子排列包含 213-子序列
- 不存在另一个满足如上条件的三元组 $(l', r', v')$ 使得 $(l, r, v) \ne (l', r', v')$ 且 $l' \ge l, r' \le r, v' \ge v$

为了解决这个问题，我们证明如下结论，并通过证明给出求解方式：

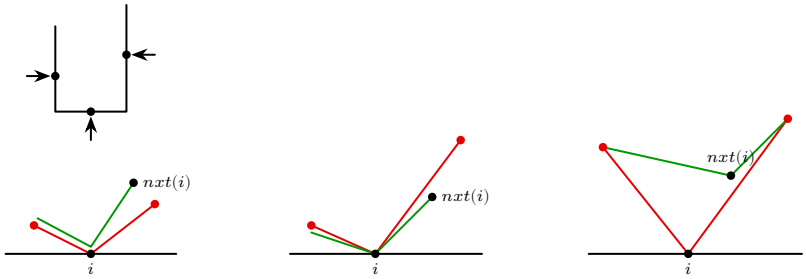
**定理 4.1.** 左右下-极小的 *213* 子序列只有不超过 $n$ 个。

证明. 按照 $val_1$ 从大到小增量维护左右下-极小的 213 子序列集合。

对于一个新加入的元素 $i$，我们证明当 $i$ 作为 1 的时候，3 的选取必定是最小的 $j$，满足 $j > i, \pi(i) < \pi(j)$，称这样的 $j$ 为 $nxt(i)$。

如果 $nxt(i)$ 不存在，那么显然不会新增任何左右下-极小 213 子序列。

否则，设我们选择了某个子序列，满足 $pos_1 = i, pos_3 \neq nxt(i)$，我们证明其一定可以被已有的子序列偏序。




分三种情况讨论：

- 若 $val_3 < val_{nxt(i)}$，那么根据 $nxt(i)$ 的定义，$pos_3 > nxt(i)$，因此直接将 $pos_3$ 替换为 $nxt(i)$ 即可
- 若 $val_3 > val_{nxt(i)} > val_2$，那么我们也可以将 $pos_3$ 替换为 $nxt(i)$
- 若 $val_3 > val_{nxt(i)}, val_2 > val_{nxt(i)}$，那么可以将 $pos_1$ 替换为 $nxt(i)$

而固定 13 之后，2 的选取即为最大的 $j$ 满足 $j < pos_1, val_1 < \pi(j) < val_3$。

因此当多加入一个元素时，左右下-极小的 213 子序列集合大小最多增加 1，最终集合大小不超过 $n$。 □

上述证明也提供了一个可以用数据结构维护的求解该极小集合的方式，可以使用扫描线配合线段树做到 $O(n \log n)$ 的复杂度。

找出左右下-极小的 213 子序列的集合之后，再使用树状数组或者类似数据结构合并 1 和 324 部分即可使用 $O(n \log n)$ 的复杂度求出所有左右-极小的 1324 子序列。

至此，可以发现上述四种情况拼起来已经足够我们解决所有 $|\sigma| \leq 4$ 的问题。并且我们也可以解决这些问题的任意组合，如求解一个排列 $\pi$ 多少子区间不包含 $S$ 中任意一个模式排列，其中 $S$ 中元素为 $|\sigma| \leq 4$ 的排列。

$|\sigma| = 5$ 的问题相比 $|\sigma| = 4$ 的复杂许多，在去除等价问题后一共有 32 类，其中仍有一些笔者尚未找到高效做法。

同时，对于一些特殊的排列，存在 $k$ 任意时的高效做法，如当 $\sigma$ 单峰或单谷时，该问题容易做到 $O(kn \log n)$ 的复杂度。

总结下来，该问题仍然有较为广阔的研究空间，笔者鼓励读者对相关问题进行探索。

# 5 排列模式匹配计数

**问题 5.1 (**排列模式匹配计数**).** 给定一个文本排列 $\pi$，以及一个模式排列 $\sigma$，求有多少 $\pi$ 的子序列 $\phi$ 满足 $\mathrm{ord}(\phi) = \sigma$。

该问题有显然的 $O(n^k)$ 算法，而可以证明，如果对于任意 $\pi$ 和 $\sigma$，该问题都有 $n^{o(k/\log k)}$ 时间复杂度的算法，则指数时间假设（Exponential Time Hypothesis）不成立，这意味着 3-SAT 有 $2^{o(n)}$ 的算法。

## 5.1 一般情况下的算法

朴素暴力是 $O(\binom{n}{k} \cdot k)$ 的，而我们可以用一个简单的想法将其变为 $O(\binom{n}{\lfloor k/2 \rfloor} \cdot n)$。

我们可以枚举 $\sigma$ 所有偶数位置上的元素，这样我们就将奇数位置上的数全部隔绝开了，分为了 $\lceil \frac{k}{2} \rceil$ 种颜色，颜色之间的位置关系已经固定，只需要满足大小关系即可。

给每一个 $\pi$ 中元素按照其在哪个块分配一个颜色 $col_i$。如果 $i$ 是某个 $\sigma$ 偶数位置 $p$ 枚举到的元素，则 $col_i = \sigma(p)$，否则 $i$ 在某个 $\sigma$ 的奇数位置 $p$ 的可行范围内，令 $col_i = \sigma(p)$。

按照值从小到大扫所有元素，维护 $f_x$ 表示当前匹配完 $\sigma(p) \le x$ 的所有 $p$ 的方案数，初值 $f_0 = 1$，其余 $f_i$ 均为 0。设当前扫到元素 $i$，那么执行 $f_{col_i} \leftarrow f_{col_i} + f_{col_i - 1}$。最后答案就是 $f_k$ 的值。复杂度 $O(\binom{n}{\lfloor k/2 \rfloor} \cdot n)$。

事实上，该问题还可以进一步做到 $O(n^{k/4+o(k)})$，限于篇幅这里不展开说明，感兴趣的读者可以自行查阅相关文献。

## 5.2 $k$ 较小的情况

对于长度为 2 的情况，计数排列中 12 子序列数量，即为计数排列逆序对数量，可以使用树状数组做到 $\tilde{O}(n)$。

长度为 3 的情况稍复杂一些，对于 123 依然可以简单求解，设状态 $f_i, g_i$ 表示以 $i$ 结尾的 12/123 子序列数量，然后用树状数组统计，复杂度 $\tilde{O}(n)$。

剩余的情况都可以通过下标翻转或值域反转规约到计数 213，直接做会发现比较困难，但使用容斥原理，我们可以得到如下计算方式：

$$a_i = \sum_{j=1}^{n} [j < i \wedge \pi(j) < \pi(i)]$$

$$\#213 = \frac{1}{2} \left( \sum_{i=1}^{n} a_i^2 - 2 \cdot \#123 - \#12 \right)$$

其中 $a_i, \#123, \#12$ 都可以在 $\tilde{O}(n)$ 的时间内求出，因此 213 的数量也可以在 $\tilde{O}(n)$ 内的时间算出。

这意味着，我们在排列上进行一些偏序的计算，可以得到一个不同排列出现次数的线性组合，考虑一般化一下这个过程。

### 5.2.1 方向模式树

**定义 5.1** (方向模式树)**.** 一个方向模式树（或简称方向树）是一棵有根树，其中每条边上都有一个 $\{NE, NW, SE, SW\}$ 中的方向。

**定义 5.2** (方向树的嵌入)**.** 称方向树 $T$ 的一个到排列 $\pi$ 上的嵌入为一个从 $T$ 中节点到 $\pi$ 中位置的一个映射 $f$，不同节点的 $f$ 可能相同。需要满足，对于 $T$ 中所有不为根的节点 $v$，设其父亲节点为 $u$，其映射到的位置需要满足其父边上的方向关系如下：

- 若父边方向为 $NW$ 或 $NE$，则 $f(v)$ 的值大于 $f(u)$；若父边方向为 $SW$ 或 $SE$，$f(v)$ 的值小于 $f(u)$。
- 若父边方向为 $NW$ 或 $SW$，则 $f(v)$ 的下标小于 $f(u)$；若父边方向为 $NE$ 或 $SE$，$f(v)$ 的下标大于 $f(u)$。

可以发现，一个大小为 $k$ 的方向树 $T$ 在一个长为 $n$ 的排列 $\pi$ 上的嵌入方案数 $\#T(\pi)$ 是可以 $\tilde{O}(n)$ 的时间内求出来的，只要对每条边用数据结构转移一遍即可。

接下来，我们说明方向树的嵌入方案数可以被表示为若干排列的嵌入方案数的线性组合，并且这个线性组合可以在可接受的时间复杂度内求出。

**定理 5.1.** 对于一棵方向树 $T$，其嵌入某个排列 $\pi$ 的方案数 $\#T(\pi)$ 可以被表示为一些排列 $\sigma$ 嵌入 $\pi$ 的方案数 $\#\sigma(\pi)$ 的线性组合。具体地，设

$$\#T(\pi) = \sum_{|\sigma| \le |\pi|} \mathrm{coe}_T(\sigma) \#\sigma(\pi)$$

，则 $\mathrm{coe}_T(\sigma)$ 可以通过下式递推求出：

$$\mathrm{coe}_T(\sigma) = \#T(\sigma) - \sum_{|\rho| < |\sigma|} \mathrm{coe}_T(\rho) \cdot \#\rho(\sigma)$$

证明是比较直观的。该式含义即为从 $\#T$ 嵌入 $\sigma$ 的方案中容斥掉所有不为满射的方式。

可以发现所有方向树的线性组合形成了所有排列出现的线性组合的一个子空间。那么使用计算机辅助，我们可以搜索出所有方向树，然后求出这个子空间。

大小 $\le 4$ 的方向树在长为 4 的排列中形成了一个 23 维的子空间，有一个自由度，这个自由度可以通过如下方式描述：设 $V$ 为所有排列的出现次数形成的线性空间，$W$ 为所有方向树形成的子空间，取唯一的内积使得所有排列单独的出现次数的集合形成一个标准正交基，那么

$$W^{\perp} = (1324 + 4231) + (1432 + 2341 + 3214 + 4123) + (2413 + 3142)$$
$$-(1342 + 1423 + 2314 + 2431 + 3124 + 3241 + 4132 + 4213)$$

特别地，这意味着上述 16 种排列是无法使用方向树直接求解的，但一些它们的线性组合仍然可以求解。同时，这也意味着我们只要额外求出上述 16 种排列中任意一种的出现次数，即可求出所有排列的出现次数。

接下来我们介绍一种在 $\tilde{O}(n^{3/2})$ 时间复杂度内解决计算 3241 出现次数的算法。

将值域分为 $B$ 个块，我们分三种情况统计：

1. 值为 4 的元素和值为 3 的元素不在一个块中
2. 值为 1 的元素和值为 2 的元素不在一个块中，且不满足 (a) 的情况
3. 值为 1 和 2 的元素在一个块中，值为 3 和 4 的元素在一个块中

这三种情况分别解法如下：

1. 枚举值为 4 的元素在哪个块中。从左到右扫描线，我们需要时刻维护出有多少 321 子序列满足当前位置在 21 之间。设这个数量为 $c$，当扫到一个元素时作为 2 时，给 $c$ 加上前面 3 的个数乘上后面 1 的个数，扫到一个数作为 1 的时候，将以它为结尾的 321 子序列从 $c$ 中减去。这两个操作都可以使用树状数组维护，复杂度 $\tilde{O}(n^2/B)$。
2. 枚举值为 1 的元素在哪个块中。从左到右扫描线，我们需要维护出每个前缀里有多少形如 324 的子序列，可以发现应用上文求 213 数量的算法自然求出了以每个元素结尾的 324 子序列数量。限制 4 和 3 不在同一个值域块也是容易的。复杂度 $\tilde{O}(n^2/B)$。
3. 由于条件的限制，合法的 21 对只有 $O(nB)$ 种，合法的 34 对也只有 $O(nB)$ 种，那么我们对 3 的值扫描线，使用高维线段树处理满足 34 和 21 对插起来的下标关系的统计，可以做到复杂度 $\tilde{O}(nB)$。

平衡复杂度可以得到 $B = n^{1/2}$，算法的时间复杂度为 $\tilde{O}(n^{3/2})$。

因此，对于长度 $\leq 4$ 的所有排列 $\sigma$，其在一个长为 $n$ 的排列中出现次数可以在 $\tilde{O}(n^{3/2})$ 的时间复杂度内统计。

方向树的结构依然具有局限性。当 $|\sigma| = 5$ 时其生成的线性空间只有 100 维，显著小于所需的 120 维。并且可能的方向树形态也仅仅是指数级的，而排列数量的增长速度（即 $k!$）是超指数级的，因此方向树注定不能为我们提供一个任意可扩展的求解短排列出现次数的问题，我们需要进一步对这个结构一般化。

### 5.2.2 偏序模式树

**定义 5.3** (偏序模式树)**.** 一个偏序模式树（或简称偏序树）是一棵有根树 $T$，这棵有根树代表了一个二维平面上的点集 $P(T)$，并且定义了一些在 $P(T)$ 上的一些限制：

每个节点 $u$ 上有一个长为 $l_u$ 的排列 $p_u$，表示这个节点对应 $l_u$ 个点，且这些点应该满足 $x_{u,1} < x_{u,2} < \cdots < x_{u,l_u}$，且对于所有 $p_u(i) < p_u(j)$ 的 $1 \le i, j \le l_u$ 均有 $y_{u,i} < y_{u,j}$。

每条从父亲 $u$ 连到儿子 $v$ 的边上都有序关系限制，满足对于任意 $x_{u,i}$ 和 $x_{v,j}$，它们都是可以比较的，且关系一定是小于、等于、大于中的一个，对于 $y$ 同理。

**定义 5.4** (偏序树的嵌入)**.** 一个偏序树 $T$ 到一个排列 $\pi$ 的嵌入为一个从 $P(T)$ 到 $\pi$ 中位置的一个映射 $f$，不同节点的 $f$ 可能相同，且取出所有嵌入的位置的 $(i, \pi(i))$，这个点集必须满足偏序树 $T$ 的所有限制。

一个大小为 $k$ 的偏序树 $T$ 到某个长为 $n$ 的排列 $\pi$ 的嵌入方案数可以在 $\tilde{O}(n^{\max l_u})$ 的时间复杂度内求出。对每一个节点存一个状态，转移时我们使用高维线段树处理边上的序关系，节点上的偏序关系可以直接状态上保证。

注意到方向树即为 $\max l_u = 1$ 的特殊情况，并且可以发现所有排列的出现次数都可以被一个偏序树描述。上文已经说明了 $\max l_u = 1$ 的偏序树（即方向树）在长为 4 时形成了一个 23 维的子空间。而我们可以使用计算机搜索所有 $\max l_u = 2$ 的偏序树，这样的树在长度 $\le 7$ 的时候可以形成满的线性空间，而在长度 $= 8$ 时形成了一个 40319 维的子空间。

因此，运用这个结构我们可以直接得到一个 $\tilde{O}(n^2)$ 计算所有长度 $\le 7$ 的排列在一个长为 $n$ 的排列中的出现次数。

事实上，这个做法也可以在 $\max l_u$ 更大的时候适用，但主要限制为现代计算机的搜索性能。

## 6 一些相关组合性质及例题

**定理 6.1** ($k$-下降排列)**.** 一个排列 $\pi$ 可以被拆为 $k$ 个下降子序列当且仅当其不包含排列 $1, 2, \cdots, k+1$。

证明. 该定理即为 Dilworth 定理的另一个表述。 □

**例题 6.1** (Antiamuny Wants to Learn Swap[1])**.** 对于一个长为 $m$ 的数组 $b$，你可以进行如下两种操作：

1. 选择一个 $1 \le i \le m-1$，交换 $b_i$ 和 $b_{i+1}$。
2. 选择一个 $1 \le i \le m-2$，交换 $b_i$ 和 $b_{i+2}$。

[1] https://codeforces.com/contest/2138/problem/B

定义只使用 1 操作将 $b$ 升序排序的最小操作次数为 $g(b)$。定义使用 1 操作与**至多一次** 2 升序排序的最小操作次数为 $f(b)$。若 $f(b)=g(b)$，则 $b$ 是完美的。

给定一个大小为 $n$ 的排列 $\pi$，$q$ 次询问，每次给定一个区间 $[l,r]$，查询 $\pi(l:r)$ 是否是完美的。$n\le 5\cdot 10^5$。

解法　$g(b)$ 即为 $b$ 的逆序对数量。而在用一次 2 操作的前提下，这次操作必须要使逆序对减小超过 1，唯一能办到这件事情的情况是 $123\to 321$，这时我们减少了三个逆序对，其余情况我们不能减少超过一个逆序对。

因此我们可以猜想：$f(b)<g(b)$ 当且仅当其存在一个 321-子序列。

当 $\pi$ 包含 321 子序列时，设这个子序列为 $\pi(a),\pi(b),\pi(c)$，其中 $a<b<c$。可以发现，$\pi(a:b)$ 中一定存在至少一个相邻逆序对。如果 $a,b$ 不相邻，那么我们就可以交换这个相邻逆序对，让总逆序对变少，对于 $b,c$ 之间同理。这个操作可以一直进行，直到 $a,b,c$ 相邻。此时我们使用一次 2 操作交换 $a,c$，可以使逆序对数一次性减小 3，之后每次再操作一个相邻逆序对。

由于我们每次操作都让逆序对数减小至少 1，且存在一次操作让逆序对数减小 3，因此操作步数小于 $g(b)$。

当 $\pi$ 不包含 321 子序列时，进行 $b_i>b_{i+1}$ 的 1 操作无法让已经满足 $x<y,pos(x)<pos(y)$ 的值的位置关系变为 $pos(x)>pos(y)$，不会产生出 321 子序列。而进行 $b_i<b_{i+1}$ 的 1 操作会让逆序对数 $+1$，与原来可以 $-1$ 的操作形成了 2 的差，而之后 321 交换为 123 使逆序对数减少 3，差也为 2。因此执行了一步 $b_i<b_{i+1}$ 的交换操作的损失在之后是补不回来的。$f(b)$ 不可能小于 $g(b)$。

因此，我们可以使用上文的区间排列模式匹配问题中的算法，得出所有出现 321 的极小区间即可回答询问，复杂度 $O(n+q)$ 或 $O(n\log n+q)$。 ■

**定理 6.2** (可栈排序排列)**.** 一个排列 $\pi$ 可以被栈排序当且仅当其不包含排列 *231*。

证明. 当 $\pi$ 包含 231 时，不妨设 $a<b<c$ 且 $\pi(c)<\pi(a)<\pi(b)$，那么 $c$ 入栈时 $a,b$ 一定都在栈中，但这意味着 $a$ 在 $b$ 之后弹出，因此无法排序。

当 $\pi$ 不包含 231 时，我们归纳证明 $\pi$ 可以用栈排序。设 $n=|\pi|$，$n=1$ 时命题显然。当 $n>1$ 时，我们找到 $\pi(p)=n$ 的位置 $p$，由于 $\pi$ 是 231-规避的，那么 $\max\pi(1:p-1)<\min\pi(p+1:n)$。且由于 $\pi(1:p-1)$ 和 $\pi(p+1:n)$ 也都是 231-规避的，根据归纳假设它们也都可以用栈排序。

那么我们首先将 $\pi(1:p-1)$ 按照其操作序列排序，然后将 $\pi(p)$ 压到栈中，再将 $\pi(p+1:n)$ 按照其操作序列排序，最后弹出栈内的 $\pi(p)$，即可将 $\pi$ 排序。 □

这意味着长为 $n$ 的 231-规避排列的数量是第 $n$ 个卡特兰数。接下来我们对 123-规避排列的数量得到同样的结论。

**定理 6.3.** 长为 $n$ 的 *123*-规避排列数量是第 $n$ 个卡特兰数。

证明. 可以发现 123-规避排列数量等于 321-规避排列数量，那么我们在 321-规避排列上考虑这个问题。当一个排列是 321-规避时，它可以拆为两个递增子序列。进一步可以发现取出这个排列的所有前缀最大值之后，剩余的子序列一定是递增的。

我们设计一个从 321-规避排列 $\pi$ 到 Dyck 路径的一个映射：从 $(0,0)$ 出发，依次从左到右扫描 $\pi$ 中每个元素，维护一个变量 $mx$，初始 $mx \leftarrow 0$。当扫到元素 $i$ 时，如果 $\pi(i) > mx$，就向右上走 $\pi(i) - mx$ 步，向右下走一步，然后令 $mx \leftarrow \pi(i)$；否则 $mx < \pi(i)$，向右下走一步，不对 $mx$ 进行修改。

这样生成的格路一定终点为 $(2n, 0)$，且由于前 $i$ 个数的最大值一定 $\geq i$，因此这条格路也始终在 $y = 0$ 及其上方。因此每个 321-规避排列都对应着一个 Dyck 路径。

对于一条 Dyck 路径，我们将其以所有下降的步分段。接下来我们从左到右扫还原对应到它的 321-规避排列 $\pi$，维护一个集合 $S$，初始为空，以及一个变量 $mx$ 初始为 0：如果当前 $k_i > 0$，那么将 $[mx+1, mx+k_i-1]$ 中的数加入进 $S$，令 $\pi(i) = mx + k_i$，然后令 $mx \leftarrow mx + k_i$；如果当前 $k_i = 0$，令 $\pi(i)$ 为 $S$ 中的最小元素，然后将 $S$ 中最小元素删除。

这样我们证明了 321-规避排列和 Dyck 路径构成了一个双射，因此长为 $n$ 的 321-规避排列的数量是第 $n$ 个卡特兰数。 □

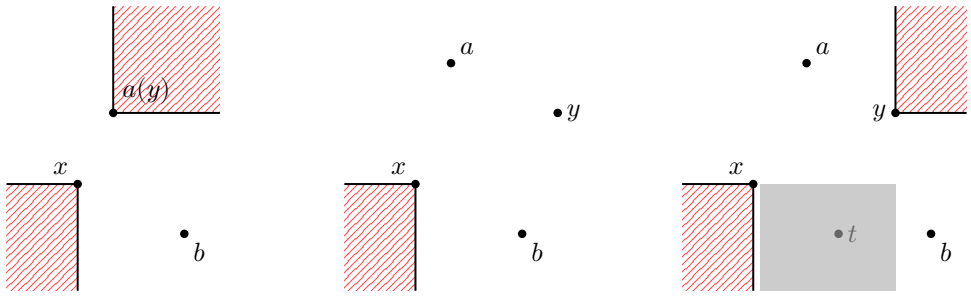
结合这两个结论，可以得到对于任意长为 3 的排列 $\sigma$，长为 $n$ 的 $\sigma$-规避排列数量是第 $n$ 个卡特兰数。

**定理 6.4** (可分离排列)**.** 定义一个长为 $n$ 的排列 $\pi$ 是可分离的，当且仅当 $|\pi| = 1$，或者存在一个 $1 \leq p < n$，使得 $\pi(1:p)$ 和 $\pi(p+1:n)$ 都是可分离排列，且它们的值域无交，即 $\max \pi(1:p) < \min \pi(p+1:n)$ 或 $\min \pi(1:p) > \max \pi(p+1:n)$。

一个排列是可分离的当且仅当其不包含排列 *2413* 与 *3142*。

证明. 当 $\pi$ 包含 2413 或 3142 时，设该子序列四个元素位置分别为 $a < b < c < d$。在分离形成的树形结构上，第一次 $a \leq p < d$ 的操作一定无法合法地将这个排列分开，两边值域必定有交。

当 $\pi$ 不包含 2413 或 3142 时，我们归纳证明 $\pi$ 是可分离的。设 $n = |\pi|$，$n = 1$ 时命题显然。当 $n > 1$ 时，我们找到 $\pi(a) = n$ 的位置 $a$ 和 $\pi(b) = 1$ 的位置 $b$，不妨设 $a < b$。我们找到 $\pi(1:a)$ 中最小值的位置 $x$，然后找到最靠后的满足 $\pi(y) > \pi(x)$ 的位置 $y$：



- 根据定义，$y \geq a$。若 $y = a$，则直接从 $a$ 处切开即可。
- 若 $y > b$，则 $x, a, b, y$ 形成一个 2413 子序列，不合法。
- 若 $a < y \leq b$，显然 $y \neq b$。此时我们从 $y$ 处切开，必定合法：如果 $\pi(1 : y)$ 内的最小值不为 $\pi(x)$，则最小值只能在 $\pi(a + 1 : y)$ 的位置 $t$，而这样 $x, a, t, y$ 就会形成一个 2413 子序列。

  因此 $\pi(1 : y)$ 的最小值为 $\pi(x)$。而根据 $y$ 的定义，$y$ 右侧一定不存在 $> \pi(x)$ 的元素，因此从 $y$ 处切开，两边值域不交。

对于 $a > b$ 的情况类似，那么在 $\pi$ 是 2413, 3142-避免的情况下必定存在一个划分点 $p$ 使得 $\pi(1 : p)$ 和 $\pi(p + 1 : n)$ 的值域不交。而由于 $\pi(1 : p)$ 和 $\pi(p + 1 : n)$ 也都是 2413, 3142-避免的，因此都是可分离排列，那么 $\pi$ 也是可分离排列。 □

**例题 6.2** (Dolls[2])**.** 给定一个长为 $n$ 的排列 $\pi$，有一个长为 $n$ 的区间序列 $a$，初始 $a_i = [\pi(i), \pi(i)]$。每次操作你可以选择两个在 $a$ 中相邻的无交区间，将它们合并为最小的包含它们的区间。求最多能进行多少次操作。$1 \leq n \leq 10^5$。

解法　最后我们得到的序列中，每个区间原来一定对应着一段 $a$ 的区间内的区间。而可以发现这个合并区间的方式就是可分离排列合并的方式，最终序列中每一个区间对应着原来 $\pi$ 中一段可分离的子区间。

而一个可分离排列的子区间一定是可分离的，因此求问题答案的方式就是每次选择一段极长的可分离前缀，然后将其删除，最后用 $n$ 减去删除次数即为答案。

将可分离排列转化为 2413, 3142-避免排列，然后用前文中区间排列模式匹配问题的算法，求出所有极小的包含 2413 和 3142 的区间，就容易维护每次删前缀的操作，复杂度 $O(n \log n)$。 ■

**例题 6.3** (缺失的子序列[3])**.** 有 $n$ 个集合 $S_1, S_2, \cdots, S_n$，每个集合都是 $\{1, 2, \cdots, n\}$ 的子集。求有多少长为 $n$ 的排列 $\sigma$ 使得 $\forall i \in [1, n], \sigma(i) \in S_i$，且 $\sigma$ 是 2413, 3142-避免的。答案对 $10^9 + 7$ 取模，$n \leq 100, \sum n^3 \leq 5 \cdot 10^6$。

解法　使用可分离排列的性质，我们可以将这样的 $\sigma$ 不断划分下去，使得最后分到的每个大小为 1 的小格 $(x, y)$ 都满足 $y \in S_x$。设 $f_{x,y,d}$ 表示我们需要在 $[x, x + d - 1] \times [y, y + d - 1]$ 这个矩形内划分出 $d$ 个格子，形成可分离排列的方案数。

一个可分离排列，分离它的过程要么第一步必须是左上-右下，要么第一步必须是左下-右上，不存在一个排列同时可以应用这两种划分。转移时枚举分界点 $p$，以及划分的方向。为了避免同一方向内算重，我们总是取最靠左的合法划分，例如如果我们要将矩形分割为

[2] https://qoj.ac/problem/9865

[3] https://acm.hdu.edu.cn/showproblem.php?pid=8044

左下-右上的两个矩形 $[x, x+i-1] \times [y, y+i-1]$ 和 $[x+i, x+d-1] \times [y+i, y+d-1]$，那么应当满足 $[x, x+i-1] \times [y, y+i-1]$ 为单点，或者下一步必须使用左上-右下分离。这个可以额外设两个状态 $g_{x,y,d}, h_{x,y,d}$ 限定第一步分离的方向解决。

总复杂度 $O(n^4)$，常数较小。 ■

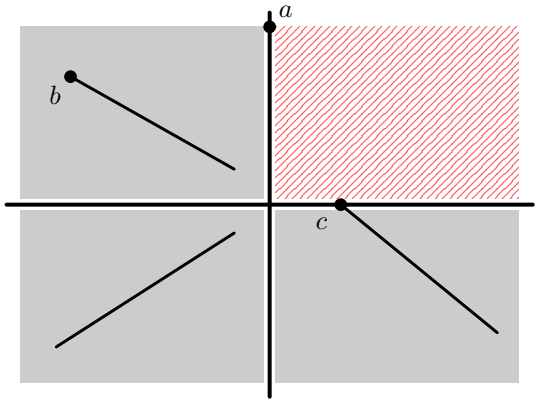
**例题 6.4** (Decinc Dividing[4]). 给定一个长为 $n$ 的排列 $\pi$，求有多少 $1 \le l \le r \le n$ 满足 $\pi(l:r)$ 的元素可以被拆成一个上升子序列和一个下降子序列（可以为空）。$n \le 2 \cdot 10^5$。

解法

**定理 6.5.** 一个排列可以被拆成一个上升子序列和一个下降子序列当且仅当其是 *2143, 3412*-避免的。

证明. 如果 $\pi$ 包含 2143 或 3412 子序列，容易发现这个子序列无法拆分成一个上升子序列和一个下降子序列。

如果 $\pi$ 不包含 2143 或 3412 子序列，我们归纳证明 $\pi$ 一定满足条件。设 $n = |\pi|$，$n = 1$ 时命题显然。当 $n > 1$ 时，找到满足 $\pi(a) = n$ 的位置 $a$，以及满足 $\pi(b) = n-1$ 的位置 $b$，不妨设 $b < a$。设从 $\pi'$ 为从 $\pi$ 中删除掉 $\pi(a)$ 得到的排列。那么根据归纳假设，$\pi'$ 可以拆为一个上升子序列和一个下降子序列：




- 如果上升子序列末尾在 $a$ 之前，那么我们直接在上升子序列后面加上 $\pi(a)$ 即可。
- 如果上升子序列末尾在 $a$ 之后，由于 $b, a$ 的存在，$a$ 右侧的序列一定递降，否则形成 3412 子序列。

  找到 $\pi(a+1:n)$ 中最大的元素 $\pi(c)$，那么 $\pi(1:a-1)$ 中值小于 $c$ 的所有元素一定递增，否则会和 $a, c$ 形成 2143 子序列。

  由于 $\pi'$ 的拆分中上升子序列末尾在 $a$ 右侧，因此 $\pi(1:a-1)$ 中大于 $\pi(c)$ 的元素一定都属于递降序列，也即这些元素单调递减。

[4]https://codeforces.com/problemset/problem/1693/D

此时我们将 $\pi(1:a-1)$ 中值小于 $\pi(c)$ 的元素与 $\pi(a)$ 组合成上升序列，其余元素必定形成下降序列。

□

因此该问题即为求 2143, 3412-避免子区间数量，使用前文区间模式匹配问题中的算法即可。复杂度 $O(n\log n)$。 ■

## 7 总结

本文总结了一些关于短排列模式匹配的经典问题及一些解决的算法，并收集了一系列可以转化为排列模式匹配问题的一些经典性质及例题，并且补充了信息学竞赛中相关结论证明缺失的一些空白。文中提到的许多问题依然有进一步探索的可能。笔者希望本文起到抛砖引玉的作用，让读者更好地认识排列匹配问题，进一步发现更多更优秀的算法和结论。

## 8 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢家人、朋友、教练对我一直以来的关心、鼓励与支持。
感谢陈旭磊同学、赵海鲲同学、武林同学为本文勘误并提出宝贵意见。
感谢其他给予我帮助的老师与同学。

## 参考文献

[1] Sylvain Guillemot, Daniel Marx. (2013). Finding small patterns in permutations in linear time.

[2] Chaim Even-Zohar, Calvin Leng. (2020). Counting Small Permutation Patterns.

[3] Gal Beniamini, Nir Lavee. (2024). Counting Permutation Patterns with Multidimensional Trees.