2023 年信息学奥林匹克
中国国家集训队论文

教练：杨耀良、彭思进

目 录

综述图论中连通性及相关问题的一些处理方法

华东师范大学第二附属中学 万成章

摘 要

本文分成无向图和有向图两个部分，讨论了在信息学竞赛中实用的一些有关连通性的算法思想，此外还探讨了一些和连通性有一定联系的图论问题。

1 定义与约定

对于无向图或有向图 G：

默认记 V 表示 G 的点集，E 表示 G 的边集，$n = |V|$ 表示 G 的点数，$m = |E|$ 表示 G 的边数。

若 E 中没有重复元素，则称 G 无重边；若任何 $(u,u) \notin E$，则称 G 无自环；

称 G 为简单图，当且仅当 G 无重边且无自环；

记 $u \to v$ 表示一条边 (u,v)；

记 $u \rightsquigarrow v$ 表示一条以 u 开始到 v 结束的路径，即 $u = x_1 \to x_2 \to \ldots \to x_k = v$；

一个环指的是一条形如 $u \rightsquigarrow u$ 的路径；

简单路径指的是不经过重复结点的路径；

简单环指的是除了起点和终点外不经过重复结点的环；

称 $G' = (V', E')$ 为 G 的子图，当且仅当 $V' \subseteq V, E' \subseteq E$；

G 关于点集 V' 的（点）导出子图为 $G' = (V', E'), E' = \{(u,v) \in E \mid u, v \in V'\}$；

G 关于边集 E' 的（边）导出子图为 $G' = (V', E'), V' = \{x \in V \mid (\exists y)((x,y) \in E' \vee (y,x) \in E')\}$。

2 无向图

无向图中的连通性是一个很容易理解的概念，若存在路径 $u \rightsquigarrow v$，则称 u, v 是连通的（就以记号 $u \rightsquigarrow v$ 表示 u, v 连通），这是一个等价关系。接下来我们定义一些后续将会用到的概念。

定义 2.1 (点/边割集)**.** 对于无向图 $G=(V,E)$ 以及 $u,v\in V$：若 $S\subseteq V$ 满足 $u,v\notin S$，且 G 关于 $V\backslash S$ 的导出子图上 $u\not\rightsquigarrow v$，则称 S 是 u,v 的点割集；若 $T\subseteq E$ 满足 $G'=(V,E\backslash T)$ 中 $u\not\rightsquigarrow v$，则称 T 是 u,v 的边割集。

定义 2.2 (点/边连通度)**.** 对于无向图 $G=(V,E)$ 以及 $u,v\in V,\ u\neq v$，若 u,v 的任意点割集大小不小于 s，则称 u,v 是 s-点连通的；若 u,v 的任意边割集大小不小于 t，则称 u,v 是 t-边连通的。u,v 间的点/边连通度为上述 s,t 的最小值。G 的点/边连通度是任意点对间点/边连通度的最小值。

例如，1-点连通和 1-边连通是等价的，它们都与一般意义上的连通等价。

定理 2.1 (Menger)**.** 对于无向图 $G=(V,E)$ 以及 $u,v\in V,\ u\neq v$：

u,v 是 k-边连通的，当且仅当存在 k 条 $u\rightsquigarrow v$ 的两两边不交的路径；

u,v 是 k-点连通的，当且仅当存在 k 条 $u\rightsquigarrow v$ 的除端点外两两点不交的路径。

这实际上就是最大流最小割定理的一个特化版本，由于证明与本文关系不大，因此省略。值得注意的是在 Menger 定理中我们不要求路径是不同的，包含两个点的完全图 K_2 对于任意的 k 是 k-点连通的，但是其中只存在一条简单路径。

2.1 DFS 树

我们首先考虑最简单的连通，即 1-连通。

对于一个连通的无向图 G，必定可以找到其一个包含所有顶点的无环子图 T，将这样的 T 称为 G 的生成树。

定义 2.3 (DFS 序，DFS 树)**.** 在连通无向图 $G=(V,E)$ 上以 $r\in V$ 为起点进行深度优先搜索，若结点 x 是第 p 个被访问到的结点，则称 p 为 x 的 *DFS* 序，记为 $dfn_x=p$。若 $x\neq r$，则在 x 与第一次访问到 x 的结点 y 间连一条边，形成的图 T 称为以 r 为根的一棵 *DFS* 树。

上述的 T 有 $n-1$ 条边，且可以归纳地说明 T 连通，因此 T 确实是 G 的一棵生成树，一般将它看作以 r 为根的有根树，将 T 上的边称为树边，其余边称为非树边。

需要注意，DFS 序和 DFS 树是一一对应的，同一个 r 可能拥有不同的 DFS 序和 DFS 树。

性质 2.1.1. 所有结点的 *DFS* 序构成 $1,\ldots,n$ 的一个排列。*DFS* 树上以任何结点 x 为根的子树中所有点的 *DFS* 序构成一段以 dfn_x 开始的连续区间。

性质 2.1.2. 任意一条非树边连接了 *DFS* 树上一对祖先-后代结点，这样的边称为返祖边。

证明只需考虑 DFS 过程的性质即可。

性质 2.1.2 是 DFS 树的核心性质，这也是许多连通图相关的问题考虑用 DFS 树解决的原因。

例 2.1.1 (Ehab's Last Theorem[1])**.** 对于 n 个点的无向连通图 G，以下两个命题至少有一个成立：

1. G 有一个长度不小于 $\sqrt{n}$ 的简单环。

2. G 有一个大小不小于 $\sqrt{n}$ 的独立集。

证明. 令 $p = \lceil \sqrt{n} \rceil$，接下来我们将给出一个构造性的证明。

首先我们求出 G 的一棵以 1 为根的 DFS 树，设 d_x 表示 x 在 DFS 树上的深度。

考虑任意非树边 $x \to y$，其中 x 是后代，则如果 $d_x - d_y \geq p - 1$，则我们已经找到一个长度至少为 p 的简单环，即由树上路径 $x \rightsquigarrow y$ 加上非树边 $x \to y$ 构成的简单环。

假设不存在这样的环，那么对于任意非树边 $x \to y$ 有 $|d_x - d_y| < p - 1$，我们通过如下过程可以构造一个大小至少为 p 的独立集：

- 定义点集 S，初始为空。
- 找到当前未被标记的深度最大的任意一个结点 x，令 $S \leftarrow S \cup \{x\}$，然后将 x 的 $0, 1, 2, \ldots, p-2$ 级祖先都进行标记。
- 重复上述过程，直到所有结点被标记，S 即为目标独立集。

注意上述过程中每一轮最多标记 $p-1$ 个结点，所以 $|S| \geq \frac{n}{p-1} > p - 1$，同时每次选出的 x 只有可能与其子树内的结点或者其 $0, 1, \ldots, p-2$ 级祖先之间可能有边，x 的选取方式（深度最大）保证了子树内的点都被标记，同时其 $0, 1, \ldots, p-2$ 级祖先也被标记，所以任意时刻与 S 中结点有边相连的结点一定被标记了，这保证了 S 确实是独立集。

从而原命题得证。

□

对于不连通的无向图，我们可以对于其每个连通分量各选择一个点为根构建 DFS 树，这就形成了一个 DFS 森林，同样可以套用 DFS 序等概念，同样可以使用上面介绍的性质。例如例 2.1.1 其实对于不连通的图也是成立的，证明和连通图的情形相似。

2.2 双连通性

接下来我们来研究比 1-连通稍复杂一些的 2-连通性，或称双连通性。

由于双连通性在 OI 中是相对来说比较熟知的，因此本文在基础部分会简略一些，不过仍会完整地说明会用到的所有性质，此部分内容在参考资料 [1] 中也有全面的描述。

[1] CF 1325 F

2.2.1 边双连通

由于边双连通性比点双连通性直观一些，因此首先介绍边双连通性。

边双连通的点对必定是 1-连通的，故不同连通块独立，因此先考虑 G 是连通图的情形。

我们首先要说明的是，边双连通性（可以推广到任意的 k-边连通性）是点集 V 上的一个等价关系，也就是说我们可以按照边双连通性对 V 进行划分。

定义 2.4 (割边)**.** 对于无向连通图 $G = (V, E)$，若 $e \in E$ 满足 $G' = (V, E\backslash\{e\})$ 不连通，则称 e 是一条割边或桥。对于不一定连通的无向图 G，其任意一个连通分量的割边都称为其割边。

一种等价的描述是：如果 $\{e\}$ 是某个点对 (u, v) 的边割集，则 e 是割边。

引理 2.2.1. 对于无向连通图 G，割边一定在 *DFS* 树上。设某条割边 e 为 $x \to y$，其中 x 为子结点，则删除 e 后会形成两个连通分量，分别是 x 的子树和 x 的子树的补。

引理 2.2.2. 设无向连通图 $G = (V, E)$ 的割边集合为 E'，$G' = (V, E\backslash E')$。则对任意的 $u, v \in V$, $u \neq v$，u, v 在 G 中边双连通当且仅当 u, v 在 G' 中连通。

证明. 若 u, v 在 G' 中连通，假设它们不是边双连通的，则存在一条边 e 使得删去 e 后 u, v 不连通，则 e 必然是割边，然而删除全部割边的 G' 中 u, v 仍是连通的，矛盾。因此 u, v 边双连通。

若 u, v 在 G' 中不连通，考虑树上路径 $u \rightsquigarrow v$，一定存在路径上一条边 e 使得 e 是割边（否则 u, v 就连通了），根据引理 2.2.1，删去这条边后 u, v 不在同一连通分量，因此 u, v 不边双连通。 □

至此我们就可以定义：

定义 2.5 (边双连通分量)**.** 无向图 G 中的结点按照边双连通关系可以划分为若干个等价类，每个等价类的导出子图称为 G 的一个边双连通分量。

边双连通分量的存在性由引理 2.2.2 保证：只需要将 G 的所有割边删去得到 G'，则 G' 中的所有连通块就是 G 的边双连通分量。

我们可根据引理 2.2.2 求出所有的边双连通分量，利用一些数据结构技巧可以做到 $O(n+m)$ 的时间复杂度。不过 Tarjan 算法是一种同样复杂度却更加简洁的算法。

考虑怎样的边会成为割边，设 DFS 树边 $e = (x, y)$，其中 x 为子结点。那么 e 是割边当且仅当不存在 x 子树中的点到 x 的祖先（但不包括 x）的边（注意非树边都是返祖边）。于是我们可以求出 low_x 表示：从 x 子树中的任意一个点出发，只能通过至多一条非树边能够到达的结点的最小 DFS 序。这可以通过一个树形 DP 求出：

$$low_x = \min\left(dfn_x, \min_{y\in \mathrm{Son}(x)}(low_y), \min_{(x,y)\in E}(dfn_y)\right)$$

满足 $low_x = dfn_x$ 的点是一个边双连通分量的最浅点，当我们 DFS 过程中发现这样的点时，其子树中未被删除的部分就构成一个边双连通分量，随后将这个子树删除即可。

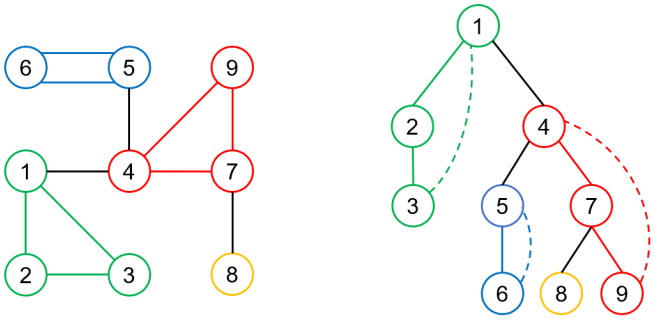


图 1: 一张无向图和它的不同边双连通分量

2.2.2 点双连通

点双连通关于点并不是一个等价关系，不过我们仍可以从 DFS 树开始研究。

定义 2.6 (割点)**.** 对于无向连通图 $G = (V, E)$，若 $u \in V$ 满足 G 关于 $V\backslash\{u\}$ 的导出子图不连通，则称 u 是一个割点。对于不一定连通的无向图 G，其任意一个连通分量的割点都称为其割点。

割点同样有等价的描述：如果 $\{u\}$ 是某个点对的点割集，则 u 是割点。

回忆 Tarjan 算法中定义的 *low* 数组：low_x 表示 DFS 树上从 x 子树内任意一点出发经过至多一条非树边能够到达的结点的最小的 DFS 序。我们可以用它来求解割点。

引理 2.2.3. 无向连通图 $G = (V, E)$ 中，x 是割点当且仅当以下两个条件至少满足一个：

1. x 是 *DFS* 树的根结点且有至少两个子结点；

2. x 是 *DFS* 树的非根结点，且存在一个子结点 y 满足 $low_y \geq dfn_x$。

证明. 第一个条件是比较显然的。

考虑第二个条件，若存在这样的子结点 y，则 y 的子树中没有通向 x 的祖先（但不包括 x）的返祖边。删去 x 后 y 的子树与外界不连通，单独形成了一个连通分量，则 x 必是一个割点。

反之，如果非根结点 x 是割点，设 V' 为 x 的子树关于 V 的补集，则删去 x 后至少有一个 x 的子结点 y 与 V' 不连通，这就表示 y 子树中任何一个点都没有通往 V' 的返祖边，即 $low_y \geq dfn_x$。 □

定义 2.7 (点双连通分量)**.** 无向图 $G = (V, E)$ 中，若点集 $V' \subseteq V$ 满足 V' 的导出子图 G' 点双连通，且对于任何 $V' \subset V'' \subseteq V$，$V''$ 的导出子图不点双连通，则称 G' 是 G 的一个点双连通分量。

Tarjan 算法可用于求解所有点双连通分量，方法与求边双连通分量有些类似：进行树形 DP 求出 low_x，若对于一对父子 (x, y) 满足 $low_y \geq dfn_x$，则取出 y 子树内未被删除的部分，与 x 共同组成一个点双连通分量。随后我们将 y 子树内未被删除的部分进行删除。时间复杂度 $O(n + m)$。

根据引理 2.2.3 及其证明，不难说明 Tarjan 算法的正确性。不过目前我们并不掌握点双连通分量的太多性质。在 2.2.1 节中我们根据对 V' 的划分定义了边双连通分量，那么点双连通分量是否也有类似的形式呢？答案是肯定的，不过我们需要将研究对象从点转移到边上。接下来我们就给出一种从边的视角刻画点双连通分量的方法。在本节剩余的讨论中，我们均默认图是无自环的。

引理 2.2.4. 对于点双连通图 G 中任意不同两点 u, v，存在经过 u, v 的简单环。

证明. 这是 Menger 定理的直接推论。 □

引理 2.2.5. 对于点双连通图 G 中任意一点 u 和一条边 e，存在经过 u, e 的简单环；对于任意两条边 e_1, e_2，存在经过 e_1, e_2 的简单环。

证明. 我们将一条边 $x \to y$ 拆成 $x \to z$ 和 $z \to y$ 两条边，显然新图仍是点双连通的，但此时边就被转化成了点，直接应用引理 2.2.4 即得证。 □

定义 2.8. 无向图 G 上的两条边 e_1, e_2，若存在一个简单环同时包含 e_1, e_2，则称它们是共环的。

引理 2.2.6. 共环是等价关系，每个点双连通分量中的边是共环意义下的一个等价类。

证明. 首先，同一点双连通分量中的任意两条边共环，这是由引理 2.2.5 得到的。

对于不同点双连通分量中的两条边 $e_1 = (x, y)$, $e_2 = (u, v)$，假设存在一个简单环同时包含 e_1, e_2，那么这个环上的所有点一定是点双连通的，这与 e_1, e_2 属于不同点双连通分量矛盾。 □

注意，无论在点双连通分量的定义还是在上面的证明中，我们都没有证明一个基本的事实：每条边只属于一个点双连通分量。不过事实上根据定义或 Tarjan 算法都可以简单地说明任意两个点双连通分量交集大小至多为 1，因此每条边确实只在一个点双连通分量中。

有了引理 2.2.6，我们就可以像看待边双连通分量一样看待点双连通分量，只不过边双连通分量是对点集的划分，而点双连通分量是对边集的划分。

2.2.3 圆方树

圆方树是用来描述无向图点双连通分量结构的一种数据结构。

定义 2.9 (圆方树)**.** 对于至少有 2 个结点的无向连通图 $G = (V, E)$，建立一张新的无向图 T，每个结点对应 G 的一个结点或点双连通分量，两个结点之间有边当且仅当它们分别代表一个 G 中的结点 x 和一个 G 的点双连通分量 $G' = (V', E')$，且 $x \in V'$。这 T 叫做 G 的圆方树。T 中对应原图上结点的结点称为圆点，对应原图点双连通分量的结点称为方点。

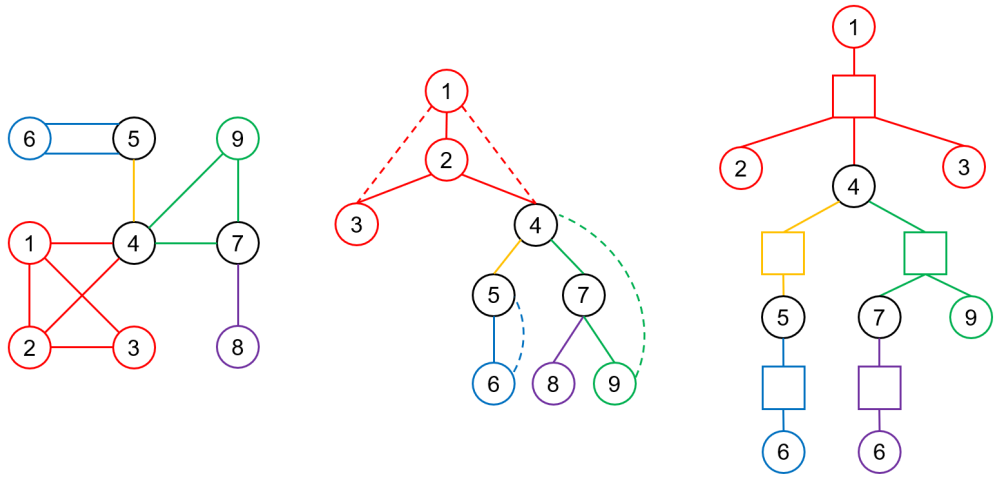


图 2: 一张无向图和它的不同点双连通分量、圆方树

考虑一条长度为 k 的路径 $u \rightsquigarrow v$，它会依次经过它的每条边所属的点双连通分量，因此它就对应于圆方树上的长度为 $2k$ 的路径 $u \rightsquigarrow v$，除了原来路径上的点外还依次经过每条边所属点双连通分量对应的方点。因此，连通图的圆方树是连通的；同时，圆方树一定是无环的，否则环可以缩成一个点双连通分量。于是，圆方树确实是树。对于不一定连通的无向图 G，可以定义它的“圆方森林”，即对每个连通分量各建圆方树。

性质 2.2.1. 圆方树是二分图，两部分别是圆点和方点。

性质 2.2.2. 一个点是割点当且仅当它在圆方树上（对应的圆点）度数大于 1。圆方树上的叶结点（包括度数为 1 的根）必定是圆点，并且它们就是全部的非割点。

性质 2.2.3. 一张 $n \geq 2$ 个点的无向图的全部点双连通分量的点数之和不超过 $2n-2$。

证明. 在圆方树上以某个圆点为根，然后我们为每个方点指定一个子结点，这些子结点都是圆点且一定不会有重复，因此方点的数量不超过非根的圆点的数量，即 $n-1$。

因此圆方树的总点数不超过 $2n-1$，总边数不超过 $2n-2$，而每个点双连通分量大小之和就是方点的度数和，即圆方树的总边数，于是得证。 □

性质 2.2.4. 无向连通图 G 上两个结点 $u, v \in V$ 间的任何简单路径，一定经过且只经过在它们圆方树路径上的所有方点对应的点双连通分量（所包含的边），一定经过它们圆方树路径上的所有圆点（对应的原图的点）。

圆方树在 OI 中是一种很重要的结构，在许多题目中有着应用，下面试举几例。

例 2.2.1 (仙人掌)**.** 每条边只在至多一个简单环上的无向连通图叫做（边）仙人掌。许多树上可以完成的操作可以通过圆方树被搬到仙人掌上进行。

考虑在仙人掌上进行 Tarjan 算法，我们会得到一棵 DFS 树，并且每条非树边对应的树上路径两两边不交，每条非树边就对应了一个简单环。因此 Tarjan 算法的过程可以直接帮我们找到所有环，这是很方便的。例如我们需要在仙人掌上进行 DP 时，就可以在圆方树（或 DFS 树）上自底向上进行，每找到一个环就在环上进行 DP，否则进行树上 DP。

而当我们需要在仙人掌上进行路径修改查询等操作时，一般就需要将仙人掌上路径转化为圆方树的树上路径，通过一些讨论将问题转化为树上问题解决。不过这类问题一般代码难度较高，在目前的 OI 题目中是很少见的。

例 2.2.2 (地地铁铁[2])**.** 给定无向连通图 $G = (V, E)$，边有黑白两种颜色，请求出有多少个无序点对 (x, y) 满足存在一条简单路径 $x \rightsquigarrow y$ 上同时有黑边和白边出现。

我们只需求出不满足条件的点对 (x, y)，而这又可以分成三种：

1. 任何简单路径 $x \rightsquigarrow y$ 都只经过黑色边；

2. 任何简单路径 $x \rightsquigarrow y$ 都只经过白色边；

3. 分别存在简单路径 $x \rightsquigarrow y$ 只经过黑色边和白色边，但没有同时经过两种边的简单路径。

我们首先来看情况 1。根据圆方树的性质 2.2.4 我们已经知道，只需要考虑位于圆方树上路径 $x \rightsquigarrow y$ 上的所有方点中的边即可，因为其他边都是 x 到 y 的简单路径不可能经过的。

[2]Luogu P8456，SWTR-8

我们称一个方点是纯黑的，当且仅当其（对应的点双连通分量）中所有边都是黑的，纯白定义类似。那么我们断言，情况 1 成立当且仅当圆方树上路径 $x \rightsquigarrow y$ 上的所有方点都是纯黑的。

引理：点双连通图中，任意给定两点 x, y $(x \neq y)$ 和一条边 e，存在经过 e 的简单路径 $x \rightsquigarrow y$。

上述引理的证明只需回忆引理 2.2.5，我们首先找到一个包含 x, e 的环 C，然后选择一条 y 到 x 的简单路径 P，满足 P 与 C 的点集交大于 1（注意 x 一定在交中，因此一定能找到这样的 P，否则 x 是割点）。设它们的点集交中在 P 上最靠近 y 的点为 u，那么选择包含 e 的环上路径 $x \rightsquigarrow u$ 并上 P 的前缀 $u \rightsquigarrow y$，就得到了 $x \rightsquigarrow y$ 的经过 e 的简单路径。

因此如果存在一个路径上的方点不是纯黑的，那么我们走到那个点双连通分量时随意通过一条其中的白色边，就与情况 1 矛盾了；同时如果路径上每个方点都是纯黑的，显然 x 到 y 的任何简单路径都是纯黑的。这证明了我们的断言。

情况 2 与情况 1 是类似的。

情况 3 看上去稍微复杂一些。我们首先需要一个引理。

引理：如果 (x, y) 满足条件 3，那么任取两条分别是纯白和纯黑的 x 到 y 的简单路径 P_1, P_2，它们除端点外是点不交的。

引理的证明并不困难：假设两条路径除端点外还是点相交的，则可以分别取一个前缀和后缀构成一条新的路径，但新的路径并不是纯黑或纯白的，与情况 3 的描述矛盾。

这个引理告诉我们，x, y 一定在同一个点双连通分量中（否则它们间的任意路径都会经过某个割点）。此外，假设 x, y 所在的点双连通分量中，除了 x, y 外还存在一个点 u 满足有一黑一白的两条边 e_1, e_2 与之关联，则 x, y 一定不满足情况 3。这是因为我们可以找到两条 $x \rightsquigarrow y$ 的简单路径，分别经过 e_1, e_2，而它们点相交于 u，与引理矛盾。

由上面的分析，每个点双连通分量中至多有一对满足情况 3 的 (x, y)，只要找到同时有黑白边与之关联的那些结点即可。

三种情况分析完毕，剩余的部分只是计算这些点对的数量，这部分比较简单，从略。

例 2.2.3 (Tom & Jerry[3])**.** 在一张无向连通图 G 上，*Tom* 追逐 *Jerry*。每一回合由 *Jerry* 先行动，*Tom* 后行动。*Jerry* 每次行动可以经过任意多条边，但不能经过 *Tom* 所在结点；*Tom* 每次行动只能经过至多一条边，若行动结束后位于 *Jerry* 所在位置则 *Tom* 获胜。现在 q 次给出 (a, b)，问当 *Tom* 和 *Jerry* 初始分别位于 a, b 两点时，*Tom* 能否在有限时间内获胜。

定义点对 (x, y) 是好的，当且仅当圆方树上路径 $x \rightsquigarrow y$ 上任意两个相邻的圆点在原图 G 上也相邻。

主结论：Tom 能获胜，当且仅当以下两个条件成立至少一个：

[3]Luogu P7353，本题是笔者为 IOI2021 精英集训作业命制的一道题目。

1. 存在一个点 u，使得对于任意的 x，(u,x) 是好的；

2. 对于删去 a 后 b 所在的连通块中任意一点 x，(a,x) 是好的。

首先证明充分性。两个条件实际上是类似的，我们以条件 2 为例。Tom 只需每次沿着圆方树上路径向 Jerry 靠近一步即可（这里的可行性就是由条件 2 保证的），由圆方树的性质可知 Jerry 始终只能在 Tom 所在点的一个分支中行动，因此 Tom 和 Jerry 最终会相遇，Tom 获胜。

再考虑必要性。假设两个条件都不满足，那么 Jerry 第一步可以走到一个点 x 使得 (a,x) 不是好的，随后在原地等待，直到 Tom 走到了一个不在 $a \rightsquigarrow x$ 树上路径上的点 y。根据第一个条件不成立，我们知道整张图上一定存在一个点 z 使得 (y,z) 不是好的。假设树上路径 $z \rightsquigarrow x$ 不经过 x，那么此时 Jerry 直接从 x 走到 z 即可；否则，Jerry 可以在 Tom 走到 y 的上一步走到 z。总之，现在又回到了 Tom 所在点和 Jerry 所在点构成的点对不是好的情形，于是 Tom 永远是追不上 Jerry 的。

在有了主结论的基础上加以圆方树上 DP 即可完成本题，不过后续部分与本文无关，从略。

2.2.4 耳分解

耳分解是对双连通图的另一种刻画，在一些问题上可以给我们提供一些不同的视角。

定义 2.10 (耳、开耳)**.** 在无向图 $G=(V,E)$ 中，有一个子图 $G'=(V',E')$，若简单路径或简单环 P：$x_1 \to x_2 \to \ldots \to x_k$ 满足：$x_1, x_k \in V'$，$x_2, \ldots, x_{k-1} \notin V'$，则称 P 是 G 关于 G' 的耳。若 P 是简单路径，则称 P 是 G 关于 G' 的开耳。

定义 2.11 (耳分解、开耳分解)**.** 对于无向连通图 G，若连通图序列 $(G_0, G_1, \ldots, G_k)$ 满足：

1. G_0 是一个简单环（可以只有一个点），$G_k = G$；

2. G_{i-1} 是 G_i 的子图；

3. 设 $G_i=(V_i,E_i)$，则 $E_i \backslash E_{i-1}$ 构成 G_{i-1} 的一个耳（开耳）。

则称 $(G_0, G_1, \ldots, G_k)$ 是 G 的一个耳分解（开耳分解）。

定理 2.2. 无向连通图 G 存在耳分解当且仅当 G 边双连通。

证明. 先证必要性。若 G 有耳分解 $(G_0, \ldots, G_k)$，则由于 G_0 是简单环，G_0 一定是边双连通的。若 G_i 是边双连通的，则在 G_i 的基础上加一条路径构成的 G_{i+1} 也是边双连通的（因为任何边都不可能是割边）。利用数学归纳法知 $G = G_k$ 是边双连通的。

再证充分性。若 $G=(V,E)$ 是边双连通的，如果它不含任何边，则结论显然。否则，我们求出它以 1 为根的一棵 DFS 树，然后按照如下方法得到 G 的一个耳分解：

1. 找到一条以 1 为端点的非树边 $1 \to x$，令 G_0 是由树上路径 $1 \rightsquigarrow x$ 加上这条非树边构成的简单环。

2. 若现在已经生成了 G_i，如果 G_i 的点集不为 V，找到一个不属于 G_i 的点 x 满足 x 的父亲 y 属于 G_i，则由边双连通性知存在一条返祖边，两端点分别是 y 的祖先 u 和 x 的后代 v，然后我们选择树上路径 $y \rightsquigarrow v$ 并上边 $v \to u$，这是 G_i 的一个耳，令 G_{i+1} 为 G_i 加上这个耳。

3. 重复上述过程直到 G_i 的点集为 V，此时若还有 G 中的边未加入 G_i，则可以每条边作为一个耳依次加入。

第二步中，我们始终保证 G_i 的点集是个包含 1 的树上连通块，因此这样的 x 总能找到。 □

例 2.2.4 (Quare[4])**.** 给定带边权无向边双连通图 G，求包含所有点的边双连通子图的最小边权和。

直接考虑边双连通图的形态是比较困难的，但是从耳分解的角度就比较容易处理。

设 $f(S)$ 表示将集合 S 中的点连成一个边双连通图的最小边权和，那么转移为 $f(S) + E(S, T \backslash S) \to f(T)$，其中 $E(S,R)$ 表示由 R 中的点和 S 中的两个端点构成的耳的最小边权和。$E(S,R)$ 的计算可以通过枚举耳上除端点外的最后一个点来计算，时间复杂度为 $O(2^n \times \mathrm{Poly}(n))$，但是计算 f 数组时我们需要枚举子集，总复杂度为 $O(3^n \times \mathrm{Poly}(n))$。

这是可优化的，我们不计算 $E(S,R)$，转而直接在计算 f 数组时枚举耳上点的顺序，每次只在耳的最后加一个点，而不是加入整个耳。具体地，令 $f(S,i,j)$ 表示已经将集合 S 中的结点连成边双连通图，现在正在加入一个耳，这个耳已经从一端开始延申到 i，钦定耳的另一个端点是 $j \in S$。在转移时只需要枚举耳上 i 的下一个点即可。时间复杂度为 $O(2^n \times \mathrm{Poly}(n))$。

定理 2.3. 至少含有三个点的无自环无向连通图 G 存在开耳分解当且仅当 G 点双连通。

证明. 证明总体和定理 2.2 类似。

先证必要性。若 G 有开耳分解 $(G_0, \ldots, G_k)$，则由于 G_0 是简单环，G_0 一定是点双连通的。若 G_i 是点双连通的，则在 G_i 的基础上加一条路径构成的 G_{i+1} 也是边双连通的（因为任何边都不可能是割边）。利用数学归纳法知 $G = G_k$ 是点双连通的。

再证充分性。若 $G = (V,E)$ 是点双连通的，我们求出它以 1 为根的一棵 DFS 树，然后按照如下方法得到 G 的一个开耳分解：

1. 找到一条以 1 为端点的非树边 $1 \to x$，令 G_0 是由树上路径 $1 \rightsquigarrow x$ 加上这条非树边构成的简单环。

2. 若现在已经生成了 G_i，如果 G_i 的点集不为 V，找到一个不属于 G_i 的点 x 满足 x 的父亲 y 属于 G_i，则由点双连通性知存在一条返祖边，两端点分别是 y **的父亲**的祖先 u 和 x

[4]SNOI 2013

的后代 v，然后我们选择树上路径 $y \rightsquigarrow v$ 并上边 $v \to u$，这是 G_i 的一个**开耳**，令 G_{i+1} 为 G_i 加上这个开耳。

3. 重复上述过程直到 G_i 的点集为 V，此时若还有 G 中的边未加入 G_i，则可以每条边作为一个开耳依次加入（这里用到了无自环的条件）。

第二步中，我们始终保证 G_i 的点集是个包含 1 的树上连通块，因此这样的 x 总能找到。 □

例 2.2.5 (双极定向（Bipolar Orientation）). 给定无向图 $G = (V, E)$ 和不同的两个结点 s, t，以下四个命题等价：

1. 在添加无向边 $s \to t$ 后，G 点双连通；

2. G 的圆方树中所有方点构成一条链，$s \rightsquigarrow t$ 是圆方树的一条直径；

3. 存在一种对 G 的边进行定向的方法，得到一个有向无环图，且 s 入度为零，t 出度为零，其余点出入度都不为零。

4. 存在一个所有点的排列 $p_1, p_2, \ldots, p_n$，使得 $p_1 = s$, $p_n = t$，且任意前缀以及任意后缀的导出子图都是连通的。

证明. 我们先证命题 1,2 和命题 3,4 分别等价，再证这两对命题也等价。

第 1,2 个命题的等价是比较显然的。

考虑第 3,4 个命题。

如果命题 3 成立，则我们任取定向后的一个拓扑排序作为命题 4 中的 p。由命题 3 的描述知除 s 外每个点都存在一个拓扑序比它小的邻居，从而对任意的 p_i，一定存在一个邻居在 $p_1, \ldots, p_{i-1}$ 中，于是由 $p_1, \ldots, p_{i-1}$ 导出子图连通可以推出 $p_1, \ldots, p_i$ 导出子图连通，归纳即得任意前缀的导出子图连通，同理任意后缀的导出子图连通。如果命题 4 成立，则我们令每条边的方向是从 p 中排在靠前的点连向靠后的点，容易验证命题 3 成立。

接下来我们由命题 1 推命题 3。连接 s, t 后我们求出 G 的一个开耳分解 $(G_0, \ldots, G_k)$，满足 G_0 包含边 $s \to t$（回忆定理 2.3 的证明，我们可以在构造中做到这一点）。G_0 是一个简单环，显然可以将 $s \to t$ 以外的边都定向为从 s 到 t 的。接下来进行归纳，如果 G_i 已经定向完毕，现在向其中加入一个开耳得到 G_{i+1}，设开耳的两端点为 u, v。若 G_i 的定向中 u 可达 v，则我们令耳的方向为从 u 到 v，否则令耳的方向为从 v 到 u。这不会破坏图的无环性，并且除 s, t 外每个点在加入时都拥有了 1 的入度和出度，因此最终 $G = G_k$ 的定向满足命题 3。

最后我们由命题 4 推命题 1。假设命题 1 不成立，那么存在一个割点 u 满足删去 u 后 s, t 在同一连通块，于是一定还有一个不包含 s, t 的连通块，设为 S。设 $S \cup \{u\}$ 中在排列 p 里出现最早和最晚的点分别是 x, y，则 x, y 中至少有一个不是 u。不妨设 $x \neq u$，那么考察前缀 $p_1, \ldots, p_i = x$ 的导出子图，由于 u 不在其中而 s, x 都在其中，所以这张图一定不连通，与命题 4 矛盾。因此命题 4 可以推出命题 1。

结合以上，我们证明了四个命题的等价性，同时我们给出了由命题 1 到命题 3,4 的构造。 □

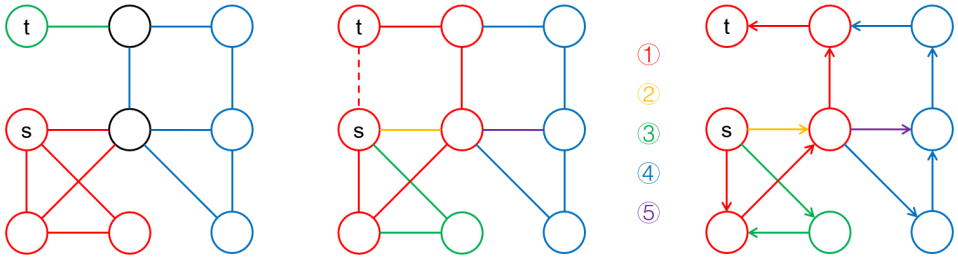


图 3: 一张无向图、连接 s,t 后的开耳分解与双极定向

2.3 割集与切边等价

2.3.1 割空间和环空间

之前我们定义了两点之间的点割集和边割集，但这里我们要说的割集意义不太一样，它是定义在点集 V 的一个二划分上的。

定义 2.12 (割集)**.** 在无向图 $G=(V,E)$ 中，对于 V 的一个二划分 $V=V_1 \uplus V_2$，定义 V_1,V_2 间的割集为 $\{e\in E \mid e=(x,y),\ x\in V_1,\ y\in V_2\}$，记为 $C(V_1,V_2)$，本文中也简记为 $C(V_1)$ 或 $C(V_2)$。

可以发现，如果 G 不连通，则 G 的割集可以拆分为 G 的各个连通分量的割集的并。因此我们基本只需要考虑连通图的割集。最简单的连通图是树，因此我们从树的割集出发进行研究。

引理 2.3.1. 对于树 $T=(V,E)$，任意边的子集 $E'\subseteq E$ 都是割集，且对应的划分 V_1,V_2 唯一。

证明. 对于任意的边的子集 $E'\subseteq E$，删去 E' 中的边后树将分裂成若干个连通分量，设这些连通分量构成的集合为 C。E' 中的边会将 C 中的元素连成树，设这个以 C 的元素为结点的树为 T'。

若 E' 确实是割集，则对于任意 $c\in C$，所有 c 中的结点必然在割集的同一侧，且对于 T' 上相邻的两个结点 $c_1,c_2\in C$，它们必然在割集的两侧。

于是我们得到了唯一可能的划分，即一部分是 T' 上深度为奇数的结点对应的连通分量中的所有点，另一部分是 T' 上深度为偶数的结点对应的连通分量中的所有点。同时容易验证 E' 确实是这个划分的割集，因此命题得证。 □

引理 2.3.2. 若连通图 $G=(V,E)$ 有一棵生成树 $T=(V,E_0)$，则对于任意 $E_0'\subseteq E_0$，G 存在唯一的割集 E' 满足 $E'\cap E_0=E_0'$，且对应的划分 V_1,V_2 唯一。

证明. 根据引理 2.3.1，由 E'_0 我们可以直接得出唯一的一组 V_1, V_2，于是对于所有非树边 e，若 e 的两端都在 V_1 或都在 V_2 中则 $e \notin E'$，否则 $e \in E'$，就得到了满足条件的唯一割集 E'。 □

我们还可以进一步分析 E' 中有哪些非树边。考虑非树边 e 的两个端点 x, y，若树上路径 $x \rightsquigarrow y$ 上有奇数条边在 E'_0 中，说明在引理 2.3.1 的证明中所说的 T' 上 x, y 所在连通分量的深度奇偶性不同，也就是说它们位于割集两侧，因此 $e \in E'$，否则 $e \notin E'$。也就是说，E' 恰好包含所有跨过奇数条 E'_0 中树边的那些非树边。

定理 2.4 (割空间)**.** 在无向图 $G = (V, E)$ 中，将任何一个边的集合看成 $\mathbb{F}_2$ 上的 m 维向量，则所有割集组成的集合是 $\mathbb{F}_2$ 上的线性空间，称为割空间。它的维数为 $n - c$，c 为 G 的连通分量个数。

证明. 先考虑连通图的情形。

要证明割集组成的集合是线性空间，只要证任意两个割集的对称差仍是割集。

若有割集 C_1, C_2，它们位于生成树上的子集分别是 E_1, E_2，则 $C_1 \oplus C_2$ 位于生成树上的子集就是 $E_1 \oplus E_2$。考虑一条非树边 e，若它跨过奇数条 $E_1 \oplus E_2$ 中的边，则根据对称差保持奇偶性的性质，可以得到 e 要么经过奇数条 E_1 中的边，要么经过奇数条 E_2 中的边，二者只居其一，这等价于 $e \in C_1 \oplus C_2$，因此 $C_1 \oplus C_2$ 确是割集。

要求空间的维数只需给出一组基，根据引理 2.3.2，对于生成树上的每条边 e 定义 $f(e)$ 为 e 与跨过 e 的非树边构成的边集，则所有的 $f(e)$ 构成割空间的一组基，因此割空间的维数为 $n - 1$。

当图不连通时，不同的连通分量彼此独立，完整的图的割空间等于各个连通分量割空间的直和，因此维数相加，算得 G 的割空间维数为 $n - c$。 □

无向图中还有另一种 $\mathbb{F}_2$ 上的线性空间，由于它的结构更加简单直观，而且讨论过程与割空间相似，因此这里不加证明地直接给出结论：

定理 2.5 (环空间)**.** 在无向图 $G = (V, E)$ 中，考虑所有满足每个点的度数都为偶数的子图 $G' = (V, E')$，其中的 E' 看成 $\mathbb{F}_2$ 上的 m 维向量，构成了一个 $\mathbb{F}_2$ 上的线性空间，称为 G 的环空间，维数为 $m - n + c$，c 为 G 的连通分量个数。

与割空间类似，环空间的一组基是由每条非树边导出的：对于每条非树边 $e = (x, y)$，它并上树上路径 $x \rightsquigarrow y$ 就构成一个环，这些环就是环空间的一组基。

定理 2.6. 同一张无向图的割空间与环空间互为正交补。

证明. 使用同一个生成森林，我们尝试证明由这个生成森林构造出的割空间和环空间的基中的元素两两正交。考虑一条树边 e_1 和一条非树边 $e_2 = (x, y)$，它们对应的割空间和环空

间的基中的元素分别是 v_1, v_2。若 e_1 不在树上路径 $x \rightsquigarrow y$ 上，则 v_1 和 v_2 对应的边集无交；若 e_1 在树上路径 $x \rightsquigarrow y$ 上，则 v_1 和 v_2 对应的边集交集只有 e_1, e_2。对于这两种情况都有 $v_1 \cdot v_2 = 0$。

同时由于两个空间的维数之和为 m，因此它们互为正交补。 □

例 2.3.1 (环覆盖[5])**.** 给定简单无向图 G，对于每个 $i \in [1, m]$ 求出环空间中对应边集大小为 i 的元素数量。

本题有两种主要的考虑方式。

解一：一种直接的做法。将每条边 $e = (x, y)$ 看成一个 $\mathbb{F}_2$ 上的 n 维向量，只有第 x, y 维为 1，则问题转化为有多少 i 元子集的和为零。

解二：找一个 G 的生成森林，然后考虑由森林导出的环空间的一组基。枚举某个元素是基的 x 元子集的和，并只保留树边对应的那些维（因为已经枚举了非树边的数量 x），问题转化为求基的 x 元子集中有多少个的和当中 1 的个数为 $i - x$。

两种转化直接使用 FWT 进行处理，复杂度均为 $O(n \times 2^n + \mathrm{Poly}(m))$。

在此基础上，解一可以使用 FWT 的定义将 FWT 转化为一个状压 DP，使用 $O(1)$ 求 popcount 和 lowbit 即可实现 $O(2^n + \mathrm{Poly}(m))$。

对解二的优化需要用到一些前置知识，由于这比较复杂因此在此不详细叙述，感兴趣的读者可以参考 CF1336E2 的题解。解二实际上是在求一个 $\mathbb{F}_2$ 上线性空间中所有元素的为 1 的位数的分布，我们的结论是：如果能够在正交补空间中解决这个问题，则可以用多项式复杂度的代价得到原空间的答案。因此我们可以在环空间的正交补——割空间上考虑这个问题，而割空间的维数为 $n - c$，可以直接枚举。最终复杂度为 $O(\frac{2^n \times m}{w} + \mathrm{Poly}(m))$。

2.3.2 割集的应用与切边等价

虽然已经介绍了割空间的有关性质，但我们至此还不知道如何判断一个集合是否是割集。

设图中共有 t 条非树边，我们将每条边看作一个 $\mathbb{F}_2$ 中的 t 维向量：第 i 条非树边对应的向量只有第 i 位为 1，而树边对应的向量中，所有跨过这条树边的非树边对应的位为 1，其他位为 0。

现在考虑由生成森林得到的割空间的基，基中的每个元素形如一条树边并上所有跨过这条树边的非树边，根据上面的定义，基中每个元素对应的边的 t 维向量之和为零。

我们知道割空间是线性空间，再结合上面对边对应 t 维向量的定义，我们就不难得到：一个边集是割集，当且仅当其中所有边对应的 t 维向量之和为零。

[5]IOI 2023 中国国家集训队互测

在实现时我们不可能真的用 t 维向量进行操作。我们可以选择将每一维都对应到一个 2^{64} 以内的权值，一个向量就用其中所有为 1 的位对应的权值异或和来表达。也就是说，我们为每条非树边设定一个随机权值，再定义树边的权值为所有跨过它的非树边的权值异或和。如果一个边集边权异或和为零，我们就相信它是割集。

例 2.3.2 (DZY Loves Chinese II[6])**.** 给定无向图 $G = (V, E)$，q 次询问每次给定一个边集，求删除该边集后图是否连通。保证边集大小不超过 15。

通过之前所说的转化，我们需要判定是否存在一个给定边集的非空子集，边权异或和为零。只需要建立一个线性基即可判断。

注意这里边集大小不超过 15 是重要的。因为我们将边权设定在 2^{64} 以内，由于本题需要对所有子集判断是否存在一个异或和为零，所以错误率比判定单个集合是否是割集要高得多，因此只能处理比较小的数据范围。（考虑如果给定边集大小大于 64，则无论如何我们都会判定为不连通，这是荒谬的。）

按照上述方式定义的边权（向量）会将所有边分为若干个等价类，每个等价类中边权相等。我们将这种等价关系称为切边等价。切边等价有一些不同的定义形式：

引理 2.3.3. 在无向图 G 中，e_1, e_2 是边，以下三个命题等价：

1. e_1, e_2 切边等价；

2. e_1, e_2 都是割边；或者 e_1, e_2 在同一边双连通分量，且删除 e_1, e_2 后这个边双连通分量不再连通；

3. 对于任意一个环，或者同时包含 e_1, e_2，或者同时不包含 e_1, e_2。

证明. 根据边权的定义即可知道命题 1 和命题 2 等价。

若命题 2 成立，则假设有一个环包含 e_1 却不包含 e_2，则 e_1, e_2 不是割边，所以删除 e_2 后边双连通分量仍然连通，并且此时还有一个包含 e_1 的环，因此再删除 e_1 也不影响连通性，这说明删除 e_1, e_2 后边双连通分量仍连通，矛盾。故命题 3 成立。

若命题 3 成立，则假设删除 e_1 后 e_2 不是割边，那么说明存在一个环包含 e_2 而不包含 e_1，矛盾。故命题 2 成立。这就证明了命题 2 和命题 3 的等价性。 □

例 2.3.3 (Tours[7])**.** 给定无向有环图 G，你需要用 k 种颜色对每条边进行染色，使得每个简单环上每种颜色的边出现次数相等，求 k 的最大值。

[6]BZOJ 3569

[7]ICPC 2015 World Final

直观地想，由于要求所有简单环上颜色都均匀分布，那么对于除割边外的每个切边等价的等价类，颜色都应该是均匀分布的。显然这导出一个可行的染色方案，即取 k 为所有除割边外等价类大小的最大公因数，然后将每个等价类均匀染色。根据引理 2.3.3 的命题 3，我们知道每个环上的颜色确实是均匀分布的。

下面我们要证明这样的 k 确实是最大的，即证明每个等价类的颜色都要均匀分布。建立一棵 DFS 树，设非树边共有 t 条，仅包含第 i 条非树边的简单环为 C_i，我们只需证明每个 $D_1 \cap D_2 \cap \ldots \cap D_t$ ——其中每个 D_i 为 C_i 或 C_i 的补，这个交集如果非空就是一个切边等价类——都可以表示成 $\{\bigoplus_{i\in S} C_i\}$（这是环空间的所有元素）的实系数线性组合，即可证明所需结论（有这个结论之后可以对每种颜色分别证明个数是对的）。

首先我们证明若干个 C_i 的交，即 $\bigcap_{i\in S} C_i$ 可以表示为 $\{\bigoplus_{i\in S} C_i\}$ 的线性组合。事实上我们有

$$\bigcap_{i\in S} C_i = \frac{1}{2^{|S|-1}} \times \sum_{\varnothing \subset T \subseteq S} (-1)^{|T|} \bigoplus_{i\in T} C_i,$$

这个结论可以由 FWT 的性质导出。由于本文没有专门介绍 FWT，而且这个结论和我们的主题没有关系，所以这里就不证明了。

接下来，通过一个容斥原理即可从若干个 C_i 的交过渡到 D_i 的交，即如果有一个 D_i 为 C_i 的补，我们就将它写成全集减去 C_i，这样就可以转化为 2^s 个子集的 C_i 的交的线性组合，s 表示 D_i 为 C_i 的补的 i 的数量。于是命题得证。

因此，除割边外所有切边等价类大小的最大公因数即为答案。利用前述随机方法，时间复杂度可以轻松地做到 $O((n+m)\log n)$。

3 有向图

有向图中不一定有双向的连通关系。在有向图中若存在路径 $x \rightsquigarrow y$，则我们称 x 可达 y。

将一个有向图中的所有有向边变为无向边，得到的无向图称为原来有向图的基图。若基图连通，则称原来的有向图弱连通。

对于有向无环图，定义它的拓扑排序：拓扑排序是一个点的排列，满足若 x 可达 y，则 x 在拓扑排序中须排在 y 之前。

3.1 可达性问题

可达性问题根据要求的是某一对/一些给定点对之间的可达性关系，或是要求所有点对间的可达性关系，可分为两类，后者称为传递闭包问题。

3.1.1 静态传递闭包

在离散数学中，二元关系 R 的闭包是指使得 $R \subseteq R'$ 且满足某种性质的最小的 R'。那么传递闭包即满足传递性的最小的 R'。显然，对于有向图 G，我们首先将其连边关系 R 补全为自反的（即补充自环），然后其传递闭包就是可达性关系 R'。

Floyd 算法是最常用的传递闭包算法。设关系 R'_k 表示只能途径编号为 $1, 2, \ldots, k$ 的点的情况下的可达性关系，通过枚举途径 k 的路径即可得到 R'_{k-1} 到 R'_k 的递推关系。由于 Floyd 算法在 OI 中是熟知的，这里就不进行具体介绍了。Floyd 算法的时间复杂度为 $O(n^3)$。

在稀疏图中 Floyd 算法很低效。事实上，我们直接枚举终点 y，然后在反图上从 y 开始搜索，就可以得到所有可达 y 的点 x。对每个 y 都这样处理即可得到传递闭包，复杂度 $O(nm)$。

上述算法可以使用 bitset 优化，但在此之前我们需要先将图变为无环的。利用即将在 3.2 节中介绍的内容，我们可以将每个强连通分量缩成一个点，这样原图就变为一个有向无环图 G'。

设 $f(x, y)$ 表示 x 是否可达 y，我们将 $f(x, *)$ 看成一个 01 向量并用 bitset 维护。我们按照 G' 的一个拓扑排序逆序枚举每个点 u，它的 $f(u, *)$ 是由它所有后继点 v 的 $f(v, *)$ 按位或得到的，当然还要加上它到自身的可达性 $f(u, u) = 1$。bitset 进行长度为 n 的 01 向量的位运算复杂度为 $O(\frac{n}{w})$，因此总复杂度 $O(\frac{nm}{w})$。

3.1.2 动态可达性问题举例

一般来说，动态传递闭包是一个比较困难的问题，在本文的参考文献部分列出了相关研究。而更常见的一类问题是给定的若干点对间的动态可达性问题。这类问题中我们一般要沿用求传递闭包的 bitset 方法，同时结合询问分块（或操作分块）等数据结构技巧来解决。

例 3.1.1 (Range Reachability Query[8])**.** 给定有向无环图 $G = (V, E)$，边有编号。现有 q 次询问，每次给定 $u, v \in V$ 和 l, r，求在只保留编号在 $[l, r]$ 中的边的情况下，u 是否可达 v。

由于只有 q 组询问，我们可以仿照之前的 bitset 方法，但是将使用 bitset 存储的第二维由点转化为询问。具体地，设 $f(x, i)$ 表示：在只保留编号为 $[l_i, r_i]$ 中的边的情况下，x 是否可达 v_i，其中 l_i, r_i, v_i 表示第 i 次询问的对应参数。那么第 i 次询问的答案即 $f(u_i, i)$。

考虑一条编号为 c 的边 $x \to y$，这条边对应的转移为：对于所有满足 $l_i \le c \le r_i$ 的 i 有 $f(x, i) \leftarrow f(x, i) \vee f(y, i)$。用 bitset 的语言表达，若记 S_c 表示所有满足 $l_i \le c \le r_i$ 的 i 组成的集合，用一个长度为 q 的 01 向量表示，那么 $f(x) \leftarrow f(x) \vee (f(y) \wedge S_c)$。

用扫描线可以求出所有的 S_c，然后再按照拓扑序逆序转移即可。来考虑一下复杂度，这个做法时间复杂度为 $O(\frac{q(m+n)}{w})$，但空间复杂度也为 $O(\frac{q(m+n)}{w})$，略大。

[8] 2022 年杭电多校训练

这类题目有一种经典的优化空间复杂度的方法，即对使用 bitset 存储的那一维进行分块。在本题中，我们对询问分块，每 B 个询问一块，对于每一块单独运行上面的算法。这样空间复杂度就被优化为了 $O(\frac{B(m+n)}{w})$，而时间复杂度是对每一块进行求和，总和依然为 $O(\frac{q(m+n)}{w})$。当然块长 B 不能太小，否则时间复杂度中的 $\frac{1}{w}$ 将失效。

例 3.1.2 (Dynamic Reachability[9])**.** 给定有向图 $G = (V, E)$，每条边是黑色或白色，初始时每条边都是黑色。接下来进行 q 次操作，每次反转一条边的颜色，或者给定 $u, v \in V$ 询问 u 仅通过黑色边是否可达 v。

由于边的颜色在变化，我们此前的静态传递闭包很难派上用处。因此我们考虑询问分块，这样在同一块内有大量边的颜色都是不变的，有利于我们沿用之前的 bitset 方法。

设块长为 B，这一块内所有修改和询问涉及到的总点数至多为 $2B$，我们称这些点为关键点，记它们依次为 $u_1, \ldots, u_k$。首先我们需要求出那些在块内永不变色的黑色边产生的影响，记 $f(i, j)$ 表示仅考虑块内永不变色的黑色边，i 是否可达 j。注意到我们其实只需要 i, j 都是关键点时的信息，所以使用 bitset 维护的第二维大小仅为 $k = O(B)$（因为第二维只需要记 $u_1, \ldots, u_k$），这部分预处理的复杂度也就是 $O(\frac{B(m+n)}{w})$。

现在我们得到了一张仅包含 k 个关键点的有向图 G'，其中 u_i 到 u_j 有边当且仅当 $f(u_i, u_j) = 1$，这样问题的规模就缩小了，所有非关键的点和边产生的影响全部体现在了 G' 中。

现在考虑如何回答询问，设要查询 u_i 到 u_j 的可达性，在 G' 的边的基础上还需要考虑由块内修改产生的新的黑边，这样的黑边有 $O(B)$ 条。我们可以直接采用暴力搜索的方法，找到 u_i 能到达的关键点集合。以 BFS 为例，每次从队列取出一个元素时，我们需要将由这个点能够新访问到的点加入队列中，这个过程显然是可以用 bitset 优化的：只需要维护当前在队列中或已经出队过的所有点的集合，就可以快速找到当前应该入队的点是哪些了，时间复杂度为 $O(\frac{B^2}{w})$。

总的时间复杂度为 $O(\frac{q(n+m)}{w} + \frac{qB^2}{w})$。和上题一样，块长不能太小，否则 bitset 的 $\frac{1}{w}$ 将失效。因此我们可以就取 $B = \frac{w}{2} = 32$，这样只需要一个 64 位整型就可以描述一个至多 $2B$ 维的 01 向量。不过在本题中将 B 取得稍大一些似乎能有更快的运行速度。

例 3.1.3 (有向无环图[10])**.** 给定有向无环图 $G = (V, E)$，点带点权，初始所有点权为零。接下来有 q 次操作，操作包含如下三种：

1. 给定 u, x，将 u 可达的所有点的点权赋值为 x；

2. 给定 u, x，将 u 可达的所有点的点权对 x 取 min；

3. 给定 u，询问 u 的点权。

本题其实不是动态可达性问题，不过处理方法有一定相似性。

[9] ZJCPC 2022, https://codeforces.com/gym/103687

[10] 本题出处不可考。

首先，我们求 G 的传递闭包，复杂度 $O(\frac{nm}{w})$。

查询 u 的点权可以分为两个子问题：先找到上一次对 u 的赋值操作，再求这次操作到当前时间之间所有对 u 的取 min 操作的最小值。

先看第一个问题。考虑以 B 为块长询问分块，每当处理完一块操作后 $O(n+m)$ 更新一下每个点的上一次赋值操作。在查询时，我们已经知道当前块的操作开始之前的答案，只需再看一下当前块中有没有涉及 u 的赋值操作即可，而这可以直接枚举当前块的所有赋值操作，检查是否有某个操作对应的 u_i 可达 u。

第二个问题处理方法基本相同，我们对于每个 u 和每一个块，都维护这个块中所有涉及 u 的取 min 操作的最小值。查询的形式是给定 u 做区间查询，同样对于整块采用预处理结果，对于零散的部分暴力枚举每个操作即可。

时间复杂度为 $O(\frac{nm}{w}+qB+\frac{q(n+m)}{B})$，取 $B=O(\sqrt{n+m})$，则复杂度为 $O(\frac{nm}{w}+q\sqrt{n+m})$。注意这里我们面临和例 3.1.1 类似的空间问题，可以采用类似的优化：对于每个操作块，我们在用 bitset 维护的那一维只保留块内修改或询问用到的点即可。

3.2 强连通性

定义 3.1 (强连通)**.** 在有向图 G 中，若两个结点 u,v 满足 u 可达 v 且 v 可达 u，则称 u 和 v 是强连通的。若 G 中任意两点都是强连通的，则称 G 是强连通的。

由于可达关系是传递的，因此强连通关系也是传递的，从而强连通关系是一种等价关系。在有向图 G 中我们可以将所有点按照强连通关系划分为等价类。

定义 3.2 (强连通分量)**.** 有向图 G 中的结点按照强连通关系可以划分为若干个等价类，每个等价类的导出子图称为 G 的一个强连通分量。

3.2.1 DFS 树和 Tarjan 算法

在有向图 G 中，若点 r 可达所有点，那么可以从点 r 开始对 G 进行 DFS。此时，与无向图类似地，我们可以定义 G 的 DFS 树。

定义 3.3 (DFS 序，DFS 树)**.** 在有向 $G=(V,E)$ 上以一个可达所有点的结点 $r\in V$ 为起点进行深度优先搜索，若结点 x 是第 p 个被访问到的结点，则称 p 为 x 的 *DFS* 序，记为 $dfn_x=p$。若 $x\neq r$，则在 x 与第一次访问到 x 的结点 y 间连一条边，形成的图 T 称为以 r 为根的一棵 *DFS* 树。

DFS 树是一棵外向树。在有向图 DFS 树上有一些和无向图类似的性质。

性质 3.2.1. 所有结点的 *DFS* 序构成 $1,\ldots,n$ 的一个排列。*DFS* 树上以任何结点 x 为根的子树中所有点的 *DFS* 序构成一段以 dfn_x 开始的连续区间。

性质 3.2.2. 非树边或者是从后代连向祖先的边（返祖边），或者是从祖先连向后代的边（前向边），或者是端点没有祖先关系，但是从 *DFS* 序大的点连向 *DFS* 序小的点的边（横叉边）。

证明是简单的，不过性质 3.2.2 说明有向图中非树边的结构比无向图更复杂一些。

下面考虑如何用 DFS 树求强连通分量，其过程和求双连通分量类似：我们依然在 DFS 过程中记录 low_x 表示从 x 子树中任意一个点出发，只能通过至多一条非树边到达的结点的最小 DFS 序，当找到 $low_x = dfn_x$ 的点 x 时就能得到一个以 x 为最浅点的强连通分量。

但是上述做法不完全正确，这是因为有横叉边的存在。如果 x 子树内有一个横叉边指向某个之前已经统计过的强连通分量，那么实际上这条横叉边是无效的（因为并不能帮助 x 走到 DFS 树上更浅的位置），我们需要忽略这样的横叉边，为此我们对算法做如下改进：

在 DFS 过程中维护一个栈，表示目前访问到的点中，尚未被划分到任何一个强连通分量的点集。依然在 DFS 过程中计算 low_x，但这里我们要求只有当非树边的终点还在栈中（也就是还没有被划入任何强连通分量）时才统计入 low_x 里。当遇到 $low_x = dfn_x$ 的点时，就将当前栈中处于 x 子树里的部分（这一定是栈顶的一段）弹出，和 x 共同构成一个强连通分量。

注意，我们任意选择一个起点，不一定能保证 DFS 到每个点。但是这并不影响，如果有没访问到的点，就任选一个没访问过的点再进行 DFS，重复这个过程直到所有点都被访问即可。

这就是求强连通分量的 Tarjan 算法，时间复杂度为 $O(n+m)$。我们并没有详细证明 Tarjan 算法的正确性，但在理解了无向图的 Tarjan 算法和有向图 DFS 树性质之后，这一切其实是自然的。

求出所有强连通分量后，将每个强连通分量看作一个点，原图中的边如果在一个强连通分量内部则忽略，否则看作连接两个强连通分量的边。这样新得到的有向图是一张有向无环图，这一过程称为缩点。

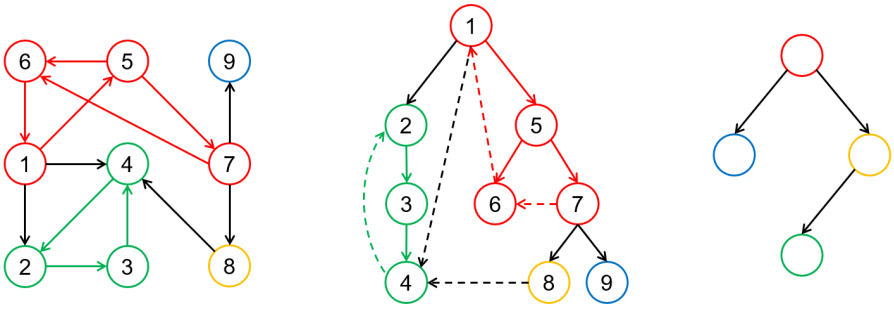


图 4: 一张有向图和它的不同强连通分量、缩点结果

如果 G 本身就是强连通的，那么它的强连通分量只有自身。根据定义，任选一个点出发进行 DFS 都能够访问到所有点。并且根据 Tarjan 算法可知，在 DFS 树上，任何一个点 x 的子树内必定存在一条边指向 DFS 序比 x 小的某个点，无论是返祖边还是横叉边。

例 3.2.1 (Indiana Jones and the Uniform Cave[11])**.** 交互题。有一张强连通的有向图 G，每个结点的出度都等于 k。每个结点处有一个石头指向一条出边，同时有一个属于 $\{0,1,2\}$ 的标记，初始时每个结点的石头指向随机的一条出边，标记为 0。

你不知道 G 的结构，但你需要从某个点出发遍历 G 的每条边。每当经过一条边到达一个点后，你可以得知这个点的标记，然后选择修改标记和石头指向的出边，最后选择一条出边离开，但修改后的标记只能是 1 或 2。一个点的所有出边可以看作是均匀地分布在一个圆周上，因此你在描述一条出边时只能通过“沿着之前石头指向的出边顺时针数的第 i 条”这样的形式。

你只被允许通过 $O(nm)$ 条边，注意你在任何时刻都是已知此前自己的所有决策的。

我们尝试模拟 DFS 的过程，设对 x 的子树搜索的过程为 $dfs(x)$，我们期望当 $dfs(x)$ 允许结束后，x 的每条出边都被访问过了。

三种标记中 0 自然地表示还没有访问过的点，而我们规定 1 和 2 分别表示对应的点是/不是 x 的祖先，x 就是当前正在执行 dfs 过程的点。

对于过程 $dfs(x)$，我们顺序枚举 x 的每条出边 $x \to y$，考虑 y 的标记：

- y 的标记为 0。这对应了 y 是 DFS 树上 x 的儿子，我们递归进行 $dfs(y)$，然后回溯即可。

- y 的标记为 1。这说明 $x \to y$ 是返祖边。在走完这条边后我们希望走回 x，为了不迷路，我们规定标记为 1 的点上的石头指向的出边就是它在 DFS 树上到 x 的路径上的边。这样我们只要沿着石头指向的出边走就能够回到 x，但一个问题是我们如何知道自己回到了 x？答案是将 x 的标记临时修改为 2，这样下次访问到标记为 2 的点时就必然是回到 x 了。

- y 的标记为 2。这说明 $x \to y$ 是后向边或横叉边。无论如何，我们还是希望走回 x，但这时不像上一种情况那么方便了，我们需要先逆着 DFS 树的方向往回走，直到走到一个标记为 1 的点，然后再按照上面的方式回到 x（这时判断回到 x 的方式有所不同，但同样比较简单，不再赘述）。那么问题在于如何走到一个标记为 1 的点。回忆 Tarjan 算法中定义的 low 数组，如果我们不断从当前点 u 走到 low_u，那么——根据图的强连通性——必然可以走回根结点，也就一定能经过标记为 1 的点了。因此我们对于每个标记为 2 的点 u，定义它的石头指向的出边为 low_u 的方向。也就是说如果

[11]NEERC 2016, https://codeforces.com/gym/101190

low_u 对应的那条边就是 $u \to low_u$，那么石头就指向这条边；否则 low_u 对应的那条边是 $v \to low_u$，v 是 u 的后代，那么石头就指向 v 所在子树的方向。

最后一个问题：在执行完 $dfs(x)$ 后我们需要回溯至 x 的父结点，但这其实和上面所说的最后一种情况是一样的，只需先走到 low_x 再向下走一段距离即可。

在实现时有许多细节，但算法的大致框架已经明确了。我们需要 $O(nm)$ 步来遍历每条边。

3.2.2 耳分解

首先回忆无向图部分定义的耳分解，并将它扩展到有向图上。

定义 3.4 (耳和耳分解)**.** 在有向图 $G = (V, E)$ 中有子图 $G' = (V', E')$，若简单路径或简单环 $P : x_1 \to x_2 \to \ldots \to x_k$ 满足 $x_1, x_k \in V'$，$x_2, \ldots, x_{k-1} \notin V'$，则称 P 是 G 关于 G' 的耳。

G 的一个子图序列 $(G_0, G_1, \ldots, G_k)$ 若满足：

1. G_0 是一个简单环（可以只有一个点），$G_k = G$；

2. G_{i-1} 是 G_i 的子图；

3. 设 $G_i = (V_i, E_i)$，则 $E_i \backslash E_{i-1}$ 构成 G_{i-1} 的一个耳。

则称 $(G_0, G_1, \ldots, G_k)$ 是 G 的一个耳分解。

与无向图类似地，我们有如下结论：

定理 3.1. 有向图 G 可耳分解，当且仅当 G 强连通。

证明. 先证必要性。若 G 有耳分解 $(G_0, \ldots, G_k)$，则简单环 G_0 是强连通的。若 G_i 是强连通的，则设 G_{i+1} 中加入的耳为 $x \rightsquigarrow y$，由 G_i 的强连通性知 y 可达 x，从而存在一个包含耳的环，因此耳上所有点与 x, y 强连通，那么 G_{i+1} 也是强连通的。用数学归纳法知 $G = G_k$ 强连通。

再证充分性。若 $G = (V, E)$ 强连通，我们下面给出一个耳分解的构造。先求出 G 的以 1 为根的一棵 DFS 树，并按照 Tarjan 算法算出 low 数组。G 的强连通性告诉我们当 $x \neq 1$ 时都有 $low_x < dfn_x$。接下来，我们进行如下过程构造 $(G_0, \ldots, G_k)$：

G_0 为仅包含 1 的图。然后按照 DFS 序从小到大依次枚举每个点，设当前点 x 的 DFS 序为 p，此时 DFS 序为 $1, \ldots, p-1$ 的点都已经在当前的 G_i 中了。如果 x 已经在当前的 G_i 中则直接跳过，否则，根据 $low_x < dfn_x$，我们知道存在 x 的后代 y 使得存在边 $y \to z$，z 是一个已经在 G_i 中的点，于是我们将 $f_x \to x \rightsquigarrow y \to z$ 这条路径作为一个耳加入 G_i（其中 f_x 表示 x 的父结点），成为 G_{i+1}。枚举完所有点后，所有点都在当前的 G_i 中了，此时再把尚未加入的边一条一条单独作为耳加入即可。 □

例 3.2.2 (Economic One-way Roads[12])**.** 给定一张无向图 G，你要为每条边指定一个方向，每条边的每种方向各有一个代价，你需要使得定向后的有向图强连通，且每条边对应方向的代价之和最小。

作为对照，本例其实是例 2.2.4 的强连通版本。

设 $f(S)$ 表示 S 的导出子图对应的答案，每次在 S 的基础上加一个耳进行转移。直接枚举耳的点集复杂度为 $O(3^n \times \mathrm{Poly}(n))$，而如果在转移时就按照耳上的顺序一个一个将点加入耳即可做到 $O(2^n \times \mathrm{Poly}(n))$。因为具体过程和例 2.2.4 基本一样，所以不再赘述细节。

定理 3.2. 无向图 G 可以被定向为强连通图，当且仅当 G 边双连通。

证明. 必要性：如果 G 不是边双连通的，则 G 不连通或有一条割边。G 不连通显然不能被定向为强连通图，而如果 G 存在割边 (x, y)，则无论这条割边如何定向，都有 x 不可达 y 或 y 不可达 x，因此不可能存在强连通定向。

充分性：如果 G 边双连通，则它存在耳分解，我们将 G_0 定向为有向环，再将之后的每个耳都定向为有向路径或有向环，就得到了有向的耳分解，根据定理 3.1，这种定向对应的有向图就是强连通的。 □

事实上定向方式的构造并不需要把耳分解求出来那么麻烦，我们只需对边双连通图求出 DFS 树，将树边都从父到子定向，非树边都从后代到祖先定向即可，这和上面说的通过耳分解构造的定向是基本相同的。

例 3.2.3 (App 管理器[13])**.** 混合图 G（即可能同时包含有向边和无向边的图）可以被（对无向边）定向为强连通图，当且仅当以下两个条件满足：

1. 将有向边都看作无向边时，图边双连通；

2. 将无向边拆成一对反向的有向边时，图强连通。

证明. 必要性是显然的，下面证明充分性。

若条件 1,2 都满足，如果没有无向边，那么就不用定向了；否则，我们任意取出一条无向边 (u, v)，证明它存在一种定向使得定向后条件 1,2 仍然满足（实际上只要考虑条件 2 即可，因为条件 1 和方向无关），这样就可以归纳地证明结论了。

记当前的图为 G_0，在 G_0 中删去无向边 (u, v) 形成的图为 G_0'。对于任意结点 x，由于 G_0 强连通，所以存在一条简单路径 $x \rightsquigarrow u$，若这条路径不经过无向边 (u, v)，则说明 G_0' 中 x 可达 u；否则说明 G_0' 中 x 可达 v。同理，G_0' 中要么 u 可达 x，要么 v 可达 x。

在 G_0' 中，若存在 x 使得 u 可达 x，x 可达 v，则说明 u 可达 v。同理若存在 x 使得 v 可达 x，x 可达 u，则说明 v 可达 u。否则，对于每个 x 都是 u, x 相互可达或 v, x 相互可达。因

[12]XXI Open Cup, Grand Prix of Korea, https://codeforces.ml/gym/102759

[13]UOJ Round 9, https://uoj.ac/contest/18

此整个图有至多两个强连通分量。假设 u,v 相互不可达，则这两个强连通分量之间没有边，这说明 G_0' 的基图不连通，即 G_0 中 (u,v) 是割边，与条件 1 矛盾。因此 u 可达 v 或 v 可达 u 至少有一个成立。

若 u 可达 v，则将 (u,v) 定向为 $v \to u$，否则定向为 $u \to v$，这使得 u,v 互相可达，因此整张图还是强连通，这正是我们想证明的。 □

3.2.3 有向环

在无向图中有环空间，而本节中我们将研究有向图的环结构。

定义 3.5 (可环覆盖图)**.** 若有向图 G 的每个结点的入度都等于出度，则称 G 是可环覆盖的。

引理 3.2.1. 可环覆盖的有向图的边集可以表示为若干个简单环的不相交并。

证明 3.2.3 的证明考虑归纳即可。可环覆盖图的概念可以类比无向图中环空间的元素。

和环空间类似地，我们要通过 DFS 树得到可环覆盖子图的一组“基”，不过它并不是真正的线性基，因此并不总是通用。一般来说，我们考虑的是边带权情况下（若没有特别规定的边权，则认为每条边边权为 1），环的边权和在诸如异或空间或 mod$m$ 剩余类之类的结构下的性质。

引理 3.2.2. 强连通图 G 中，若存在一条路径 $P : x \rightsquigarrow y$ 边权和为 S，则存在路径 $y \rightsquigarrow x$ 使得其边权和 S' 满足 $S' \equiv -S \pmod{m}$。

证明. 若 $m = 1$ 则结论显然。否则，任意找一条路径 $Q : y \rightsquigarrow x$，设其边权和为 T，考虑路径 $Q + P + Q + P + \ldots + Q$，其中包含 $m-1$ 个 P 和 m 个 Q，加法表示路径的拼接。则这条路径是 y 到 x 的，且边权和 $(m-1) \times S + m \times T \equiv -S \pmod{m}$。 □

引理 3.2.3. 强连通图 G 中，对于每条边 $x \to y$，边权为 z，建立一条新边 $y \to x$，边权 $-z$。这样会得到一张边权具有反对称性的新图 G'。G 和 G' 的所有可环覆盖子图边权和的集合在 mod$m$ 意义下等价。

证明. 这是引理 3.2.2 的推论，因为 G' 的每个环可以通过引理 3.2.2 对应到 G 的一个环。 □

于是我们只要研究这种边权具有反对称性的图即可。这种图的 DFS 树可以认为不再是单向的了，而是可以在树上双向行走，不过通过简单的计算可知，（不一定简单的）树上的路径 $x \rightsquigarrow y$ 的边权和必然为 $dep_y - dep_x$，其中 dep_x 表示 DFS 树上根结点到 x 的路径边权和。

引理 3.2.4. 在边权具有反对称性的图 G 上，将仅包含一条非树边的简单环（这里树边的反向边不算作非树边）称为基本环。则任何一个 G 的可环覆盖子图的边权和等于一些基本环的边权和的整系数线性组合。

证明. 若某个环依次经过非树边 $e_1,\dots,e_k$，e_i 形如 $x_i \to y_i$，边权为 z_i。则这个环的边权和为 $\sum_{i=1}^{k}(z_i+dep_{y_{i+1}}-dep_{x_i})$，其中 $y_{k+1}=y_1$。将求和项中的 y_i 项进行交换得到 $\sum_{i=1}^{k}(z_i+dep_{y_i}-dep_{x_i})$，现在求和的每一项正好是一个基本环的边权和。再结合可环覆盖子图可以拆成若干个环的性质，引理得证。 □

定理 3.3. 强连通图 G 的所有可环覆盖子图的边权和的最大公因数，等于所有 $z+dep_y-dep_x$ 的最大公因数，其中 (x,y,z) 取遍一切非树边的起点、终点和边权构成的三元组，dep_x 表示树上根结点到 x 的路径长度。

证明. 设所求最大公因数为 g，根据引理 3.2.3 和引理 3.2.4，我们知道 G 的可环覆盖子图边权和的集合，与所有 $z+dep_y-dep_x$ 的集合在 $\bmod g$ 意义下等价，因此如果其中任何一个集合中所有元素都是 g 的倍数，则另一个集合亦然，这说明了用两种方式求得的最大公因数是一致的。 □

上面我们只讨论了 $\bmod m$ 剩余类下的性质，不过只要引理 3.2.2 仍然成立，以上性质就可以推广到其他一些结构中，比如异或空间，或更加广义的 k 进制不进位加法的结构中。不过考虑 $\bmod m$ 的性质时只需要掌握最大公因数就基本可以得到全部的结构信息，但在上述的其他结构中可能额外需要线性基等工具。

例 3.2.4 (有向图的周期)**.** 在有向图 $G=(V,E)$ 中，如果存在一组整数 p,k，使得对图上任意两点 x,y 和任意的 $t\ge k$，存在一条包含 t 条边的路径 $x \rightsquigarrow y$ 当且仅当存在一条包含 $t+p$ 条边的路径 $x \rightsquigarrow y$，且以 p 为第一关键字，k 为第二关键字，上述整数对是最小的，则称 p,k 分别为 G 的一组周期和幂敛指数。

一种等价的说法是，设 G 的邻接矩阵为 A，对于任意 $t\ge k$ 有 $A^t=A^{t+p}$，这里的矩阵乘法定义为乘法是按位与、加法是按位或。

考虑强连通图的周期，这其实就是定理 3.3 所说的所有环长的最大公因数 g，因为当 t 充分大时，可以在强连通图中随意行走，根据裴蜀定理，我们可以用各环的整系数线性组合拼出 g 的任意倍数。当步数足够大时，所有系数都可以是正整数。

而一般图的周期必然要是每个强连通分量的周期的倍数，取所有强连通分量的周期的最小公倍数即可，这比较容易理解，我们略去严格证明。

对于幂敛指数，没有直接的公式可以用来计算。

例 3.2.5 (Phoenix and Odometers[14])**.** 给定一张边带权有向图 G，进行 q 次询问，每次给出 v, b, m，求是否存在一个经过 v 的（不一定简单的）环边权和 S 满足 $S \equiv b \pmod m$。

这是 mod$m$ 剩余类结构的一个例子。显然只需要考虑 v 所在的强连通分量，按照定理 3.3 求出其中所有环长的最大公因数 g，判断是否有 $\gcd(g, m) \mid b$ 即可。

例 3.2.6 (术树数[15])**.** 给定一张带权无向图 G 和整数 k，进行 q 次询问，每次给定 x, y, v，求是否存在一条路径 $x \rightsquigarrow y$，满足所有边权的 k 进制不进位加法运算结果等于 v。

这是 k 进制不进位加法的一个例子。与之前不同，这是一张无向图，所以我们需要将每条无向边拆成一对有向边，因此在进行定理 3.3 类似操作时，除了非树边外还要考虑树边的反向边，即对于边权为 w 的树边，$2w$ 也是基中的元素。然后我们就可以用所有这些基本环的边权和建立一个 k 进制线性基，用来查询了。

本题中还有一个不同就是目标并非是环，而是路径 $x \rightsquigarrow y$，但我们只需任选一条路径 $x \rightsquigarrow y$，比如树上路径，再用它加上基中元素的一个子集，就能得到所有 x 到 y 的路径。因此查询时我们以树上路径 $x \rightsquigarrow y$ 边权和为初值，找线性基中是否存在一些数与这个初值的和为 v 即可。

3.3 竞赛图

竞赛图即对于任意一对不同结点 u, v，在 $u \to v$ 和 $v \to u$ 之中恰有一条边存在的简单有向图。现在，我们简单讨论一下竞赛图的一些与连通性有关的结论。

3.3.1 哈密顿路

定义 3.6 (哈密顿路、哈密顿回路)**.** 图 G 中，经过所有点恰好一次的路径称为哈密顿路，经过所有点恰好一次（起点和终点算作同一次）的环称为哈密顿回路。

定理 3.4. 竞赛图有哈密顿路。

证明. 对点数归纳，若 $n-1$ 个点的竞赛图有哈密顿路，则在 n 个点的竞赛图中，不妨设前 $n-1$ 个点的导出子图的哈密顿路就是 $1 \to 2 \to \ldots \to n-1$，则：

1. 若有边 $n \to 1$，则将 n 连在哈密顿路的开头；

2. 若有边 $n-1 \to n$，则将 n 连在哈密顿路的末尾；

3. 若有边 $1 \to n$, $n \to n-1$，则一定存在一个 $x \in [1, n-2]$ 使得 $x \to n$, $n \to x+1$（可以使用二分法找这样的 x），将 n 插在 x 与 $x+1$ 之间。

于是 n 个点的竞赛图也有哈密顿路，由数学归纳法知结论成立。 □

[14]CF 1515 G

[15]CTT 2020，题意不完全一致

定理 3.5. 强连通的竞赛图有哈密顿回路。

证明. 根据定理 3.4，首先构造一个哈密顿路，不妨设为 $1 \to 2 \to \ldots \to n$。再找到最大的 t 使得存在边 $t \to 1$，那么对于 t 之后的点 x 都有边 $1 \to x$，即 $1 \to t+1, \ldots, 1 \to n$。现在我们有一个环 $1 \to 2 \to \ldots \to t \to 1$，接下来尝试将 $t+1, \ldots, n$ 中的点加入环中。

我们重新描述一下问题：有一个包含 1 的环，并且 1 到所有不在环上的点都有边，我们想要证明，存在一个当前不在环上的点可以被加入环中形成一个更大的环。事实上，由于一定存在一个不在环上的点 x 和一个环上的点 y 使得有边 $x \to y$（否则不在环上的点不可达环上的点，图就不是强连通的了），结合 $1 \to x$，我们就可以找到环上一个合适的位置 z（在 $1, y$ 之间），使得 $z \to x \to next_z$，将 x 插入 z 和 $next_z$ 即可，其中 $next_z$ 表示环上 z 的下一个点。 □

3.3.2 强连通分量

一般图缩点之后会变成有向无环图，对于竞赛图来说，有更好的结果。

引理 3.3.1. 竞赛图缩点后形成一条链。

证明. 对于不同两个强连通分量中的点 x, y，由于一定存在 $x \to y$ 或 $y \to x$，因此 x 可达 y 或 y 可达 x。所以强连通分量间的可达关系形成全序，也就是缩点后形成一条链。 □

在竞赛图中求强连通分量，并不需要用 Tarjan 算法，有许多更简单的方法。

我们可以首先找到竞赛图的一个哈密顿路，不妨设为 $1 \to 2 \to \ldots \to n$，则强连通分量必定是一段连续的区间。对于一条边 $x \to y$，如果 $x < y$ 那么它对连通性无影响（因为链上 x 就能到 y）；而如果 $x > y$，则说明 $y \to y+1 \to \ldots x$ 并上这条边形成一个环，即 $[y, x]$ 中所有点属于同一强连通分量。我们将所有这样的区间 $[y, x]$ 都进行合并，会得到 $[1, n]$ 的一个区间划分，这些区间就是所有的强连通分量。

有一个更简单实用的方式求解竞赛图的强连通分量，那就是借助下面的 Landau 定理。

定理 3.6 (Landau)**.** 设竞赛图 G 中所有点按照出度从小到大排序为 $p_1, \ldots, p_n$，p_i 的出度为 d_i，则对于每个 $1 \le k \le n$，有 $\sum_{i=1}^{k} d_i \ge \binom{k}{2}$。

扩展：G 的每个强连通分量是排列 p 中的一个连续段，且 p_k 是一个强连通分量的右端点当且仅当 $\sum_{i=1}^{k} d_i = \binom{k}{2}$。

证明. 仅考虑前 k 个点的导出子图，共有 $\binom{k}{2}$ 条边，这已经为 $\sum_{i=1}^{k} d_i$ 贡献了 $\binom{k}{2}$，因此 $\sum_{i=1}^{k} d_i \ge \binom{k}{2}$。

若 x, y 属于不同强连通分量，且 x 可达 y。那么对于所有 y 的出边 $y \to z$，易证一定存在边 $x \to z$，在此基础上 x 至少还多一条出边 $x \to y$，因此 x 的出度大于 y 的出度。这说明每个强连通分量确实是排列 p 中连续的一段。

若 $\sum_{i=1}^{k} d_i \geq \binom{k}{2}$ 取到等号，说明 $p_{k+1}, \ldots, p_n$ 到 $p_1, \ldots, p_k$ 没有任何边，则 p_{k+1} 之后的点不可达 p_k 之前的点，这两侧的点不可能属于同一强连通分量。另一方面，若 $p_i, \ldots, p_k$ 构成强连通分量，也能推出 p_{k+1} 之后的点不可达 p_k 之前的点，即等号成立。这就证明了定理扩展。 □

在 Landau 定理中将出度换为入度，相应结论显然也是成立的，我们下面采用入度的描述。

将所有点按入度从小到大排序，按照 $\sum_{i=1}^{k} d_i = \binom{k}{2}$ 的 k 为右端点，划分出的每个连续段就是所有的强连通分量，且靠左的强连通分量可达靠右的强连通分量。

例 3.3.1 (基础图论练习题[16])**.** 给定竞赛图 G，对于每条边求翻转这条边后图的强连通分量个数。

解一：用哈密顿路解决这个问题。

如果翻转的边连接的是两个不同强连通分量，那么显然翻转后会将哈密顿路上位于这两个强连通分量之间的所有强连通分量合并为一，这部分很容易计算。

如果翻转的边在同一个强连通分量内部，那么我们求出这个强连通分量的一个哈密顿回路。如果翻转的边不在哈密顿回路上，显然对强连通性没有影响；否则，在哈密顿回路上删除这条边后，还剩一个哈密顿路，根据上文提到的，对于所有和哈密顿路反向的边 $x \to y$，其效果是将 $[y, x]$ 并成一个强连通分量。现在 $[y, x]$ 对应的是环上区间，用一些数据结构技巧维护所有 $[y, x]$ 后可以 $O(1)$ 查询删除一条环上的边之后的强连通分量情况，但具体的实现相当麻烦。

解二：用 Landau 定理解决这个问题。

将所有点按照入度从小到大排序为 $p_1, \ldots, p_n$，p_i 的入度为 d_i。翻转一条边 $p_x \to p_y$ 会使得 p_x 的入度增加 1，p_y 的入度减少 1。我们仅考虑 d_i，发现上述修改可以表示为 $O(1)$ 个 d_i 的单点修改，而我们要回答的是满足 $\sum_{i=1}^{k} d_i = \binom{k}{2}$ 的 k 的个数，为此我们只需维护每个区间中 $\sum_{i=1}^{k} d_i - \binom{k}{2}$ 的最小值和最小值数量，即可简单地回答。

两个解法时间复杂度都为 $O(n^2)$，但解二更简便一些。

4 总结

本文讨论了许多与图的连通性有关的问题和算法，提供了一些解决这些问题的常用结构和思路。在无向图中我们以 DFS 树为中心，重点讨论了双连通关系与割集的一些性质和应用。在有向图中我们以可达性关系和强连通分量为中心，讨论了对应的一些性质和应用。

事实上有许多本文并未深入提及的内容，比如无向图的最小割、三连通分量、有向图的支配关系、动态传递闭包等。没有提及的原因有二，一是限于篇幅，二是这些内容都曾在

[16]CTT 2020

之前几年的集训队论文中有过讨论了。笔者将这些内容对应的论文全部列在了参考文献中，感兴趣的读者可以作为延申阅读。

5 致谢

- 感谢中国计算机学会提供学习和交流的平台。
- 感谢华东师范大学第二附属中学的金靖老师的关心与指导。
- 感谢与我交流相关内容的同学们的帮助。
- 感谢父母的关心与支持。

参考文献

[1] 虞皓翔，《再谈图连通性相关算法》，IOI 2021 集训队论文集

[2] 常瑞年，《线性代数在 OI 中的应用》，IOI 2022 集训队论文集

[3] Wikipedia, “Ear Decomposition”, https://en.wikipedia.org/wiki/Ear_decomposition

[4] Wikipedia, “Bipolar Orientation”, https://en.wikipedia.org/wiki/Bipolar_orientation

[5] zx2003, 浅谈双极定向及其应用，https://www.luogu.com.cn/blog/user19567/yi-dao-tu-lun-ti-ji-ji-yan-sheng

[6] CCF，2020 全国信息学奥林匹克年鉴

[7] 王文涛，《浅谈无向图最小割问题的一些算法及应用》，IOI 2016 集训队论文集

[8] 孙耀峰，《动态传递闭包问题的探究》，IOI 2017 集训队论文集

[9] 陈孙立，《浅谈支配树及其应用》，IOI 2020 集训队论文集

浅谈信息学竞赛中数据的构造与生成

山东省潍坊第一中学 刘一平

摘 要

本文介绍了序列、树、图、括号串等多种数据的随机生成和构造方法，以及常见的哈希错误和对应数据的构造方法.

前言

信息学竞赛中，常常需要在本地测试程序的正确性.

一种常用的方法是自己构造测试数据. 使用规模较小的数据，配合时间复杂度较高但正确性有保证的另一份程序，可以测试自己程序的正确性；使用规模较大的数据，可以测试自己程序的效率. 选手通常称这种方法为“对拍”.

在对拍中，如果生成的数据强度不够，就有可能难以查出自己程序的错误，或让时间复杂度高的算法通过规模较大的数据.

本文将介绍几种常见数据的生成方式，如何生成更为随机，或更具针对性的数据.

1 序列

1.1 排列

随机排列的方式通常有两种，它们看起来比较相似，但原理并不相同.

以下是第一种算法.

算法 1 随机排列

输入： n

输出： 一个 1 到 n 的随机排列

令 p 为一个长度为 n 的序列，初始 $p_i = i$
for $i = 1$ **to** n **do**
  $j \leftarrow \text{random}(1, i)$
  $\text{swap}(p_i, p_j)$
end for
return p

这里 random(l, r) 的作用是等概率随机一个 $[l, r]$ 内的整数.

引理 1.1.1. 算法 *1* 可以等概率随机生成一个 1 到 n 的排列.

证明. 注意到这个打乱算法就是先对 p 的前 $n-1$ 项做了规模为 $n-1$ 的打乱，再令 $j = \text{random}(1, n)$，并交换 p_n 与 p_j.

考虑对 n 归纳.

命题对于 $n = 1$ 显然成立，考虑由 $n-1$ 成立推出 n 成立.

交换后有 $p_j = n$. 因为 j 是从 1 到 n 等概率随机的，所以只需要证明 $p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n$ 是一个等概率随机的排列即可.

而交换前的 $p_1, \ldots, p_{n-1}$ 显然与交换后的 $p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n$ 构成双射，因此交换得到的是一个等概率随机的排列.

□

以下是第二种算法.

算法 2 随机排列

输入： n

输出： 一个 1 到 n 的随机排列

令 p 为一个长度为 n 的序列，初始 $p_i = i$
for $i = 1$ **to** n **do**
  $j \leftarrow \text{random}(i, n)$
  $\text{swap}(p_i, p_j)$
end for
return p

引理 1.1.2. 算法 *2* 可以等概率随机生成一个 1 到 n 的排列.

证明. 这个算法可以看成先将 p_1 等概率交换为 1 到 n 的值之一，然后对 p_2 到 p_n 做规模为 $n-1$ 的算法.

考虑对 n 归纳，正确性显然.

□

1.2 无重复元素的序列

考虑如下问题：你需要等概率随机一个长度为 n 的序列，值域为 1 到 m（$m \geq n$）的整数，且序列中的元素不重复.

一种方法如下：

算法 3 随机无重复元素的序列

输入： n

输出： 一个 1 到 n 的随机排列

```
令 a 为一个长度为 n 的序列
for i = 1 to n do
    a_i ← random(1, m)
    while a_i ∈ {a_1, a_2, ..., a_{i-1}} do
        a_i ← random(1, m)
    end while
end for
return p
```

引理 1.2.1. 算法 *3* 可以等概率随机生成一个长度为 n，值域为 $[1,m] \cap \mathbb{Z}$，且元素不重复的序列.

证明. 显然，其生成的序列必然合法，且它可以生成所有合法序列，且所有合法序列的生成概率相等，都为 $\frac{1}{m^n}$.

□

这种方法的弊端是，在 m 较小时，生成会比较慢.

例如，在 $m = n$ 时，调用 random 的期望次数可以计算为

$$\sum_{i=1}^{n} n/i = \Theta(n \log n)$$

判断 $a_i \in \{a_1, a_2, \ldots, a_{i-1}\}$ 可以使用哈希表做到 $\Theta(1)$，这样的总复杂度为 $\Theta(n \log n)$.

但注意到在 m 较小时可以采用另一种算法：生成一个 1 到 m 的随机排列，然后取前 n 项.

在 $m \le 2n$ 时采用第二种算法，这部分的复杂度就为 $\Theta(n)$.

在 $m > 2n$ 时采用第一种算法，对于每个 i，每次随机到之前没出现过的值的概率为 $(m-i+1)/m \ge 1/2$，因此期望复杂度为 $\Theta(n)$.

另一种方法是使用算法 2，我们只需要将 i 循环到 n，使用哈希表储存被操作过的位置以及它们的值，复杂度可以做到 $\Theta(n)$.

1.3 和固定的序列

考虑如下问题：你需要等概率随机一个长度为 n 的正整数序列 $a_1, a_2, \ldots, a_n$，要求 $\sum_{i=1}^{n} a_i = m$.

设 $s_i = \sum_{j=1}^{i} a_j$，考虑转而生成 s.

不难发现，所有满足 $1 \le s_1 < s_2 < \cdots < s_n = m$ 的 s 与 a 一一对应. 因此等概率随机一个 a 等价于等概率随机一个这样的 s.

而生成这样的 s 就等价于：在 $[1, m-1]$ 中等概率随机选择 $n-1$ 个不同的正整数. 这就转化为了生成无重复元素序列的问题.

对于 a_i 为非负整数的情况，可以将所有 a_i 都加上 1，从而转化为正整数的情况.

更进一步，如果 a_i 的下界为 b_i，可以将 a_i 减去 $(b_i - 1)$，从而转化为下界为 1 的情况.

2 图

2.1 树

2.1.1 方法 1：随机父亲

一种信息学竞赛中常见的随机生成树的方式如下：对每个 $2 \le i \le n$，在 1 到 $i-1$ 中等概率随机选择一个点作为 i 的父亲.

对于一棵有根树，定义一个点的**深度**为它到根节点的最短路径上的边数，定义这棵树的**高度**为所有点的深度最大值.

引理 2.1.1. 对每个 $2 \le i \le n$，在 1 到 $i-1$ 中等概率随机选择一个点作为 i 的父亲，这样生成的树高的期望是 $\Theta(\log n)$.

引理 2.1.2 (琴生不等式)**.** 对于函数 f 在区间 $[L, R]$ 内下凸，对于任意实数 $x_1, x_2, \ldots, x_n \in [L, R]$ 以及 $a_1, a_2, \ldots, a_n$ 满足 $\sum_{i=1}^{n} a_i = 1$ 且 $a_i > 0$，有

$$f\left(\sum_{i=1}^{n} a_i x_i\right) \le \sum_{i=1}^{n} a_i f(x_i)$$

引理 2.1.2 的证明略，具体可参考 [1].

引理 *2.1.1* 的证明. [1]

设树高为 h，根据琴生不等式，有 $2^{\mathrm{E}[h]} \le \mathrm{E}[2^h]$. 考虑计算 $\mathrm{E}[2^h]$ 的上界.

设点 i 的深度为 d_i，那么

$$\begin{aligned}\mathrm{E}[2^h] &= \mathrm{E}\left[\max_{i=1}^{n}(2^{d_i})\right] \\ &\le \mathrm{E}\left[\sum_{i=1}^{n} 2^{d_i}\right] \\ &= \sum_{i=1}^{n} \mathrm{E}[2^{d_i}]\end{aligned}$$

设 $f_i = \mathrm{E}[2^{d_i}]$，那么 $f_1 = 1$，对于所有 $2 \le i \le n$ 有

$$f_i = \frac{2}{i-1}\sum_{j=1}^{i-1} f_j$$

设 $F_i = \sum_{j=1}^{i} f_j$，那么对于所有 $2 \le i \le n$ 有

$$F_i - F_{i-1} = \frac{2}{i-1}F_{i-1}$$

那么

$$\begin{aligned}F_i &= \frac{i+1}{i-1}F_{i-1} \\ &= \prod_{j=2}^{i}\frac{j+1}{j-1} \\ &= \frac{(i+1)i}{2}\end{aligned}$$

所以

$$2^{\mathrm{E}[h]} \le \mathrm{E}[2^h] \le F_n = \frac{(n+1)n}{2}$$

两边取对数得 $\mathrm{E}[h] = O(\log n)$.

从另一个方向，

$$\mathrm{E}[h] \ge \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n} d_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathrm{E}[d_i]$$

[1] 部分参考自 [2]

设 $g_i = \mathrm{E}[d_i]$，那么 $g_1 = 0$，对于 $2 \le i \le n$ 有

$$g_i = 1 + \frac{1}{i-1}\sum_{j=1}^{i-1} g_j$$

设 $G_i = \sum_{j=1}^{i} g_j$，对于 $2 \le i \le n$ 有

$$G_i - G_{i-1} = 1 + \frac{1}{i-1} G_{i-1}$$

因此

$$\begin{aligned} G_i &= 1 + \frac{i}{i-1} G_{i-1} \\ &= \sum_{j=2}^{i} \prod_{k=j}^{i-1} \frac{k+1}{k} \\ &= \sum_{j=2}^{i} \frac{i}{j} \\ &= \Theta(i \log i) \end{aligned}$$

所以

$$\mathrm{E}[h] \ge \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}[d_i] = \frac{1}{n} G_n = \Theta(\log n)$$

所以 $\mathrm{E}[h] = \Omega(\log n)$.

综上，$\mathrm{E}[h] = \Theta(\log n)$.

□

树高的期望为 $\Theta(\log n)$，很多时候不能很好地测试算法效率.

例如一些算法的复杂度为 $\Theta(\sum_i \mathrm{siz}_i) = \Theta(\sum_i \mathrm{dep}_i)$，其中 siz_i 和 dep_i 分别表示点 i 的子树大小和深度，在这种数据下的期望复杂度就为 $\Theta(n \log n)$，而实际上的最高复杂度为 $\Theta(n^2)$.

如果启发式合并时没有选择最大的儿子作为重儿子，或者在一些对链修改的情境下出现错误，都有可能导致算法复杂度达到 $\Theta(\sum_i \mathrm{dep}_i)$，使用这种数据往往无法发现错误.

2.1.2 方法 2：在一定范围内随机父亲

修改方法 1 的常见方法是限定随机父亲的范围.

我们可以设定一个阈值 B，点 i 的父亲在 $[\max(1, i-B), i-1]$ 内随机.

使用这种方法通常需要适时调整 B 的大小，以生成不同类型的数据.

一般会将 B 设置得较小，例如对于 $n = 2 \times 10^5$ 的数据可以设定 $B = 10$ 或 20，这样生成的树比较适合测试启发式合并.

2.1.3 方法 3：Prufer 序列

对于一棵有标号无根树，它的 Prufer 序列由如下方法生成：

算法 4 Prufer 序列

输入： 一棵 n 个点的有标号无根树 T

输出： T 的 Prufer 序列

令 a 为一个长度为 $n-2$ 的序列.
for $i = 1$ **to** $n-2$ **do**
  $u \leftarrow$ 标号最小的叶子（度数为 1 的点）
  $a_i \leftarrow$ 与 u 相邻的点的标号
  删除 u 以及它的临边
end for
return a

可以看到，n（$n \geq 2$）个点的有标号无根树的 Prufer 序列是一个长度为 $n-2$，值域为 $[1,n] \cap \mathbb{Z}$ 的序列.

一棵树唯一对应了一个 Prufer 序列，那么一个合法序列是否对应了一棵树呢？

引理 2.1.3. *Prufer* 序列构建了一个从 n 个点的树到长度为 $n-2$、值域为 $[1,n] \cap \mathbb{Z}$ 的序列的双射.

证明. 只需要证明，每个 Prufer 序列都存在唯一的树 T 与之对应.

考虑对 n 归纳. $n = 2$ 的证明是简单的，考虑用 $n-1$ 推出 n.

设给定的 Prufer 序列为 $a_1, a_2, \ldots, a_{n-2}$.

不难发现，一个节点在 Prufer 序列中出现的系数为它的度数减 1.

由此可以确定树中所有叶子，设标号最小的叶子为 u，那么与 u 相邻的点为 a_1.

考虑 Prufer 序列 $a_2, \ldots, a_{n-2}$，标号集合为 $\{1, \ldots, n\} \setminus \{u\}$. 由归纳假设，存在唯一的树与之对应，设这棵树为 T'.

在 T' 中加入点 u 并与 a_1 连边，得到树 T，显然这是唯一的与 a 对应的树.

□

以上的证明同时给出了一个由 Prufer 序列构造对应树的方法，即从后向前构造，每次加一个叶子.

如果要从所有有标号无根树中等概率随机一个，只需要随机它的 Prufer 序列即可. 这种生成方式还可以固定每个点的度数.

2.1.4 方法 4：链

对于点 i（$2 \leq i \leq n$），选取 $i-1$ 作为 i 的父亲，即可生成一条链.

2.1.5 方法 5：完全二叉树

对于点 i（$2 \le i \le n$），选取 $\lfloor i/2 \rfloor$ 作为 i 的父亲，即可生成一棵完全二叉树.

这样生成的树，如果进行树链剖分，所有点的轻儿子的大小和为 $\Theta(n \log n)$ 级别.

2.1.6 方法 6：结合链与完全二叉树

我们可以取两个阈值 B_1, B_2，取前 B_1 个点构造完全二叉树，将后面每 B_2 个点连成一条链，连接在前 B_1 个点上.

一般 B_1 和 B_2 都会选取在 $\Theta(\sqrt{n})$ 级别.

一种常见的取值方式是取 $B_1 = \lfloor 2\sqrt{n} \rfloor$, $B_2 = \lfloor \sqrt{n} \rfloor$. 在下面的 2.1.7 小节中我们也将这样取值.

这样生成的数据既有较长的链，所有点的轻儿子大小和又为 $\Theta(n \log n)$.

2.1.7 对比

接下来对比六种生成方法.

我们先生成一棵大小为 n 的树，钦定点 1 为根. 然后将每个点的邻边随机打乱，进行 dfs. 最后进行 Q 次查询，每次随机两个点 u, v，查询 u 到 v 的路径信息.

上述过程会进行 T 次，然后测得以下数值，T 次取平均值：

- S_{size} 表示所有点子树大小之和. 部分高复杂度的算法依赖于此.
- S_{son} 表示进行轻重链剖分后，所有点的轻子树大小之和. 可以用来评估启发式合并的效率.
- S_{random} 表示每个点随机选择一个点作为重儿子后，所有点的轻子树大小之和. 可以用来评估错误的启发式合并方法的效率.
- S_{len} 表示 Q 次查询的路径长度平均值. 部分高复杂度的算法依赖于此. 也可以用来评估链上数据结构的效率.
- S_{jump} 表示进行轻重链剖分后，Q 次查询路径上的轻边个数平均值. 可以用来评估树链剖分的效率.

取 $n = 10^5$, $Q = 10^5$, $T = 10^3$，测量结果如下（保留三位小数）：

	$S_{\text{size}}=$	$S_{\text{son}}=$	$S_{\text{random}}=$	$S_{\text{len}}=$	$S_{\text{jump}}=$
方法 1	1212656.229	400679.705	873426.103	20.198	7.720
方法 2（$B=10$）	909341143.306	116601.017	300801584.156	6069.061	2.332
方法 2（$B=20$）	476433950.830	157012.831	166402833.765	3192.626	3.139
方法 2（$B=40$）	244176851.716	198905.253	87602621.886	1664.193	3.972
方法 3	39602294.635	325800.562	14507184.812	394.728	6.245
方法 4	5000050000.000	0.000	0.000	33333.811	0.000
方法 5	1568946.000	715030.000	734861.455	27.184	13.525
方法 6	16675299.000	396342.000	418815.824	328.686	7.168

可以看到：

- 方法 1：优点是代码简短，S_{son} 较大. 缺点是 S_{random}，S_{len}，S_{jump} 都过小，难以测试例如启发式合并和树链剖分的复杂度问题.
- 方法 2：S_{size}，S_{random}，S_{len} 较大，并且随着 B 减小有增加趋势.
- 方法 3：在测试的数据类型中各项数值大小较为居中.
- 方法 4：$S_{\text{size}},S_{\text{len}}$ 明显较大，链比较适用于测试链上数据结构在极限大小下的效率.
- 方法 5：$S_{\text{son}},S_{\text{jump}}$ 较大，比较适用于测试启发式合并、树链剖分的效率.
- 方法 6：强度介于方法 4 和方法 5 之间.

2.2 任意图

生成 n 个点 m 条边的无重边自环的无向图，就是要在 $\binom{n}{2}$ 条边中选择 m 条，使用之前序列的技巧即可.

如果要生成连通图，可以不断生成直到它连通为止.

2.3 仙人掌

仙人掌就是每条边至多在一个简单环中的图. 本文认为仙人掌可以由重边但不能有自环.

感性地理解，仙人掌就是若干环串成了一棵树的形状. 树也是一种仙人掌，所以仙人掌常常被作为树的扩展.

下面介绍两种随机仙人掌的方法.

2.3.1 方法 1

考虑一张连通图，对它做如下转化. 建一张新图，包含原图的所有点，称为**圆点**，对原图的每个点双连通分量新建一个点，称为**方点**（特别地，我们认为点双连通分量可以只包含两个点和一条边）. 每个方点向它在原图中包含的点连边，得到的必然是一棵树，称这棵树为原图的**圆方树**.

容易发现，圆方树满足如下性质：

- 每条边必然连接一个圆点与一个方点. 换句话说，圆点和方点构成了这张图的一个黑白染色.
- 所有度数小于等于 1 的点是圆点.

对于仙人掌，它的圆方树相当于对每个环建了一个方点.

如果要随机一个仙人掌，我们可以先随机一棵树，然后将其黑白染色，作为它的圆方树.

如果要生成的圆方树点数为 n，我们可以先随机一棵点数为 $2n$ 的树，黑白染色后取数量较少的颜色为圆点，那么圆点个数一定小于等于 n.

因为圆方树的叶子必须是圆点，我们需要将所有作为叶子的方点删除.

2.3.2 方法 2

先随机一棵树，然后在这棵树上加一些边，使其成为仙人掌.

对这棵树 dfs，回溯的时候，以一定概率向一个随机的祖先连边，并将这条路径上的边打标记，表示它们已经在一个环上了. 问题是如何确定这个概率以及连边的祖先.

我们可以设定一个阈值 P. 在点 u 处回溯时，可以以如下方法确定向祖先的连边：[2]

[2]参考自 [3]

算法 5 连边

```
v ← u
while v 不为根且 v 的父边未标记 do
    x ← (一个[0, 1]内的随机实数)
    if x < P then
        v ← (v的父亲)
    else
        break
    end if
end while
if u ≠ v then
    连边 (u, v)
    将 u 到 v 的路径打上标记
end if
```

例如取 $P = 0.1$，生成出的环会比较小.

2.3.3 方法 3

依然考虑先随机一棵树，然后加边的方式.

我们取一个参数 K，然后随机 K 次，每次随机两个点 u, v，判断是否可以在 u, v 间连一条边，如果可以就连.

判断方法与方法 2 类似，在每条树边上打标记，表示这条边是否已经在环上. 判断时只需检查 u 到 v 的路径上是否有带标记的边.

如果要保证生成大规模数据的时间复杂度，需要使用数据结构维护标记. 但实际上，沿路径逐步判断，遇到标记就退出，其效率也较高.

2.3.4 对比

我们测试 T 次，每次随机一个仙人掌，然后测得以下数值，T 次取平均值：

- N 表示节点个数.
- C 表示环的个数.
- $\overline{L}$ 表示环长平均值.
- σ 表示环长方差.

- M 表示环长最大值.

取 $n = 10^5$, $T = 10^3$，生成树的方法是均匀随机 Prufer 序列（即 2.1 中的方法 3），测量结果如下（保留三位小数）：

	$N =$	$C =$	$\overline{L} =$	$\sigma =$	$M =$
方法 1	99909.960	44791.272	2.819	0.735	8.761
方法 2（$P = 0.1$）	100000.000	9889.691	2.110	0.122	5.728
方法 2（$P = 0.2$）	100000.000	19072.550	2.243	0.301	7.814
方法 2（$P = 0.4$）	100000.000	32611.841	2.567	0.853	11.825
方法 3（$K = 10^5$）	100000.000	221.323	35.973	2280.336	479.667

可以看到：

- 方法 1：可以通过每个点的度数控制环长的分布，但会略微减小 N. 在均匀随机 Prufer 序列时，环长较短，环的个数较多，方差较小.
- 方法 2：可以通过调节 P 来调节环长，在 P 增加时，$C, \overline{L}, \sigma, M$ 都有增加趋势. 总体上环长较短，环的个数较多，方差较小.
- 方法 3：特点是环的个数较少，环长较长，方差较大.

3 括号串

括号串是由 (和) 两种字符构成的字符串.

对于一个括号串 $s = s_1 s_2 \dots s_n$，定义它的**路径序列** $a_0, a_1, a_2, \dots, a_n$ 如下：

$$a_i = \begin{cases} 0 & i = 0 \\ a_{i-1} + 1 & i > 0,\ s_i = (\\ a_{i-1} - 1 & i > 0,\ s_i =) \end{cases}$$

定义 s 是**平衡的**，当且仅当 $a_n = 0$.

定义 s 的**缺陷度** defect(s) 为满足 $\min(a_i, a_{i+1}) < 0$ 的 i 的个数除以 2 的结果（显然其个数总是偶数）.

定义 s 是**合法的**，当且仅当 s 的平衡的，且 s 的缺陷度为 0.

定义 $D_{n,k}$ 为长度为 n，缺陷度为 k 的**平衡**括号串集合.

定义 s^R 表示 s 反转后的串.

对于偶数 n，$D_{n,0}$ 就表示长度为 n 的合法括号串集合，根据卡特兰数有 $|D_{n,0}| = \frac{1}{n/2+1}\binom{n}{n/2}$

对于一个**平衡**括号串 s，定义一种变换 $\Phi(s)$ 如下：[3]

- 如果 s 为空，那么 $\Phi(s)$ 为空串.
- 如果 s 非空，那么一定可以将其分解为 $s = lr$ 的形式，其中 l 为 s 最短的非空平衡前缀，容易发现 r 也是一个平衡串.

  可以发现，$|l|$ 就是最小的 i 满足 $i > 0,\ a_i = 0$.

  然后讨论 l 的合法性：

  - 如果 l 是合法的，那么 $\Phi(s) = l\,\Phi(r)$.
  - 否则，l 一定可以表示为 $l =$) t (的形式，那么 $\Phi(s) =$ ($\Phi(r)$) t^R

容易证明，$\Phi(s)$ 是一个合法括号串，即 $\Phi(s) \in D_{n,0}$.

引理 3.0.1. 对于 n 为偶数，$0 \le k \le n/2$，Φ 建立了 $D_{n,k}$ 到 $D_{n,0}$ 的双射.

证明. 首先证明单射.

考虑对 n 归纳，命题对于 $n = 0$ 显然成立.

考虑两个串 $s_1, s_2 \in D_{n,k}$ 且 $s_1 \neq s_2$，分别将其分解为 $s_1 = l_1 r_1$，$s_2 = l_2 r_2$ 的形式，然后分类讨论.

- 情况 1：l_1, l_2 都合法.

  此时 l_1, l_2 分别为 $\Phi(s_1), \Phi(s_2)$ 的最短非空平衡前缀，如果 $l_1 \neq l_2$ 那么命题成立，否则一定有 $r_1 \neq r_2$，根据归纳假设有 $\Phi(r_1) \neq \Phi(r_2)$，命题成立.

- 情况 2：l_2, l_2 都不合法.

  此时 ($\Phi(r_1)$), ($\Phi(r_2)$) 分别为 $\Phi(s_1), \Phi(s_2)$ 的最短非空平衡前缀，然后同理情况 1，命题成立.

- 情况 3：l_1 合法，l_2 不合法. 设 $l_2 =$) t_2 (.

  此时 l_1， ($\Phi(r_2)$) 分别为 $\Phi(s_1), \Phi(s_2)$ 的最短非空平衡前缀.

  假设 $\Phi(s_1) = \Phi(s_2)$，那么一定有 $\Phi(r_1) = t_2^R$.

  因为 l_1 合法，所以 $k = \text{defect}(s_1) = \text{defect}(r_1)$，所以 $2k \le |r_1| = |\Phi(r_1)| = |t_2^R| < |l_2|$，也就是 $2k < |l_2|$.

  又因为 l_2 是 s_2 最短的非空平衡前缀，所以一定有 $k = \text{defect}(s_2) \ge \text{defect}(l_2)$，也就是 $2k \ge |l_2|$，矛盾.

  所以 $\Phi(s_1) \neq \Phi(s_2)$，命题成立.

[3] 参考自 [4]

- 情况 4：l_1 不合法，l_2 合法. 和情况 3 相同.

然后证明满射.

因为有单射，所以对于所有 $0 \le k \le n/2$ 有 $|D_{n,k}| \le |D_{n,0}|$. 我们只需要证明 $|D_{n,k}| = |D_{n,0}|$.

假设存在 $0 \le k \le n/2$ 满足 $|D_{n,k}| < |D_{n,0}|$，那么

$$\binom{n}{n/2} = \sum_{k=0}^{n/2} |D_{n,k}| < (n/2+1)|D_{n,0}| = \binom{n}{n/2}$$

矛盾.

综上，对于所有 $0 \le k \le n/2$，Φ 建立了 $D_{n,k}$ 到 $D_{n,0}$ 的双射.

□

有了引理 3.0.1，就可以十分方便地随机一个合法括号串了.

只需要在所有 $\binom{n}{n/2}$ 个平衡串中等概率随机一个，然后对其使用 Φ 即可.

4 哈希

哈希是信息学竞赛中十分常用的算法，通过较为随机的方式压缩信息，来加速信息的比较.

通常情况下，要使哈希出错，必须要针对代码构造数据.

然而有一些哈希方法是不需要针对代码，可以直接构造使其出错的数据的，而对拍用到的随机数据通常不足以发现这类问题.

下面将列出几个常见的、容易被构造出错的哈希算法.

4.1 生日悖论

一个房间中有 23 个人，假设一年总是有 365 天，且每个人的生日在这 365 天中独立等概率随机选取，问：存在两个人生日在同一天的概率是多少？

假设一年有 n 天，房间中有 m 个人，那么这 m 个人生日**互不相同**的概率为

$$\prod_{i=1}^{m} \frac{n-i+1}{n}$$

代入 $n = 365$, $m = 23$ 可得概率约为 49%，也就是存在两个人生日相同的概率大于 50%.

这是十分反直觉的，故我们称这种现象为**生日悖论**.

那么 m 要达到多少才能让存在两个人生日相同的概率大于等于 50% 呢？

可以列出如下方程：

$$\prod_{i=1}^{m} \frac{n-i+1}{n} \le \frac{1}{2}$$

由 $1+x \le e^x$ 可以进行估计：

$$\prod_{i=1}^{m} e^{-\frac{i-1}{n}} \le \frac{1}{2}$$

也就是

$$e^{-\frac{m(m-1)}{2n}} \le \frac{1}{2}$$

可以得到 $m \approx \sqrt{2n \ln 2}$.

一般的哈希方法是选择模数 M，将需要哈希的信息压缩到 $[0, M-1]$ 内的整数.

这个过程可以近似地看做在 $[0, M-1]$ 内随机一个整数.

如果你要处理的问题形如“判断 n 条信息内有无重复”，那么当 n 约等于 $\sqrt{2M \ln 2}$ 时，将无重复信息判为有重复的概率会达到 $\frac{1}{2}$.

一般的单哈希模数在 10^9 级别，在 n 达到约 3×10^4 时，就有约 36% 的概率出现冲突，这个概率是非常大的.

这种情况下，应选用两个互质的模数 M_1, M_2，此时随机的值域就可以看做 $M_1 M_2$，将冲突概率大大减小.

4.2 自然溢出

在做字符串哈希时，通常会选用模数 M 和进制 B，对于字符串 $s = s_1 s_2 \dots s_n$，它的哈希值计算为：

$$\text{hash}(s) = \left(\sum_{i=1}^{n} B^{i-1} \text{ord}(s_i) \right) \bmod M$$

这里 $\text{ord}(c)$ 表示字符 c 在码表中的位置.

C++ 中的无符号 64 位整数在运算时会自动对 2^{64} 取模，如果选用 $M = 2^{64}$，就可以少一次取模运算，从而加快运行速度. 这种哈希方法被称为**自然溢出**.

但这种哈希方法可以很容易构造冲突.

如果 B 为偶数，那么 $B^{64} \bmod M = 0$，也就是 s 只有至多后 64 位是有效的，很容易构造冲突. 下面讨论 B 为奇数的情况.

考虑如下字符串 $s = s_1 s_2 \dots s_{2^k}$ 满足

$$s_i = \begin{cases} \underline{A} & \text{popcount}(i-1)\text{为偶数} \\ \underline{B} & \text{popcount}(i-1)\text{为奇数} \end{cases}$$

其中 popcount(x) 表示 x 二进制表示下 1 的个数.

对于字符串 t，设 t^F 表示将 t 中的 $\underline{A}$ 替换为 $\underline{B}$，同时将 $\underline{B}$ 替换为 $\underline{A}$ 后的结果.

考虑字符串序列 $t_0, t_1, \ldots, t_k$ 满足 $t_0 = \underline{A}$, $t_i = t_{i-1} + t_{i-1}^F$，那么 $s = t_k$.

设 $d = \text{ord}(\underline{A}) - \text{ord}(\underline{B})$，容易得到

$$\text{hash}(s) - \text{hash}(s^F) \equiv \sum_{i=1}^{2^k} (-1)^{\text{popcount}(i-1)} B^{i-1} d \pmod{M}$$

我们需要证明，在 k 足够大的时候，这个值在 mod$M$ 意义下总是等于 0.

设

$$f_i = \text{hash}(t_i) - \text{hash}(t_i^F)$$

容易得到

$$f_i = f_{i-1}(1 - B^{2^{i-1}})$$

而 $f_0 = d$.

可以得到

$$f_i = d \prod_{j=0}^{i-1} (1 - B^{2^j})$$

注意到

$$1 - B^{2^j} = (1 - B^{2^{j-1}})(1 + B^{2^{j-1}})$$

也就是 $1 - B^{2^{j-1}}$ 乘上一个偶数. 如果 $2^p \mid (1 - B^{2^{j-1}})$ 那么一定有 $2^{p+1} \mid (1 - B^{2^j})$.

那么 f_i 一定可以被 $2^1 2^2 \ldots 2^i = 2^{\frac{i(i+1)}{2}}$ 整除.

在 $k = 11$ 时，有 $2^{64} \mid f_k$，也就是 $\text{hash}(t_k) = \text{hash}(t_k^F)$.

4.3 树哈希

树哈希一般用于判断两棵有根树是否同构.

一种较为正确的树哈希方法是括号序，使用最小表示法的思想，让同构的树得到的括号序相同.

具体来说，一个子树 u 的哈希值可以通过如下方式递归求出：

1. 求出 u 的所有儿子的哈希值.

2. 将它们以（哈希值，子树大小）为关键字排序.

3. 按照排好的顺序，将 u 的所有儿子的括号序相接.

4. 在最外层添加一对括号.

5. 得到的序列就是 u 的括号序，其哈希值就是 u 的哈希值.

容易证明，同构的树得到的哈希值相同.

在第二步的排序关键字中加入子树大小，是为了防止子树中因生日悖论产生哈希值相等的、不同大小的子树，影响结果.

设树的节点数为 n，因为要排序，以上算法的复杂度为 $\Theta(n \log n)$.

还有一种复杂度为 $\Theta(n)$ 哈希算法广为人知.

设点 u 的子树大小为 size_u，儿子集合为 son_u.

我们生成一个序列 $W_1, W_2, \ldots, W_n$，一般采用质数序列或随机序列.

然后点 u 的哈希值计算如下：

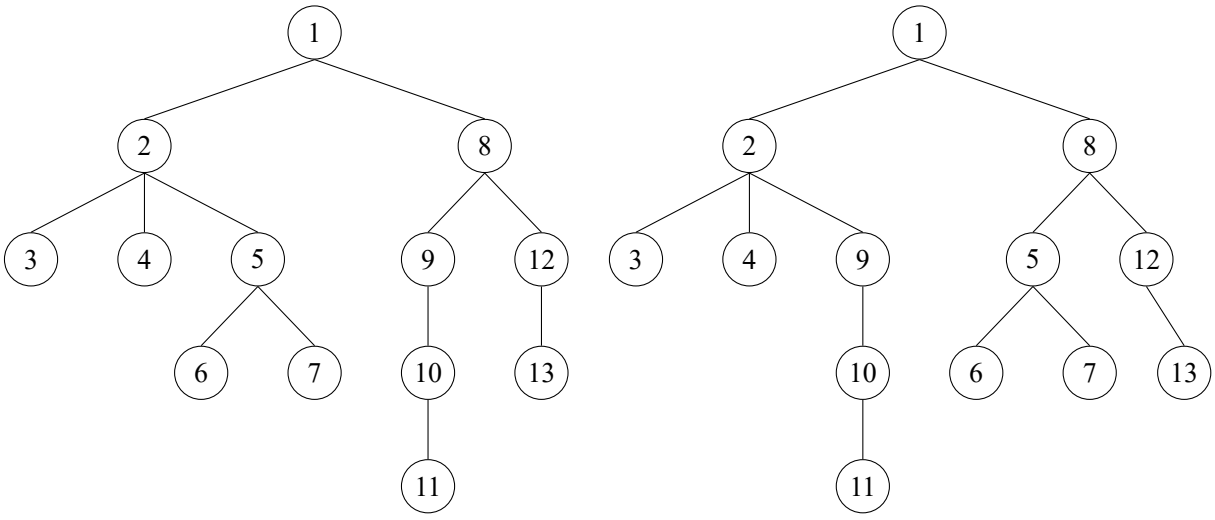
$$h_u = W_{\text{size}_u} \left(1 + \sum_{v \in \text{son}_u} h_v \right)$$

考虑点 u 到根节点的路径，将路径上的 size 依次写出得到序列 S_u.

注意到整棵树 T 的哈希值只与无序可重集合 $H(T) = \{S_1, S_2, \ldots, S_n\}$ 有关，集合相等的树得到的哈希值一定相等.

这种哈希的问题是，有些时候仅凭 $H(T)$ 是无法唯一确定树的形态的.

例如，无论 W 如何选择，下面两棵树的哈希值一定相等.



实际上，只需要交换两个 S 相等的子树，就可以保持 $H(T)$ 不变. 上图中交换了 5 和 9 的子树.

这组数据可以让所有只基于 $H(T)$ 的哈希算法出错.

5 总结

本文介绍的数据类型是在信息学竞赛中较为常见的.

序列、树等数据类型，虽然生成简单，但如果方法不当，就容易构造出强度不够的数据.

仙人掌、括号串等数据类型，在生成上就有一定难度.

而如果哈希方法不当，可能让选手在通过随机数据的前提下，仍然丢失分数.

更多时候，选手需要根据题目性质构造更具针对性的数据. 希望本文能引发选手思考，对测试程序、构造数据的方法有进一步的理解.

致谢

感谢父母对我的培养、关心与支持.

感谢中国计算机学会提供学习和交流的平台.

感谢国家集训队杨耀良教练和彭思进教练的指导.

感谢冷学农老师对我的培养与教导.

感谢给予我鼓励与帮助的老师以及同学.

参考资料

[1] Wikipedia 琴生不等式，https://zh.wikipedia.org/wiki/%E7%B0%A1%E6%A3%AE%E4%B8%8D%E7%AD%89%E5%BC%8F

[2]「营业日志 2020.11.21」随机父节点树的期望树高, https://blog.csdn.net/EI_Captain/article/details/109910307

[3] 玩转仙人掌，https://vfleaking.blog.uoj.ac/blog/89

[4] M.D. Atkinson and J.-R. Sack, Generating binary trees at random, http://www.cs.otago.ac.nz/staffpriv/mike/Papers/RandomGeneration/RandomBinaryTrees.pdf

浅谈一类有界树宽问题

北京市十一学校 吴畅

摘 要

本文介绍了树宽相关概念，以及有界树宽图上的树分解与若干经典问题的优秀算法。

1 前言

树宽是计算机科学中的一个重要概念。许多图上问题的复杂度，都是随树宽快速增长的。而现实中的图（如道路网）的树宽通常很小。因此，给出宽较小的树分解算法与在树宽较小的图上求解一系列问题，成为了近年来备受关注的研究方向。信息学竞赛中也出现过树宽较小的特殊图上的问题，但树宽相关的概念与算法并没有被普及。

本文第二节给出了所需的定义；第三节列出了一些约定与基本性质；第四节分析了几种特殊图上的树宽，着重强调了广义串并联图的缩减过程与应用；第五节介绍了有界树宽的意义及一些经典问题的更优秀的算法；第六节阐述了 Bodlaender 提出的一种有界树宽下的树分解算法。

2 定义

定义 2.1 (树分解)**.** 对于无向图 $G = (V, E)$，若树 $T = (I, F)$ 和一组点集 $\{X_i \subseteq V \mid i \in I\}$ 同时满足下述三个条件，则称 (X, T) 是 G 的一个树分解：

- $\bigcup_{i \in I} X_i = V$
- 对于任意一条边 $(u, v) \in E$，存在 $i \in I$ 满足 $u \in X_i$ 且 $v \in X_i$
- 对于任意 $i, j, k \in I$，若 j 在 T 中 i, k 间的路径上，则有 $X_i \cap X_k \subseteq X_j$

定义 2.2 (宽)**.** 树分解 (X, T) 的宽为 $\max_{i \in F} |X_i| - 1$。

定义 2.3 (树宽)**.** 图 G 的树宽 $tw(G)$ 为所有 G 的树分解中最小的宽。

定义 2.4 (k-树)**.** k-树是由 $k + 1$ 个点的完全图，进行若干次加点，满足每次添加的点恰好与之前的 k 个点之间有边，且这 k 个点之间两两有边，生成的图。

定义 2.5 (不完全 k-树，Partial k-tree)**.** 不完全 k-树是任意 k-树的子图。

3 约定与基本性质

定理 3.1. $tw(G) \le k$ 当且仅当 G 是不完全 k-树。

定理 3.1 的证明较长，此处略去。

如无特殊说明，下文在 G 非空且无重边自环下讨论。

如无特殊说明，下文在 T 是有根树下的讨论为取任一点 $r \in I$ 作为 T 的根。

对 $i \in I$，记 dep_i 为 i 的深度，记 $\mathrm{subtree}(i)$ 表示 i 的子树构成的点集。

对 $u \in V$，记 $S_u = \{i \in I \mid u \in X_i\}$。

记 $N_G(u) = \{v|(u,v) \in E\}, d_G(u) = |N_G(u)|$，即邻居集与度数。

记 top_u 表示 S_u 在 T 上深度最浅的点（显然只有一个），记 $\mathrm{lev}_u = \mathrm{dep}_{\mathrm{top}_u}$。

引理 3.0.1. 对任意 $(u,v) \in E$，不妨设 $lev_u \ge lev_v$，有 $v \in X_{top_u}$。

证明. 因为 $S_u \subseteq \mathrm{subtree}(\mathrm{top}_u)$ 且 $S_u \cap S_v \ne \emptyset$，若 $\mathrm{top}_u \notin S_v$，则 $S_v \subset \mathrm{subtree}(\mathrm{top}_u)$，与 $\mathrm{lev}_u \ge \mathrm{lev}_v$ 矛盾。 □

引理 3.0.2. 若 G 是不完全 k-树且 $k < |V|$，$|E| \le k|V| - \frac{k(k+1)}{2}$。

证明. 记 V 中 lev 最大的点为 u。由引理 3.1，$d_G(u) \le k$。删掉 u 归纳得证。 □

称对 $G = (V,E)$ 进行 $(u,v) \in E$ 的缩边是指，将 (u,v) 从 E 中删去并将 u,v 合并为一个新点（原图中 u,v 的邻接边继承给新点）。

引理 3.0.3. 若对 $G = (V,E)$ 进行 $(u,v) \in E$ 的缩边得到新图 G'，$tw(G') \le tw(G)$。

证明. 设缩边得到的新点为 w。取任一 G 上树分解 (X,T)。对 $i \in I$，若 $\{u,v\} \cap X_i \ne \emptyset$，令 $Y_i = X_i - \{u,v\} + \{w\}$；否则，$Y_i = X_i$。易证 (Y,T) 是 G' 的树分解，且宽不超过 (X,T)。 □

4 特殊图

4.1 不完全 1-树

定理 4.1. 不完全 1-树是森林。

证明. 若 G 是不完全 1-树，显然 G 的每个连通块也是不完全 1-树。由引理 3.3，每个连通块是树，G 是森林。 □

若 $G = (V,E)$ 是森林，G 的宽为 1 的树分解也很容易得到。令 I 包含 $|V| + |E|$ 个节点，分别代表原图中的所有点和所有边。对所有 $(u,v) \in E$，设 $u,v,(u,v)$ 分别对应 $x,y,z \in I$，将 $(x,z),(y,z)$ 加入 F。最后再随便连边使其联通即可。对于 $i \in I$，若 i 代表原图中的点 u，令 $X_i = \{u\}$；否则，若 i 代表原图中的边 (u,v)，令 $X_i = \{u,v\}$。容易发现 $(X, T = (I,F))$ 是 G 宽为 1 的树分解。

4.2 不完全 2-树的缩减

考虑图 G 上的以下三种操作：

- 叠合重边。
- 删 0/1 度点。
- 缩 2 度点（将它的两个邻接点之间连一条边并把它删掉）。

称任意一次某种操作为一次**缩减**，记对 G 进行一次缩减得到的新图为 G'。

引理 4.2.1. *G 是不完全 2-树当且仅当 G' 是不完全 2-树。*

证明. 前两种操作易证。设第三种操作删除的点是 u，邻接点为 v, w。

先证若 G 是不完全 2-树，G' 一定也是。取 G 任一宽为 2 的树分解 (X, T)，只需令所有 $S_i := S_i - \{u\} + \{v\}$，满足 $u \in S_i$，即可获得一个 G' 的宽为 2 的树分解。

再证若 G' 是不完全 2-树，G 一定也是。取 G' 任一宽为 2 的树分解 (X, T)，取 $i \in S_v \cap S_w$，只需在 T 上添加一个和 i 连边的点 j，且 $X_j = \{u, v, w\}$，即可获得一个 G 的宽为 2 的树分解。 □

引理 4.2.2. *若 $G = (V, E)$ 是不完全 2-树，存在 $u \in V$ 使得 $d_G(u) \le 2$。*

证明. 记 V 中 lev 最大的点为 u。由引理 3.1，$d_G(u) \le 2$。 □

若 G 上连续的若干次缩减过程会使得 G 无法进行更多的缩减操作，称为极大缩减过程。

定理 4.2. *一个不完全 2-树上的任何极大缩减过程都可以将其删空。*

证明. 由引理 4.1、引理 4.2 显然。 □

注意到上述性质与另一个概念“广义串并联图”几乎一致。唯一的区别是，后者的缩减过程可以不考虑“删 0 度点”的操作。也就是说，“广义串并联图”是连通的不完全 2-树。

4.3 串并联树

上述缩减操作可以被刻画为树形结构。

（广义串并联图的）串并联树是一棵有根三叉树。每个叶子节点是原图中的一条边或一个点，任意时刻图中的一条边或一个点在串并联树上对应一个当前节点。每次缩减操作会在串并联树上新建一个节点 x。

- 叠合重边：将原来两条重边的当前节点的父亲设为 x，将新边的当前节点设为 x。
- 删 1 度点：设删除的是 u，邻接点为 v，将 $u, v, (u, v)$ 的当前节点的父亲设为 x，将 v 的当前节点设为 x。
- 缩 2 度点：设删除的是 u，邻接点为 v, w，将 $u, (u, v), (u, w)$ 的当前节点的父亲设为 x，将新边的当前节点设为 x。

4.4 基于缩减的动态规划

许多在任意图上较为困难的问题，在不完全 2-树上可以应用缩减过程进行动态规划，获得优秀的复杂度，此举一例[1]。

4.4.1 题目大意

给定 n 个点 m 条边的无向连通图 G，已知 G 在删掉一条边后是一颗仙人掌，求 G 的生成树个数，答案对 998244353 取模。

4.4.2 数据范围

$1 \le n \le m \le 5 \times 10^5$

4.4.3 解题过程

可以证明满足条件的图是广义串并联图，在此不多赘述。记 G 的边集为 E。

考虑上述缩减过程，在某个时刻，对 $e = (u, v) \in E$，记录 f_e 表示缩减为 e 的过程（也就是 e 在串并联树上的子树）中使得 u, v 联通的方案数，g_e 表示缩减为 e 的过程中使得 u, v 不联通的方案数，注意两者都需要使得缩减为 e 的过程中被删掉的点至少与 u, v 之一联通。

对于删 1 度点。此时这个点的邻边 e 必须出现在生成树上，将最终答案乘上 f_e 即可。

对于叠合重边，设两条重边分别为 e_1, e_2，叠合出来的边为 e_0。显然两条边不能同时联通，$f_{e_0} = f_{e_1} g_{e_2} + f_{e_2} g_{e_1}$，$g_{e_0} = g_{e_1} g_{e_2}$。

对于缩 2 度点，设两条邻边分别为 e_1, e_2，新添加的边为 e_0。显然两条边不能同时不连通，$f_{e_0} = f_{e_1} f_{e_2}$，$g_{e_0} = f_{e_1} g_{e_2} + f_{e_2} g_{e_1}$。

复杂度 $O(n + m)$。

容易发现上述算法实质就是在串并联树上由底向上做树形 dp。

4.5 其它特殊图

完全图 K_n 的树宽为 $n - 1$。

平面图的树宽是根号级别的。

仙人掌，outerplanar graphs 的树宽不超过 2。

Halin graphs，Apollonian networks 的树宽不超过 3。

弦图的树宽等于最大团的大小 -1，可以在多项式时间内计算。

值得一提的是，树宽的等价定义之一是包含原图的弦图中最小的最大团大小 -1。

[1] [SNOI2020] 生成树

5 有界树宽

5.1 寻找宽较小的树分解

前文已提到，许多图上问题可以通过给出宽较小的树分解取得优秀的复杂度。而计算任意图的树宽被证明是 NP 完全的。在此背景下许多从不同角度出发的树分解算法被陆续提出。这里先简单罗列一些学术成果。

定理 5.1 ([1]). 对于 n 个点的树宽为 k 的图，存在给出宽为 k 的树分解的算法，时间复杂度为 $n^{k+O(1)}$。

定理 5.2 ([2]). 对于 n 个点的树宽为 k 的图，存在给出宽为 k 的树分解的算法，时间复杂度为 $2^{\tilde{O}(k^3)}n$。

定理 5.3 ([3]). 对于 n 个点的树宽为 k 的图，存在给出宽不超过 $5k+4$ 的树分解的算法，时间复杂度为 $2^{O(k)}n$。

定理 5.4 ([4]). 对于 n 个点的树宽为 k 的图，存在给出宽为 $O(k\sqrt{\log k})$ 的树分解的算法，时间复杂度为 $n^{O(1)}$。

在任意图上的树宽，也有许多效率不尽相同的算法。不难注意到，这些算法大部分随树宽快速增长。除了各种搜索以外，还有大量的算法是通过缩小问题规模、定义转化等来寻找树分解。

值得一提的是，其它一些算法通过给出次优的树分解减小复杂度，例如在上述最后一个定理中，复杂度是多项式的。

5.2 有界树宽

注意到定理 5.2 的复杂度关于 n 都是线性的。

换句话说，对于任意常数 k，判断给定图是否具有宽为 k 的树分解（如果有则给出构造）是线性的。我们将在第 6 节简单分析这个算法。

另外，如果限制给定的图的树宽有界，也就是不超过某个常数 k，许多 NP 问题都能在多项式时间内解决。其中，对于一大类在树上易于解决的问题，可以在给定的图的树分解上动态规划，取得优秀的复杂度。这便是研究有界树宽的一个重要意义。

定理 5.5 (Courcelle's theorem,[5]). 如果一个图论问题可以被表达为一元二阶逻辑，那么在有界树宽图上它可以用线性时间被解决。

例如：染色问题，哈密顿回路问题，最大团问题，最大独立集问题。

我们将在 5.4 节演示有界树宽图上的最大独立集问题。

5.3 完美树分解

基于一般树分解的动态规划，通常较为麻烦。但如果给出的树分解满足一些性质，动态规划将更容易展开。

定义 5.1 (完美树分解)**.** 称 $G = (V, E)$ 的一个树分解 (X, T) 是完美的，当且仅当 T 是一棵有根二叉树，且每个节点 $i \in I$ 满足下述四种情况之一（v 是任意 V 中元素）：

- **叶子** 没有儿子且 $|X_i| = 1$。
- **介入点** 有一个儿子 j 且 $X_i = X_j \cup \{v\}$（v 称为介入元素）。
- **遗忘点** 有一个儿子 j 且 $X_j = X_i \cup \{v\}$（v 称为遗忘元素）。
- **合并点** 有两个儿子 j, k 且 $X_i = X_j = X_k$。

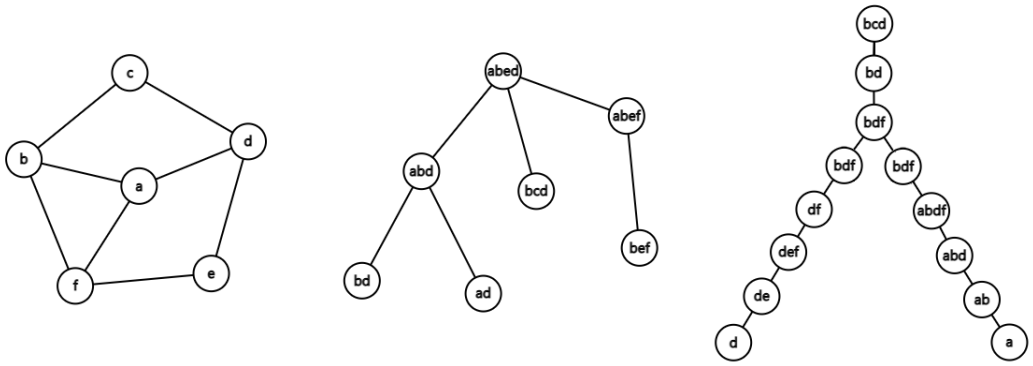


图 1: 原图，树分解，完美树分解

定理 5.6. 给定 G 的一个宽有界的树分解，存在算法在线性时间内，给出一个宽相同的完美树分解。

此处简要陈述一下定理 5.6 中的算法过程。

如果某个节点 i 没有儿子，且 $|X_i| > 1$，取 $v \in X_i$，新建节点 j 作为 i 的儿子并令 $X_j = X_i - \{v\}$。

如果某个节点 i 有一个儿子 j，且 $|(X_i - X_j) \cup (X_j - X_i)| > 1$，不妨设 $X_i - X_j \neq \emptyset$ 且 $v \in X_i - X_j$。在 (i, j) 之间插入一个新节点 l 并令 $X_l = X_j \cup \{v\}$。重复上述操作直到满足条件。

如果某个节点 i 有两个儿子，且某个儿子 j 不满足 $X_j = X_i$，可以在 (i, j) 之间插入一个新节点 l 并令 $X_l = X_i$。

如果某个节点 i 有超过两个儿子。记其中两个为 j, k，新建节点 l 作为 j, k 的新父亲，同时为 i 的儿子，并令 $X_l = (X_i \cap X_j) \cup (X_i \cap X_k)$。重复上述操作直到满足条件。

5.4 最大独立集问题

定义 5.2 (独立集)**.** 称 $\mathcal{I}$ 是无向图 $G=(V,E)$ 的独立集，当且仅当 $\mathcal{I}\subseteq V$ 且 $\forall u,v\in\mathcal{I},(u,v)\neq E$。

顾名思义，最大独立集问题是求解图中集合大小最大的独立集。

若已知 G 上宽为常数的完美树分解 (X,T)，使用下述算法可以在线性时间内求解最大独立集问题。

由定义，任意 $(u,v)\in E$，都存在 $i\in I$ 满足 $u\in X_i$ 且 $v\in X_i$。因此 $\mathcal{I}\subseteq V$ 是独立集当且仅当任意 $i\in I$，$\mathcal{I}\cap X_i$ 是独立集。

据此，考虑一个自底向上的树形 dp 的过程。

设 $dp_{i,S}$ 表示在节点 i，钦定 X_i 中的节点只有 $S\subseteq X_i$ 被选进独立集（若 S 不是独立集则未定义，即价值无限小）。

若 i 是叶子：$dp_{i,\emptyset}=0,dp_{i,X_i}=1$。

若 i 是介入点：（设介入元素为 v，儿子为 j）$dp_{i,S}=dp_{j,S-\{v\}}+[v\in S]$。

若 i 是遗忘点：（设遗忘元素为 v，儿子为 j）$dp_{i,S}=\max(dp_{j,S},dp_{j,S\cup\{v\}})$。

若 i 是合并点：（设儿子为 j,k）$dp_{i,S}=dp_{j,S}+dp_{k,S}-|S|$。

设树分解宽为 w，时间复杂度为 $O(2^w w^2 n)$。

5.5 最短路径问题

此处研究的最短路径问题指的是在不带权无向图上，多次询问两个点 s,t 间的最短路径。

定理 5.7. 假设 (X,T) 是 $G=(V,E)$ 的树分解。对于任意 $s,t\in V$，记 m 表示 top_s 和 top_t 在 T 上的最近公共祖先。若将 X_m 中的点从 G 中删除，s,t 不连通。

最短路径显然会经过 X_m 中的至少一个点，因此 $dis(s,t)=\min\limits_{i\in X_m} dis(s,i)+dis(i,t)$。

记树高为 h，树分解宽为 w。我们可以通过 bfs 在 $O(hn)$ 的时间内预处理每对祖先关系的点对间的最短路径，然后 $O(w)$ 查询。

6 Bodlaender 的树分解算法

本节会简要介绍 Bodlaender 在 1996 年提出的有界树宽图上的线性树分解算法（定理 5.2,[2]）。

由于整个算法比较长，且作者对这个算法的认识也非常浅薄，本节只会介绍 Bodlaender 论文中的重点工作，部分定理略去证明。尽管此处无法给出完整的算法，但其思想与副产品仍具有教育意义。

6.1 总述

引理 6.1.1 ([6])**.** 对于任意有界常数 k, l，若给出 G 的宽为 l 的树分解，判断 G 的树宽是否不超过 k，如果是则给出宽不超过 k 的树分解。上述问题存在线性做法。

此处不给引理 6.1.1 的证明，感兴趣的同学可以自行查阅资料。

定理 6.1.1. 对于任意有界常数 k，判断 G 的树宽是否不超过 k，如果是则给出宽不超过 k 的树分解。上述问题存在线性做法。

本节的内容为基于引理 6.1.1 的正确，介绍定理 6.1.1 中的算法。

这个算法的基本想法是，在线性复杂度内，以一定比例缩小问题规模。

分为以下两种情况讨论：（“绿点”与“I-单形点”分别在 6.2 和 6.3 提及）

1. 若“绿点”“足够”多，可以证明 G 的极大匹配较大。考虑将极大匹配中的边缩起来递归处理。

2. 否则，在对 G 进行不改变树宽的变形后，可以证明“I-单形点”“足够”多。考虑将所有“I-单形点”删去递归处理。

在两种情况中，问题规模都以某个固定比例缩小，因此总复杂度为线性。

6.2 节会给出后面需要用到的一些引理和定义，6.3 节主要考虑第一种情况，6.4 节主要考虑第二种情况，6.5 节完整表述整个算法。

6.2 铺垫

引理 6.2.1. 假设 (X, T) 是 $G = (V, E)$ 的树分解。

(1) 若 $W \subseteq V$ 在 G 上形成了完全图（团），存在 $i \in I, W \subseteq X_i$。

(2) 若 $W_1, W_2 \subseteq V$ 在 G 上形成了完全二分图（所有 W_1 中的节点都和所有 W_2 中的节点相邻），存在 $i \in I$，$W_1 \subseteq X_i$ 或 $W_2 \subseteq X_i$。

证明. (1) 任取叶子 $i \in I$。若存在 $u \in W, S_u = \{i\}$，则 $W \subseteq X_i$。否则，可以将 i 删掉，对于 W 仍是合法的树分解。

(2) 由 (1)，若不存在 $i \in I, W_1 \subseteq X_i$，则存在树边 $(i, j) \in F$ 满足存在 $x, y \in W_1$，使得 S_x, S_y 分属于从 T 上删掉 (i, j) 得到的两个不同的子树中。因此，则 $W_2 \subseteq X_i$。 □

引理 6.2.2. 假设 (X, T) 是在 $G = (V, E)$ 的树分解，假设 $u, v \in V$ 满足 $\exists i \in I, \{u, v\} \subseteq X_i$，$(X, T)$ 也是 $G + (u, v)$ 的树分解。

引理 6.2.3. 对 $G = (V, E)$，假设 $u, v \in V$ 满足 $|N_G(u) \cap N_G(v)| > k$，$tw(G) \leq k$ 当且仅当 $tw(G + (u, v)) \leq k$。

证明. 记 $W = N_G(u) \cap N_G(v)$。若存在 $i \in I, W \subseteq X_i$，由引理 6.2.1，不妨将 W 中元素两两间连边加入 G，此时会形成一个大小超过 $k+1$ 的团 $W \cup \{u\}$，矛盾。否则，存在 $i \in I, \{u, v\} \subseteq X_i$。 □

定义 6.2.1 (均匀树分解,smooth tree-decomposition)**.** 称 $G = (V, E)$ 的一个树分解 (X, T) 是均匀的，当且仅当 $\forall i \in I, |X_i| = k + 1$ 且 $\forall (i, j) \in F, |X_i \cap X_j| = k$。

引理 6.2.4. 给定 G 的一个宽有界的树分解，存在算法在线性时间内，给出一个宽相同的均匀树分解。

算法比较简单，与上文完美树分解的转换有相似之处，此处不再赘述。

6.3 绿点

我们挑选两个常数 $0 < c_1, c_2 < 1$ 满足

$$c_2 = \frac{1}{4k^2 + 12k + 16} - \frac{c_1 k^2 (k + 1)}{2}$$

其中 c_1 用来将节点分类，而 c_2 表示了第二种情况中问题规模缩小的比例。你暂时不用关心 c_2，只需要保证上述等式成立的情况下 c_1, c_2 都是 $(0, 1)$ 的常数。

令 $d = \max(k^2 + 4k + 4, \lceil \frac{2k}{c_1} \rceil)$。称度数不超过 d 的顶点是**低度点**，度数超过 d 的顶点是**高度点**，和另外至少一个低度点相邻的低度点是**绿点**。

引理 6.3.1. 高度点的数量少于 $c_1|V|$。

设高度点的数量为 n_l，根据引理 3.2，$\frac{n_l d}{2} < k|V|$。

定义 6.3.1 (极大匹配)**.** 匹配是端点不交的边集。如果一个匹配不是任何其它匹配的真子集，称其为极大匹配。

容易通过贪心线性求出任意图的一个极大匹配。

引理 6.3.2. 假设 G 有 n_f 个绿点，G 的任意极大匹配包含至少 $\frac{n_f}{2d}$ 条边。

证明. 若 M 是极大匹配，任意绿点 v 要不就出现在 M 的某条边上，要不就和 M 上的一个绿点 u 相邻。如果是前者，我们将 v 的贡献记在自己头上；如果是后者，我们将 v 的贡献记在 u 头上。由于 M 上的一个绿点只能被算 d 次贡献，因此 M 至少有 $\frac{n_f}{d}$ 个绿点，因此 $|M| \geq \frac{n_f}{2d}$。 □

将 G 对任意匹配 M 缩边，假设得到的新图为 G'。显然可以根据 G' 任意宽为 k 的树分解还原一个宽为 $2k + 1$ 的树分解：假设某一条边 (u, v) 被缩为 w，若 $i \in I, w \in X_i$，将 X_i 变为 $X_i - \{w\} + \{u, v\}$。且应用定理 6.1.1 中的算法可以将 $2k + 1$ 缩小至 k。

一方面，缩边不会增大树宽。另一方面，由缩边之后的新图的一个树分解，可以快速还原原图的树分解。因此，当绿点“足够”多时，可以考虑将极大匹配中的边缩起来递归处理，且这部分复杂度为线性。

6.4 I-单形点

定义 6.4.1. 给定图 $G=(V,E)$，对于所有 $u,v\in V$ 满足 u 和 v 有至少 $k+1$ 个相同的低度邻居，添加边 (u,v)，最终得到的新图 $G'=(V,E')$ 称为 G 的补充图。

由引理 6.2.3，$tw(G)\le k$ 当且仅当 $tw(G')\le k$。

定义 6.4.2. 若 v 在 G 的补充图 G' 上的邻居形成了团，且它在 G 上是低度点但不是绿点，则称它是 ***I*-单形的**。

定义 6.4.3. 若对 G 的某个树分解 (X,T)，$v\in V$ 是低度点但不是绿点，且存在 $i\in I$，满足 v 的所有邻居都属于 X_i，则称 v 是树分解 (X,T) 下的**红点**。

红点只是分析是需要用到，实际算法中不需要计算红点。

如无特殊说明，下文中的红点默认为给定树分解下的红点。

引理 6.4.1. 假设 (X,T) 是树宽为 k 的图 $G=(V,E)$ 的一个均匀树分解。

(1) 对 T 的每个叶子 i，都能找到 $v\in X_i$ 是绿点或红点，且不存在 $j\in I,j\ne i,v\in X_j$。

(2) 对 T 中所有长为 k^2+3k+4 的路径 $i_0...i_{k^2+3k+3}$，满足 $i_1...i_{k^2+3k+2}$ 的度数都为 2，都能找到 $v\in\bigcup_{j=1}^{k^2+3k+2}X_{i_j}$ 是绿点或红点，且不存在节点 $j\in I$，j 不在这条路径上且 $v\in X_j$。

证明. (1) 令 j 是 i 唯一的邻居。由均匀树分解的定义，X_i-X_j 中唯一的节点一定满足条件。

(2) 注意到 $|\bigcup_{j=0}^{k^2+3k+3}X_{i_j}|=k^2+4k+4\le d$，因此 $\bigcup_{j=1}^{k^2+3k+2}X_{i_j}-(X_{i_0}\cup X_{i^{k^2+3k+3}})$ 中的点都是低度点。如果它们都不是绿点，那么只能和 $X_{i_0}\cup X_{i_{k^2+3k+3}}$ 中的高度点有边。取出 X_{i_0} 中的所有高度点，记它们在路径上的最后一次出现位置分别为 $w_1...w_r$，显然有 $r\le k+1$。对于 $\bigcup_{j=1}^{k^2+3k+2}X_{i_j}-(X_{i_0}\cup X_{i^{k^2+3k+3}})$ 这至少 k^2+2k+2 个点，因为它们都不是绿点，所以必须出现在 $X_{w_1}...X_{w_r}$ 这至多 $k+1$ 个集合中的至少一个。由鸽巢原理，至少有一个集合大小大于 $k+1$，与假设矛盾。 □

定义 6.4.4. 称树 T 上的一个叶子-路径集合，由所有的叶子，以及若干长度为 k^2+3k+4 的路径构成。满足路径上的点不在别的路径上出现，且度数为 2。一个叶子-路径集合的大小是叶子的数量加上路径的数量。

引理 6.4.2. 任意含有 r 个点的树 T 存在一个大小不小于 $\frac{r}{2k^2+6k+8}$ 的叶子-路径集合。

证明. 令 r_b 表示度数至少为 3 的节点个数，r_l 表示叶子个数，r_2 表示度数为 2 的节点个数。

考虑将叶子和度数至少为 3 的节点从 T 中删去后，剩下的少于 r_b+r_l 条链每条链至多有 k^2+3k+3 个点不被选进集合中。

因此集合大小至少可以做到

$$r_l+\max(0,\frac{r_2-(r_b+r_l)(k^2+3k+3)}{k^2+3k+4})$$

结合树的基本性质 $r_b<r_l$，易证原命题。 □

推论 6.4.1. 若 G 具有宽为 k 的均匀树分解 (X,T)，那么至少有 $\frac{|V|}{2k^2+6k+8}-1$ 个绿点或红点。

证明. T 至少有 $|V|-k$ 个节点。根据引理 6.4.1，引理 6.4.2 易证。 □

定义 6.4.5. 若由高度点构成的集合 $Y\subseteq V$ 满足存在 $i\in I, Y\subseteq X_i$，称 Y 是半重要的。若半重要集 Y 不是任何另一个半重要集的子集，称 Y 是重要的。

引理 6.4.3. 若 G 具有树分解 (X,T)，重要集的数量不超过高度点的数量。

证明. 令 L 表示所有高度点构成的集合。对 $i\in I$，记 $X'_i=X_i\cap L$。则 (X',T) 是 G 在点集 L 下的诱导子图 G' 的树分解。不断进行：若存在 $(i,j)\in F$，$X'_i\subseteq X'_j$，将 (i,j) 缩成 j。

当无法进行上述操作时，设此时的树分解为 $(X',T'=(I',F'))$。所有重要集构成的集合 $\mathcal{Y}=\{X'_i\mid i\in I'\}$，只需说明 $|I'|\le|L|$。取出 T' 的一个叶子节点 i，设它的邻接点为 j，则 $X'_i-X'_j\neq\emptyset$。将 i 节点从树上删掉得到 $T''=(I'',F'')$ 并将 $X'_i-X'_j$ 从 L 中删掉得到 L'。归纳得到 $|I''|\le|L'|$，则 $|I'|=|I''|+1\le|L'|+1\le|L|$。 □

定义 6.4.6. 一个从每个红点映射向一个重要集的函数 f，满足 $\forall v\in V, N_G(v)\subseteq f(v)$，被称为 *hi*-函数。

引理 6.4.4. 假设 $G=(V,E)$ 具有宽为 k 的均匀树分解 (X,T)，令 f 是一个 *hi*-函数，Y 是一个重要集。$\{v\in V\mid f(v)=Y\}$ 中最多有 $\frac{1}{2}k^2(k+1)$ 个非 I-单形的红点。

证明. 设 $G'=(V,E')$ 为 G 的补充图。若 $v\in V, f(v)=Y$ 且 v 是非 I-单形的红点。存在 $x,y\in Y,(v,x)\in E',(v,y)\in E',(x,y)\neq E'$。而根据补充图的定义，$x,y$ 不能共享超过 k 个邻居。

因此，如果把 v 的贡献记在点对 (x,y) 头上，由于 Y 中的每个点对最多只能被记 k 次，最多只有 $k\binom{|Y|}{2}\le\frac{1}{2}k^2(k+1)$ 是非 I-单形的红点。 □

推论 6.4.2. 若 $tw(G=(V,E))\le k$，且含有 n_f 个绿点和 n_h 个高度点，那么至少有

$$\frac{|V|}{2k^2+6k+8}-1-n_f-\frac{1}{2}k^2(k+1)n_h$$

个 I-单形点。

引理 6.4.5. 假设 G' 是将所有 I-单形点从 G 的补充图中删除得到的图，(X,T) 是 G' 的一个树分解。对于删除的每个 I-单形点 v，存在 $i\in I, N_G(v)\subseteq X_i$。

证明. 由定义显然。 □

这个引理告诉我们，由删掉 I-单形点的图的树分解，可以快速还原原图的一个等宽的树分解。因此，当 I-单形点“足够”多时，可以考虑将 I-单形点删掉递归处理，且这部分复杂度为线性。

6.5 算法流程

先重新表述一下要解决的问题。

给定无向图 $G = (V, E)$ 和常数 k，给出宽不超过 k 的树分解，或告知 G 的树宽大于 k。

这里假设 $|V|$ 超过某个足够大的常数，否则可以用简单的搜索算法求解。

首先，检查 $|E| \le k|V| - \frac{1}{2}k(k+1)$ 是否成立，如果不是一定无解。

若绿点的数量不少于 $\frac{|V|}{4k^2+12k+16}$：

- 求 G 的极大匹配 M，将 M 中的每条边缩起来得到新图 $G' = (V', E')$。
- 对新图递归求解，无解则原图也无解。
- 由新图宽为 k 的树分解还原一个原图宽为 $2k+1$ 的树分解。
- 应用定理 6.1.1 的算法。

否则：

- 求 G 的补充图 G'。
- 若 G' 中存在 I-单形点的度数大于 k，无解。
- 求出所有 I-单形点构成的集合 S。若 $|S| < c_2|V|$，无解。否则，对从 G 中删去 S 得到的新图递归求解。
- 对删掉的每一个 I-单形点 v，找到 $i \in I, N_G(v) \subseteq X_i$，添加一个新点 j 与 i 相连，且令 $X_j = \{v\} \cup N_G(v)$。最后的结果是原图的一个宽为 k 的树分解。

分析一下复杂度。令

$$c_3 = \frac{1}{2d(4k^2+12k+16)}, c_2 = \frac{1}{4k^2+12k+16} - \frac{c_1k^2(k+1)}{2}, c_4 = \max(1-c_3, 1-c_2)$$

(c_2 定义同上)。

第一种情况会删掉至少 $c_3|V|$ 个点，第二种情况会删掉至少 $c_2|V|$ 个点。

注意到 $0 \le c_4 < 1$，而每次将点集大小至少缩小为原来的 c_4 倍，因此时间复杂度 $T(n) = T(c_4n) + O(n)$ 为线性。

7 总结

本文从几类特殊图出发，分析了树宽的一些特征和性质，并介绍了有界树宽在解决若干经典问题上的重要意义，最后阐述了一种优秀的树分解方法的核心思想。总的来说，本文只是半科普性质的对树宽相关知识作了基本介绍，至于其更广泛的联系和更强力的树分解方法仍等待我们探索。希望本文能够让读者对树宽有初步的了解，在未来可以进行更加深入的研究。期待其在信息学竞赛中的应用。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢北京市十一学校的汪星明老师的关心和指导。
感谢家人的关心和支持。
感谢与我交流讨论的同学。

参考文献

[1] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. Siam Journal on Algebraic and Discrete Methods, 8:277–284, 1987.

[2] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing, 25(6):1305–1317, 1996.

[3] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An $o(c^k n)$ 5-approximation algorithm for treewidth. In 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pages 499–508, 2013.

[4] Uriel Feige, Mohammad Taghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In Symposium on the Theory of Computing, 2005.

[5] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Information and Computation, 85(1):12–75, 1990.

[6] Hans L. Bodlaender and Ton Kloks. Better algorithms for the pathwidth and treewidth of graphs. In ICALP, 1991.

[7] Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, page 709–724, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Wikipedia. Treewidth.

对互异关系的容斥

华南师范大学附属中学 周子衡

摘 要

本文提出了处理带有“各个元素互异”条件时，一种利用容斥原理的新思路，并通过几个问题来具体阐述这一方法。

1 写在前面

1.1 前言

在信息学竞赛的计数问题中，常常遇到“各个元素互不相同”这一限制条件。通常来说，这需要大量细致的分析、推导。我们接下来将介绍一种基于容斥的方法，可以将这种问题部分地转化成没有互异限制的问题。

1.2 符号与约定

这里介绍一些本文中会用到的记号。

$a \equiv b \pmod{n}$：同余记号，表示存在某个整数 k 使得 $a + kn = b$。

$\binom{m}{n}$：组合数，在 m 个物体中选择 n 个，不考虑顺序的方案数。当 $0 \le n \le m$ 时有 $\binom{m}{n} = \dfrac{m!}{n!(m-n)!}$。

2 问题引入

2.1 问题的提出

我们从一道经典问题引入。

问题 给定奇质数 p 和正整数 k，试计算 $\{1,2,...,kp\}$ 有多少个 p 元子集的各元素之和是 p 的倍数？□

这个问题有一个较为简单的解法，我们将其放在本节的最后。下面来讲解如何运用针对互异关系的容斥来解决本题。

2.2 初步转化

考虑另一个问题。

问题 给定奇质数 p 和正整数 k，试计算正整数 $(x_1,x_2,...,x_p)$ 的组数，使得：

- **条件 A** $1\le x_i\le kp$。
- **条件 B** 对任意 $1\le i,j\le p$，$i\ne j$，都有 $x_i\ne x_j$。
- **条件 C** $\sum_{i=1}^{p}x_i\equiv 0\pmod p$。□

注意到上述这个问题实际上相当于原问题中集合元素之间的顺序也纳入到考量的范围中。对一个 p 元集合，任意排列它共有 $p!$ 种方案，因而显见本问题的答案是原问题的 $p!$ 倍。

考虑对条件 B 应用容斥原理。我们将在所有的 (i,j) 中钦定若干对，并强制这些对中的两个元素值相等。不妨将钦定关系看作一张图上的所有边，则全部限制条件等价于每个连通块内的元素值相等。不妨设 E_n 是 n 阶完全图 K_n 的边集，$V(E)$ 表示钦定了 E 的边集内的相等关系后，符合条件 A、C 的 $(x_1,...,x_p)$ 数量。则可知答案为：

$$\sum_{E\subset E_p}(-1)^{|E|}V(E)$$

2.3 $V(E)$ 的计算以及维度的上升

观察 设 $c(E)$ 表示 E 中的边形成的连通块数。那么

$$V(E)=\begin{cases}kp, & c(E)=1\\ (kp)^{c(E)-1}\times k, & c(E)>1\end{cases}\tag{1}$$

证明 $c(E)=1$ 时所有 p 个变量的取值全部相等，则这些变量都可以任意取值，共 kp 种。$c(E)>1$ 时，全部连通分量大小都与 p 互质，我们可以任取一个连通块，对每一种其他连通块的任意取值（总计 $(kp)^{c(E)-1}$ 种），该连通块变量的取值都被限定在一个唯一的剩余类里，

也就是 k 种。□

即使我们得到了这一公式，将其直接代入计算仍然有难度。我们需要一些巧妙的观察。注意到：对任意整数 $t \in \{1, 2, ..., p-1\}$ 和边集 E，我们仿照上面的做法计算在钦定 E 中的相等关系后，$\sum_{i=1}^{p} x_i \equiv t \pmod p$ 的方案数，记为 $V(E, t)$。方便起见，下面记 $V(E, 0) = V(E)$。可以观察到，当 $c(E) > 1$ 时，$V(E, t) = V(E, 0)$；当 $c(E) = 1$ 时 $v(E, t) = 0$。也就是说，

$$V(E,t) = \begin{cases} kp, & c(E) = 1, t = 0 \\ 0, & c(E) = 1, t > 0 \\ (kp)^{c(E)-1} \times k, & c(E) > 1 \end{cases} \tag{2}$$

我们再回到待解问题上。这次我们定义对每个剩余类都定义一个新问题，总共得到了 p 个问题，分别是问题 $0, 1, ..., (p-1)$：

问题 t 给定奇质数 p 和正整数 k，试计算正整数 $(x_1, x_2, ..., x_p)$ 的组数，使得：

- **条件 A** $1 \le x_i \le kp$。
- **条件 B** 对任意 $1 \le i, j \le p$，$i \ne j$，都有 $x_i \ne x_j$。
- **条件 C** $\sum_{i=1}^{p} x_i \equiv t \pmod p$。□

记这 p 个问题的答案分别为 $a_0, a_1, ..., a_{p-1}$。可知

$$a_t = \sum_{E \subset E_p} (-1)^{|E|} V(E, t)$$

不难得到：

- $a_1 = \cdots = a_{p-1}$。
- $a_0 - a_1 = \sum_{E \subset E_p, c(E)=1} (-1)^{|E|} V(E, 0) = kp \sum_{E \subset E_p, c(E)=1} (-1)^{|E|}$。
- $a_0 + \cdots + a_{p-1} = \binom{kp}{p} \times p!$。

如果能计算出

$$f_p = \sum_{E \subset E_p, c(E)=1} (-1)^{|E|}$$

的值，我们就可以解出 a_0 了。

2.4 计算 f

我们用反面考虑的方法来计算各项 f_n。假设 E 使得 $c(E) > 1$，即图不连通。我们枚举 1 所在的那个连通块的大小，即可完成计算。即：

观察 设 $g_n = \sum_{E \subset E_n} (-1)^{|E|}$。那么

$$f_n = g_n - \sum_{i=1}^{n-1} \binom{n-1}{i-1} f_i g_{n-i}$$

证明 枚举的 i 即为 1 所在连通块大小。需要从剩下 $n-1$ 个点中选 $i-1$ 个和 1 放进同一连通块，方法数为 $\binom{n-1}{i-1}$；接着图被分为两部分，一部分是强制连通的，其总和为 f_i；另一部分可以任意连边，总和为 g_{n-i}。即得上式。□

注意到 g 大部分值都为 0。准确来说，$g_1 = 1$，$g_i = 0\ (i \geq 2)$。这样我们可以解出：

$$f_n = 0 - (n-1) g_1 f_{n-1} = -(n-1) f_{n-1}$$

由上即可得：

$$f_n = (-1)^{n-1} (n-1)!$$

这样我们终于求得了原问题的答案。因为 p 是奇数，故 $(-1)^{p-1} = 1$。可以计算得到 $a_0 = \dfrac{p! \binom{kp}{p} + kp(p-1)(p-1)!}{p}$。代回即可得到原问题的答案 $= \dfrac{a_0}{p!} = \dfrac{\binom{kp}{p} + k(p-1)}{p}$。

从上面的做法容易推出：对任意整数 c，若 c 不是 p 的倍数，则 $\{1, 2, ..., kp\}$ 的 c 元子集中，模 p 余每种余数的集合个数都相等。证明留给读者。我们将在之后再详细分析这个问题。

2.5 与其他做法的对比

本题有一个简单的做法：考虑把全部 pk 个数排成一个 p 行 k 列的数表，第 $i(1 \leq i \leq p)$ 行第 $j(1 \leq j \leq k)$ 列填的数为 $(j-1)k + i$。对每一个 p 元子集，如果它的的元素不全在同一列，那么我们找到最小的出现了至少一个元素的列，并给这一列每个元素顺次向下挪 1 位，挪 2 位，……，挪 $(p-1)$ 位（每次如果某个元素在最下面一行，挪动之后就变成这一列最上面一行的元素）。容易证明得到的 p 个集合的元素和恰好遍历模 p 的完系。故我们可以将这些集合分成 p 个一组，每组中元素和模 p 余 $0, 1, 2, ..., (p-1)$ 各一个。剩下还有 k 个集合没有考虑到，这些集合全部元素都在同一列。由于 p 是奇数，这 k 个集合的元素和都是 p 的倍数。这样我们就推得了和上面类似的结论，可以计算答案了。

这个做法的优点是简单明了、计算量很少，缺点是可扩展性不足。接下来我们将研究更多的问题，从而充分展现针对互异关系的容斥这一思路的丰富应用。

3 更多应用

3.1 关于异或

问题 给定正整数 n,k。集合 $\{0,1,...,2^k-1\}$ 有多少 n 元子集异或和为 0？

我们沿用 2.2 ~ 2.4 中的思路。令 $m=2^k$，同样是考虑加入顺序，将问题变为 m 个，并容斥一个边集表示强制相等的元素对。令 $V(E,t)$ 表示容斥的边集为 E 时，各元素异或和为 t 的方案数。可知

$$V(E,t)=\begin{cases} m^{c(E)}, & E \text{ 中全部连通块大小均为偶数}, t=0 \\ 0, & E \text{ 中全部连通块大小均为偶数}, t>0 \\ m^{c(E)-1}, & E \text{ 中至少一个连通块大小为奇数} \end{cases} \tag{3}$$

设 a_t 表示问题 t 的答案。可得：

- $a_1=\cdots=a_{m-1}$。
- $a_0+\cdots+a_{m-1}=\binom{m}{n}\times n!$。

我们只需要计算 a_0-a_1 即可。注意到

$$a_0-a_1=\sum_{E\subset E_n,E \text{ 中全部连通块大小均为偶数}}(-1)^{|E|}m^{c(E)}$$

不妨记上述结果为 h_n。我们来推导计算 h_n 的方法。同样，我们来枚举 1 所在的连通块大小，可得：

$$h_n=\sum_{2\le i\le n,2|i} m\binom{n-1}{i-1}f_i h_{n-i}$$

其中 f_i 表示大小为 i 的连通图中，偶数边数的图的数量减去奇数边数的图的数量。由 2.4 中的结果知 $f_i=(-1)^{i-1}(i-1)!$。由于 i 都是偶数，可以将 $f_i=-(i-1)!$ 代入得：

$$h_n=\sum_{2\le i\le n,2|i}-m(i-1)!\binom{n-1}{i-1}h_{n-i}=\sum_{2\le i\le n,2|i}-m(i-1)!\binom{n-1}{i-1}h_{n-i}=(n-1)!\sum_{2\le i\le n,2|i}-m\frac{h_{n-i}}{(n-i)!}$$

令 $h(x)=\sum_{i=0}\frac{h_i}{i!}x_i$ 为 h 的指数型生成函数。可得：

$$h'=-m(x+x^3+x^5+\cdots)h=-\frac{m}{2}(\frac{1}{1-x}-\frac{1}{1+x})h$$

可以解得：

$$h(x)=\mathrm{e}^{-\frac{m}{2}(-\ln(1-x)-\ln(1+x))+C}=\mathrm{e}^{C}\times(1-x^2)^{\frac{m}{2}}$$

由于 $h_0=1$，可以得到 $C=0$。故

$$h(x)=(1-x^2)^{\frac{m}{2}}$$

故

$$h_n=\begin{cases}(-1)^{n/2}\dbinom{m/2}{n/2}, & n\equiv 0 \pmod 2\\ 0, & n\equiv 1 \pmod 2\end{cases} \tag{4}$$

代回即可求得答案。可以发现，用传统的做法很难求得答案。

3.2 对第 2 节结论的拓展

问题 给定奇质数 p 和正整数 k,n，试计算 $\{1,2,...,kp\}$ 有多少个 n 元子集的各元素之和是 p 的倍数？□

用 2.5 中提到的画数表的方法，可以简单得到：当 $p\nmid n$ 时，元素和落入各剩余类的数量相等；$p\mid n$ 时，落入剩余类 0 的比其他类要多 $\binom{k}{n/p}$。下面我们来利用容斥的方法验证这一结论。

大部分定义同第 2 节。方便起见，令 $m=kp$。由于 n 可能大于 p，我们需要补充 $V(E,t)$ 的定义：

$$V(E,t)=\begin{cases}m^{c(E)}, & E\text{ 中全部连通块大小均为 }p\text{ 的倍数},t=0\\ 0, & E\text{ 中全部连通块大小均为 }p\text{ 的倍数},t>0\\ m^{c(E)-1}, & E\text{ 中至少一个连通块大小不为 }p\text{ 的倍数}\end{cases} \tag{5}$$

仿照 3.1 节中的思路，我们来计算 $h_n=a_0-a_1$。

$$h_n=\sum_{p\le i\le n,p\mid i}(-1)^{i-1}m(i-1)!\binom{n-1}{i-1}h_{n-i}=\sum_{p\le i\le n,p\mid i}(-1)^{i-1}m(i-1)!\binom{n-1}{i-1}h_{n-i}=(n-1)!\sum_{p\le i\le n,p\mid i}(-1)^{i-1}m\frac{h_{n-i}}{(n-i)!}$$

同样令 $h(x)=\sum_{i=0}\dfrac{h_i}{i!}x_i$ 为 h 的指数型生成函数。可得：

$$h'=m(x^{p-1}-x^{2p-1}+x^{3p-1}-x^{4p-1}+\cdots)h=m\frac{x^{p-1}}{1+x^p}h$$

可以解得

$$h = \mathrm{e}^{\frac{m}{p}\ln(1+x^p)+C} = \mathrm{e}^C \times (1+x^p)^{m/p}$$

显见 $C = 0$。故可得

$$h_n = \begin{cases} \binom{m/p}{n/p} n!, & p \mid n \\ 0, & p \nmid n \end{cases} \tag{6}$$

这与我们已经得到的结果是一致的。

4 结语

通过上述几道题目，我们大致介绍了“对互异关系的容斥”这一思路的应用。然而本文内容有限，只能介绍几个简单的例子，希望本文能起到抛砖引玉的效果，各位同僚能在这一问题上继续探索！

5 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队杨耀良教练的指导。

感谢梁则贤老师、何泽霖老师、黄秉刚老师对我的培养与教导。

感谢父母对我的理解与支持。

《魔术师》命题报告

浙江省诸暨中学 孟煜皓

摘 要

《魔术师》是笔者为 IOI2023 中国国家集训队队员互测命制的一道试题，本题要求选手仅使用固定长度区间的翻转操作对排列进行排序，同时要求操作次数尽可能地少。这是一道构造题，本文介绍了有解的充要条件，同时给出了若干种构造方法，并分析了每种方法的操作次数。构造题是近年来信息学竞赛中一类常见的问题，本文通过介绍《魔术师》的命制过程和解法，总结了解决这类构造题的一种思考方向。

1 试题

1.1 题目描述

King 是一名魔术师，他总是能运用一些简单的原理完成一些不可思议的事情。

现在 King 手里有一叠背面向上的扑克牌，一共有 n 张。King 为每张牌都赋予了一个在 1 到 n 之间的独一无二的编号，初始时自顶向下第 i 张牌的编号为 p_i。在变魔术时，King 会执行若干次操作，使得自顶向下的第 i 张牌编号恰好为 i。

为了使魔术效果更具欺骗性，King 只会做一些看上去平常的动作。我们略去魔术师的具体手法，King 一次操作可以简化为以下步骤：

1. 从牌叠顶部拿起一叠牌放到桌面上，称为牌叠 A（可以为空）；
2. 从牌叠顶部一张一张背面朝上往桌面上发 m 张牌，得到牌叠 B；
3. 将牌叠 B 放回牌堆顶部；
4. 将牌叠 A 放回牌堆顶部。

请你帮助 King 给出一种操作方案，或告诉 King 不存在这样的操作方案。

1.2 输入格式

第一行三个整数 T, n, m，表示测试包编号、牌堆中牌的数量和每次操作发牌的数量。

第二行 n 个整数 $p_1, p_2, \ldots, p_n$，表示自顶向下每张牌的编号。

1.3 输出格式

若不存在操作方案，输出 -1。否则：

第一行一个非负整数 k，表示操作次数。

接下来 k 行，第 i 行一个非负整数 c_i $(0 \le c_i \le n - m)$，表示第 i 次操作的牌叠 A 中牌的数量。

1.4 样例 1

1.4.1 样例输入

```
0 8 4
6 2 5 1 7 4 3 8
```

1.4.2 样例输出

```
4
0
2
3
1
```

1.5 样例 2

1.5.1 样例输入

```
0 6 5
3 4 1 5 6 2
```

1.5.2 样例输出

```
-1
```

1.6 评分方式

每个测试包附有参数 $lim_1, lim_2, \ldots, lim_5$。设一个测试包的分值为 S，对于该测试包内的测试点，评分方式如下：

- 若该测试点无解：
  - 若选手输出有解，得 0 分。
  - 否则得 S 分。
- 若该测试点有解：
  - 若选手输出无解或选手的方案不合法，得 0 分。
  - 否则设选手的方案操作次数为 k，则得分为

$$\frac{S}{5} \times \sum_{i=1}^{5} [k \le lim_i]$$

该测试包的得分为测试包内每个测试点得分的最小值。

1.7 数据范围

对于 100% 的数据，$1 \le m \le n \le 1000$，p 是一个 1 到 n 的排列。
对于有解的数据，数据生成方式为：确定 n, m 后，p 在有解的排列集合中随机生成。
各测试包的附加限制如表 1 所示。
对于编号为奇数的测试包，保证 m 为奇数；对于编号为偶数的测试包，保证 m 为偶数。

1.8 时空限制

时间限制：1s
空间限制：512MB
特别地，对于测试包 $1, 2, 11, 12$，时间限制为 2s。

表 1: 各测试包的附加限制

测试包编号	$n \le$	$m \in$	$lim =$	分值
1	10	$[1, n]$	{50, 200, 500, 2000, 10000}	5
2			{120, 300, 1000, 3000, 10000}	
3	30		{1000, 30000, 60000, 300000, 1000000}	
4			{2500, 50000, 110000, 300000, 1000000}	
5	50		{3000, 25000, 150000, 500000, 1000000}	15
6			{6000, 50000, 300000, 700000, 1500000}	5
7	200		{20000, 50000, 200000, 500000, 1000000}	
8			{40000, 100000, 300000, 700000, 1500000}	
9	1000	$[4, 5]$	{70000, 150000, 300000, 700000, 1500000}	
10			{100000, 200000, 400000, 800000, 1500000}	
11		$[6, 8]$	{50000, 100000, 200000, 500000, 1000000}	
12			{60000, 120000, 250000, 500000, 1000000}	
13		$[40, 60]$	{10000, 25000, 50000, 200000, 1000000}	
14			{12000, 30000, 60000, 200000, 1000000}	
15		$[1, n]$	{450000, 700000, 950000, 1250000, 1500000}	15
16			{1000000, 1200000, 1600000, 2200000, 2500000}	5

2 记号与约定

下文中，我们将用一些符号来表示操作：

- $\text{rev}_m(l)$ 表示翻转以 l 为左端点，长度为 m 的区间。
- $\text{lshift}_{m,k}(l)$ 表示将以 l 为左端点，长度为 m 的区间循环左移 k 格。
- $\text{rshift}_{m,k}(l)$ 表示将以 l 为左端点，长度为 m 的区间循环右移 k 格。
- $\text{shift3}(x_0, x_1, x_2)$ 表示同时对 $0 \le i \le 2$，将 $x_{(i+1) \bmod 3}$ 位置的值改为 x_i 位置的值。

我们用记号 $A \Longrightarrow B$ 表示可以用集合 A 中的操作构造出集合 B 中的操作，$A \Longleftrightarrow B$ 表示 $A \Longrightarrow B$ 且 $B \Longrightarrow A$。

我们用置换的形式表示排列。对于一个排列 $\pi = \{p_1, p_2, \ldots, p_n\}$，有以下记号：

- $\pi(i) = p_i$，表示 π 中第 i 项的值。
- π^{-1} 表示 π 的逆，即 π^{-1} 满足 $\pi^{-1}(\pi(i)) = i$。
- $\pi \circ g$ 表示 π 执行操作 g 后得到的排列。

3 解题思路

3.1 题意简化

题目本质上是给定了一个排列 $\pi = \{p_1, p_2, \ldots, p_n\}$ 和操作区间长度 m，要求使用尽量少的 rev_m 操作对该排列排序。

3.2 $n = m$ 的情况

当 $n = m$ 时，合法的排列只有 $\{1, 2, \ldots, n\}$ 和 $\{n, n-1, \ldots, 1\}$ 两种。前者不需要操作，对于后者，执行 $\mathrm{rev}_m(1)$ 即可。

3.3 $n = m + 1$ 的情况

引理 3.3.1. $\mathrm{rev}_m \Longrightarrow \mathrm{lshift}_{m+1,2} \Longrightarrow \mathrm{lshift}_{m+1,2k}, \mathrm{rev}_m \Longrightarrow \mathrm{rshift}_{m+1,2} \Longrightarrow \mathrm{rshift}_{m+1,2k}$

证明. 通过如下两次 rev_m 操作即可实现 $\mathrm{lshift}_{m+1,2}$ 操作：

$$\begin{aligned} & \{\ldots, p_l, p_{l+1}, p_{l+2}, \ldots, p_{l+m}, \ldots\} \\ \xrightarrow{\mathrm{rev}_m(l+1)} & \{\ldots, p_l, p_{l+m}, \ldots, p_{l+2}, p_{l+1}, \ldots\} \\ \xrightarrow{\mathrm{rev}_m(l)} & \{\ldots, p_{l+2}, \ldots, p_{l+m}, p_l, p_{l+1}, \ldots\} \end{aligned}$$

类似地，先执行 $\mathrm{rev}_m(l)$ 再执行 $\mathrm{rev}_m(l+1)$ 即可实现 $\mathrm{rshift}_{m+1,2}$。 □

引理 3.3.2. $\mathrm{lshift}_{m,2k} \Longleftrightarrow \mathrm{rshift}_{m,2k}$

证明. 当 m 为偶数时，$\mathrm{lshift}_{m,2k} = \mathrm{rshift}_{m,m-2k} \Longleftrightarrow \mathrm{rshift}_{m,2k}$。

当 m 为奇数时，$\mathrm{lshift}_{m,2k} \Longleftrightarrow \mathrm{lshift}_{m,k} \Longleftrightarrow \mathrm{rshift}_{m,2k}$。 □

连续两次对同一位置的 rev_m 操作没有意义，所以当 $n = m + 1$ 时，根据以上引理，最终排列一定是原排列进行一次 $\mathrm{lshift}_{m+1,2k}$ 操作，然后进行零次或一次 rev_m 操作得到的。枚举所有情况判断即可。

3.4 $n \geq m + 2$ 且 m 为偶数的情况

引理 3.4.1. 当 m 为偶数时，$\mathrm{lshift}_{m+1,2k} \Longleftrightarrow \mathrm{lshift}_{m+1,k}$。

3.4.1 还原 $m+1$ 到 n

考虑从大到小将每个数归位。假设现在 $i+1$ 到 n 已经归位，即 $\forall j > i, \pi(j) = j$。令 $x = \pi^{-1}(i)$，注意到有 $(\pi \circ \mathrm{rev}_m(x))^{-1}(i) = x + m - 1$。考虑不断执行以下过程：

1. 令 $x = \pi^{-1}(i)$。
2. 若 $x + m - 1 \le i$，执行 $\mathrm{rev}_m(x)$，回到第 1 步。
3. 否则执行 $\mathrm{rshift}_{m,i-x}(x)$，完成归位。

该算法可以将 $m+1$ 到 n 的所有数归位，期望需要的 rev_m 操作次数约为

$$\frac{n^2}{4(m-1)} + \frac{(n-m)m}{2}$$

事实上，当 $i > 2m$ 时，我们可以将上述过程改为：

1. 令 $x = \pi^{-1}(i)$。
2. 若 $x + m - 1 \le i$，执行 $\mathrm{rev}_m(x)$，回到第 1 步。
3. 若 x 与 i 奇偶性不同，执行 $\mathrm{rev}_m(i - m + 1)$，回到第 1 步。
4. 否则：
   (a) 令 $l = i - m + 1 - (i - x)/2$。
   (b) 若 $l > 0$，执行 $\mathrm{rev}_m(l)$，然后执行 $\mathrm{rev}_m(i - m + 1)$，完成归位。
   (c) 否则执行 $\mathrm{rshift}_{m,i-x}(x)$，完成归位。

使用该算法可以将 rev_m 操作的期望次数降为

$$\frac{n^2}{4(m-1)} + \frac{m \cdot \min(n-m, m)}{2}$$

3.4.2 shift3 操作

引理 3.4.2. $\{\mathrm{lshift}_{m+1,2}, \mathrm{rshift}_{m+1,2}\} \Longrightarrow \mathrm{shift3}(1, m, m+2)$。

证明. 通过如下两次操作即可实现 $\mathrm{shift3}(1, m, m+2)$：

$$\begin{aligned} & \{p_1, p_2, \ldots, p_{m-1}, p_m, p_{m+1}, p_{m+2}, \ldots\} \\ \xrightarrow{\mathrm{rshift}_{m+1,2}(2)} & \{p_1, p_{m+1}, p_{m+2}, p_2, \ldots, p_{m-1}, p_m, \ldots\} \\ \xrightarrow{\mathrm{lshift}_{m+1,2}(1)} & \{p_{m+2}, p_2, \ldots, p_{m-1}, p_1, p_{m+1}, p_m, \ldots\} \end{aligned}$$

□

引理 3.4.3. $\text{lshift}_{m+1,k} \Longrightarrow \{\text{shift3}(x_0, x_1, x_2) \mid 1 \le x_0, x_1, x_2 \le m+2 \wedge x_0 \ne x_1 \wedge x_0 \ne x_2 \wedge x_1 \ne x_2\}$。

证明. 令 $a = \pi(x_0), b = \pi(x_1), c = \pi(x_2)$。

考虑使用若干次 lshift 操作使得 $\pi(1) = a, \pi(m) = b, \pi(m+2) = c$，过程如下：

1. 若 $x_0 = m+2$，使用一次 lshift 操作进行微调。
2. 执行 $\text{lshift}_{m+1,\pi^{-1}(a)-1}(1)$。
3. 执行 $\text{lshift}_{m+1,\pi^{-1}(c)-2}(2)$。
4. 此时 $\pi(1) = a, \pi(2) = c$，对 $\pi^{-1}(b)$ 分类讨论：
   - 若此时 $\pi^{-1}(b) = m+1$，只需要执行 $\text{lshift}_{m+1,1}(2)$ 即可。
   - 若此时 $\pi^{-1}(b) = m+2$，依次执行以下操作：

$$\begin{aligned} & \{a, c, \ldots, \#, \#, b\} \\ \xrightarrow{\text{rshift}_{m+1,2}(1)} & \{\#, \#, a, c, \ldots, b\} \\ \xrightarrow{\text{lshift}_{m+1,3}(2)} & \{\#, \ldots, b, \#, a, c\} \\ \xrightarrow{\text{rshift}_{m+1,1}(1)} & \{a, \#, \ldots, b, \#, c\} \end{aligned}$$

   - 否则，执行 $\text{lshift}_{m+1,\pi^{-1}(b)}(1)$ 使得 $\pi(m+1) = b$。此时 a, c 一定仍然相邻且 $\pi^{-1}(a) \ge 2$。接下来依次执行 $\text{rshift}_{m+1,1}(2), \text{lshift}_{m+1,\pi^{-1}(a)-1}(1)$，转化成 $\pi^{-1}(b) = m+2$ 的情况。

接下来执行 $\text{shift3}(1, m, m+2)$ 操作。

最后，将上述过程除了最后的 shift3 操作倒过来，还原即可。

当 $m = 2$ 时，上述证明方法不适用，但仍可以通过简单地构造证明，也可以通过计算机验证该命题的正确性。 □

直接使用上述证明中的过程实现 shift3 操作，期望需要大约 $10m$ 次 rev_m 操作。

3.4.3 基于 shift3 操作的算法

引理 3.4.4. 当 $m \bmod 4 = 0$ 且 $n \ge m+2$ 时，排列可以通过 rev_m 操作排序当且仅当排列的逆序对数为偶数。

当 $m \bmod 4 = 2$ 且 $n \ge m+2$ 时，排列一定可以通过 rev_m 操作排序。

证明. 当 $m \bmod 4 = 0$ 时，rev_m 操作相当于偶数次交换操作，由于一次交换操作必然改变逆序对数奇偶性，所以此时 rev_m 操作不会改变逆序对数的奇偶性。排序后的排列逆序对数为 0，所以初始排列的逆序对数必须是偶数。必要性得证。

首先使用 3.4.1 中的方法将 $m+3$ 到 n 归位。然后使用 shift3 操作，依次将 1 到 m 归位，最后剩下 $m+1, m+2$。此时逆序对不大于 1。又由于初始排列逆序对为偶数，且 shift3 操作不会改变逆序对数，所以最终逆序对数为 0，即排列有序。充分性得证。

当 $m \bmod 4 = 2$ 时，rev_m 操作相当于奇数次交换操作，所以此时 rev_m 操作一定会改变逆序对数的奇偶性。

仍然首先将 $m+3$ 到 n 归位。此时若排列逆序对数为奇数，任意进行一次 rev_m 操作改变排列逆序对奇偶性。然后使用 shift3 操作，依次将 1 到 m 归位，最后剩下 $m+1, m+2$。同样地，此时排列一定有序。 □

使用上述证明中的方法，可以得到一个期望次数不超过

$$\frac{n^2}{4(m-1)} + \frac{m \cdot \min(n-m, m)}{2} + 10m^2$$

的算法（忽略一次项）。结合朴素的广度优先搜索算法，在 m 为偶数的 40 分中，可以得到 27 分。

3.4.4 对基于 shift3 操作的算法的改进

注意到，在实现 shift3 操作的过程中，主要的操作开销在于，要在已经将两个数放到边上的同时，将第三个数移到指定位置。

事实上，在上一小节介绍的算法中，每次使用 shift3 算法还原一个数时，我们只关心两个位置。

引理 3.4.5. $\text{lshift}_{m+1,k} \Longrightarrow \text{shift3}(1, m+1, m+2)$。

证明. 通过如下两次操作即可实现 $\text{shift3}(1, m+1, m+2)$：

$$\begin{aligned} & \{p_1, p_2, \ldots, p_m, p_{m+1}, p_{m+2}, \ldots\} \\ \xrightarrow{\text{rshift}_{m+1,1}(2)} & \{p_1, p_{m+2}, p_2, \ldots, p_m, p_{m+1}, \ldots\} \\ \xrightarrow{\text{lshift}_{m+1,1}(1)} & \{p_{m+2}, p_2, \ldots, p_m, p_1, p_{m+1}, \ldots\} \end{aligned}$$

□

类似地，也可以简单证明 $\text{lshift}_{m+1,k} \Longrightarrow \text{shift3}(1, 2, m+2)$。

考虑依次归位 1 到 m 的过程，假设当前 1 到 $i-1$ 已经归位，需要归位的数是 i。设 $\pi(i) = a$。考虑依次执行 $\text{lshift}_{m+1,i-1}(1), \text{lshift}_{m+1,\pi^{-1}(i)-1}(2)$ 使得 $\pi(1) = a, \pi(m+2) = i$。此时 1 到 $i-1$ 一定仍然是连续的一段。

因此，执行完上述操作后，$\pi(2)$ 和 $\pi(m+1)$ 至少有一个不是已归位的数，所以 $\text{shift3}(1, 2, m+2)$ 和 $\text{shift3}(1, m+1, m+2)$ 中一定有一个操作不会影响已归位的数，选用该操作即可。

事实上，类似引理 3.4.2，我们也可以简单证明 $\{\text{lshift}_{m+1,2}, \text{rshift}_{m+1,2}\} \Longrightarrow \text{shift3}(1, 3, m+2)$。

当 $i \le m-2$ 时，执行完两次 lshift 操作后，$\pi(3)$ 和 $\pi(m)$ 至少有一个不是已归位的数，所以 shift3$(1,3,m+2)$ 和 shift3$(1,m,m+2)$ 中一定有一个操作不会影响已归位的数。shift3$(1,3,m+2)$ 和 shift3$(1,m,m+2)$ 相比 shift3$(1,2,m+2)$ 和 shift3$(1,m+1,m+2)$，前者分别只需要 4 次 rev_m 操作，后者分别需要 $2m$ 次。此时选用前者更优。

最后，与原算法相同，将前面的两次 lshift 操作还原即可。

该算法在最坏情况下的操作次数约为 $3m^2$，期望操作次数约为 $2m^2$。结合朴素的广度优先搜索算法，在 m 为偶数的 40 分中，可以得到 34 分。

3.4.5 更优秀的算法

上述算法的主要开销在于最后的还原，即将已归位的数放回原来的位置。事实上，我们只要保证已归位的数始终是连续的一段即可。

假设已归位的数在序列的末尾，考虑将 i 放到末尾，且不改变已归位的数的相对顺序。我们对 $\pi^{-1}(i)$ 分类讨论：

- 若 $\pi^{-1}(i) = 1$，依次执行 $\text{lshift}_{m+1,1}(2), \text{lshift}_{m+1,1}(1)$，此时 i 已经接在已归位的数后面，再执行 $\text{rshift}_{m+1,1}(2)$ 即可将这些数移到序列末尾。
- 若 $\pi^{-1}(i) > 1$，执行 $\text{lshift}_{m+1,\pi^{-1}(i)-1}(2)$，使得 $\pi(m+2) = i$。此时已归位的数仍然连续，执行 $\text{lshift}_{m+1,\pi^{-1}(i-1)}(1)$，将这些数移到 i 的前面即可。

事实上，对于 $\pi^{-1}(i) = 1$ 的情况，若未归位的数超过 3 个，后面两次操作可以均改为移动两格，可以将 rev_m 的次数从 $3m$ 降至 $m+4$。

我们仍然在执行该算法前保证排列逆序对数为偶数。然后我们只要把 3 到 $m+2$ 的数依次使用上述算法归位，由于保证了逆序对奇偶性，剩下的 1 和 2 也一定已经归位。

3.4.6 操作次数分析

上述算法中，每次将 i 放到末尾需要的操作次数为：

- 若 $\pi^{-1}(i) = 1$，此时若 $i < m+2$，则操作次数为 $m+4$，否则操作次数为 $3m$。
- 若 $\pi^{-1}(i) > 1$ 且为奇数，则操作次数为 $m+2$。
- 若 $\pi^{-1}(i)$ 为偶数，则操作次数为 m。

在随机数据下，第一种情况的期望出现次数约为 $\ln m$，因此当 $i < m+2$ 时出现第一种情况对操作次数的影响可以忽略不计。当 $i = m+2$ 时有 $\frac{1}{3}$ 的概率出现第一种情况，此时操作次数需要额外加 $2m$。

对于后两种情况，我们可以将这两种情况出现的概率分别视为 $\frac{1}{2}$。参考 [1] 和 [2]，使用在线计算器[1] 计算得到，当 $m = 998$ 时，这一部分的操作次数超过 $998\times998+550\times2 = 997104$ 的概率约为 6×10^{-4}，超过 $998 \times 998 + 600 \times 2 = 997204$ 的概率约为 6×10^{-11}。

该算法可以通过所有 m 为偶数的测试包，期望得分 40 分。

3.5 $n \geq m + 2$ 且 m 为奇数的情况

m 为奇数时，一个数所在位置的奇偶性始终不会改变。由于最终要达到 $\pi(i) = i$，因此初始排列满足对于任意 i，$\pi(i)$ 与 i 的奇偶性相同，是有解的必要条件。在本小节中，我们仅考虑满足该条件的排列。

接下来我们将用如下形式表示一个排列：

$$\begin{matrix} p_1 & p_3 & p_5 & \cdots \\ p_2 & p_4 & p_6 & \cdots \end{matrix}$$

一次 rev_m 操作即为对一个等腰梯形区域内的上下两部分分别翻转：

$$\begin{matrix} p_1 & p_3 & p_5 & \cdots \\ p_2 & p_4 & p_6 & \cdots \end{matrix} \qquad \begin{matrix} p_1 & p_3 & p_5 & \cdots \\ p_2 & p_4 & p_6 & \cdots \end{matrix}$$

一次 $\mathrm{lshift}_{m+1,2k}$ 或 $\mathrm{rshift}_{m+1,2k}$ 操作即为对一个平行四边形区域内的上下两部分分别循环左移或右移 k 格：

$$\begin{matrix} p_1 & p_3 & p_5 & p_7 & \cdots \\ p_2 & p_4 & p_6 & \cdots \end{matrix} \qquad \begin{matrix} p_1 & p_3 & p_5 & p_7 & \cdots \\ p_2 & p_4 & p_6 & \cdots \end{matrix}$$

3.5.1 $n = m + 2$ 的情况

引理 3.5.1. 当 $n = m + 2, m > 3$ 且 m 为奇数时，若进行任意次 rev_m 操作后下半部分保持不变，则上半部分的逆序对数量奇偶性也保持不变。

证明. 选取下半部分中任意两个相邻的数 a, b（第一个数和最后一个数也视为相邻）。令 $f(a, b)$ 表示 a 是否在 b 前面（若 a, b 分别是第一个数和最后一个数，则视为 b 在 a 的前面）。

执行一次 rev_m 操作后，下半部分操作前相邻的数仍然相邻。同时，执行一次 rev_m 操作一定会改变 $f(a, b)$ 的值。

因此，为了使下半部分保持不变，执行的 rev_m 操作必然为偶数次。

若 $m \bmod 8 = 1$，一次 rev_m 操作不会改变上下两部分的逆序对数奇偶性，命题成立。

[1] https://keisan.casio.com/exec/system/1180573396

若 $m \bmod 8 = 5$，一次 rev_m 操作会同时改变上下两部分的逆序对数奇偶性，因为操作次数为偶数，所以命题成立。

若 $m \bmod 4 = 3$，一次 rev_m 操作恰好改变上下两部分中某一部分的逆序对数奇偶性，因为总次数为偶数，其中改变下半部分逆序对数奇偶性的操作次数为偶数，所以改变上半部分逆序对数奇偶性的操作次数也为偶数，命题成立。 □

根据该引理，可以得到当 $n = m + 2, m > 3$ 且 m 为奇数时，有解的充要条件：

- 对于任意 $1 \le i \le \frac{n-1}{2}$，$2i$ 和 $2(i \bmod \frac{n-1}{2}) + 2$ 在下半部分中相邻。
- 使用任意次操作还原下半部分后，上半部分的逆序对数为偶数。

下半部分的还原与 3.3 类似。

引理 3.4.2 仍然适用，因此我们仍然可以采用 3.4.4 中的算法，在保持下半部分不变的前提下还原上半部分。

3.5.2 还原 $m + 1$ 到 n

尽管当 m 是奇数时，lshift 和 rshift 操作中位移的长度必须为偶数，但是，由于有每个位置上的数必须与该位置的奇偶性相同这一限制，我们仍然可以使用 3.4.1 中的方法还原 $m+1$ 到 n。

3.5.3 基于 shift3 操作的算法

引理 3.5.2. 当 $m \bmod 8 = 1$ 且 $n \ge m + 3$ 时，排列可以通过 rev_m 操作排序当且仅当上下两部分的逆序对数奇偶性均为偶数。

当 $m \bmod 8 = 5$ 且 $n \ge m + 3$ 时，排列可以通过 rev_m 操作排序当且仅当上下两部分的逆序对数奇偶性均为偶数或均为奇数。

当 $m \bmod 4 = 3$ 时，排列一定可以通过 rev_m 操作排序。

证明. 当 $m \bmod 8 = 1$ 时，一次 rev_m 操作不会改变上下两部分的逆序对数奇偶性。必要性得证。

首先将 $m+4$ 到 n 归位。由于保证了上下两部分的逆序对数均为偶数，所以上下两部分均可使用 3.4.4 中的算法还原。充分性得证。

当 $m \bmod 8 = 5$ 时，一次 rev_m 操作一定会同时改变上下两部分的逆序对数奇偶性。必要性得证。

仍然首先将 $m + 4$ 到 n 归位。此时若上下两部分逆序对数均为奇数，则任意进行一次 rev_m 操作使得逆序对数为偶数。然后分别还原上下两部分即可。充分性得证。

当 $m \bmod 4 = 3$ 时，一次 rev_m 操作恰好改变上下两部分中某一部分的逆序对数奇偶性。

将 $m+4$ 到 n 归位后，可以进行至多两次 rev_m 操作使得上下两部分逆序对数均为偶数。同样地，上下两部分分别还原即可。 □

直接使用上述证明中的方法，即可得到一个基于 shift3 操作的算法。

事实上，我们可以采用 3.4.5 中的方法先还原下半部分，再使用 3.4.4 中的算法，在保持下半部分不变的前提下还原上半部分。这样做可以使操作次数更少。

3.5.4 建立绑定关系

上一小节中介绍的算法，瓶颈在于我们要在保持其中一部分不变的前提下还原另一部分。不妨反过来考虑，我们能不能设计一个算法，事先安排好两部分的顺序，使得还原一部分后，另一部分恰好也能还原？

假设 $n=m+2$。考虑对每个 $1\le i\le \frac{m+1}{2}$，将 $2i-1$ 和 $2i$ 绑定，即使得 $2i-1$ 和 $2i$ 始终相邻。

我们将已经绑定的数放在末尾（不包括位置 $m+2$）：

$$\begin{array}{cccccccc} p_1 & p_3 & \cdots & p_{m-2} & p_m & p_{m+2} \\ & & & \searrow & \searrow & \\ p_2 & p_4 & \cdots & p_{m-1} & p_{m+1} & \end{array}$$

考虑将 p_2-1 与 p_2 绑定。令 $a=p_2-1$，根据 a 所在的位置分类讨论：

- $\pi^{-1}(a)=m+2$。执行 $\mathrm{lshift}_{m+1,2}(2)$ 即可。

$$\begin{array}{cccccccc} p_1 & p_3 & \cdots & p_{m-2} & p_m & a & & \\ & & & \searrow & \searrow & & & \\ p_2 & p_4 & \cdots & p_{m-1} & p_{m+1} & & & \end{array} \Longrightarrow \begin{array}{cccccccc} p_1 & \cdots & p_{m-2} & p_m & a & p_3 & & \\ & & \searrow & \searrow & & \searrow & & \\ p_4 & \cdots & p_{m-1} & p_{m+1} & p_2 & & & \end{array}$$

- 否则，执行 $\mathrm{lshift}_{m+1,\pi^{-1}(a)-1}(1)$ 使得 $\pi(1)=a$。

$$\begin{array}{cccccccc} p_1 & \cdots & a & \cdots & p_{m-2} & p_m & p_{m+2} & \\ & & & & \searrow & \searrow & & \\ p_2 & \cdots & \cdots & \cdots & p_{m-1} & p_{m+1} & & \end{array} \Longrightarrow \begin{array}{cccccccc} a & \cdots & p_{m-2} & p_m & p_1 & \cdots & p_{m+2} & \\ & & \searrow & \searrow & & & & \\ \cdots & \cdots & p_{m-1} & p_{m+1} & p_2 & \cdots & & \end{array}$$

接下来执行 $\mathrm{lshift}_{m+1,\pi^{-1}(p_2)-2}(2)$ 使得 p_2 回到原位。

$$\begin{array}{cccccccc} a & \cdots & p_{m-2} & p_m & p_1 & \cdots & p_{m+2} & \\ & & \searrow & \searrow & & & & \\ \cdots & \cdots & p_{m-1} & p_{m+1} & p_2 & \cdots & & \end{array} \Longrightarrow \begin{array}{cccccccc} a & \cdots & p_{m+2} & \cdots & p_{m-2} & p_m & p_1 & \\ \times & & & & \searrow & \searrow & & \\ p_2 & \cdots & \cdots & \cdots & p_{m-1} & p_{m+1} & & \end{array}$$

然后执行 $\text{lshift}_{m+1,2}(1)$ 将 a 和 p_2 移到尾部即可。

当未绑定的数只剩下一对时，上述过程不适用。注意到，上述过程中，上半部分和下半部分的逆序对数奇偶性变化相同，而最终所有数都建立绑定关系后的排列上半部分和下半部分的逆序对数奇偶性相同，因此我们将初始排列调整成上半部分和下半部分的逆序对数奇偶性相同即可。这样就可以避免出现最后一对数无法绑定的情况。

当 $n = m + 3$ 时，对前 $m + 2$ 个数执行该算法，最终 $\pi(m + 2)$ 与 $\pi(m + 3)$ 恰好匹配，因此该算法同样适用于 $n = m + 3$ 的情况。

3.5.5 算法总结

在建立绑定关系后，我们可以将 $2i - 1$ 和 $2i$ 看成一个整体，使用 3.4.5 中的算法还原即可。

要注意的是，建立绑定关系的过程，与最终还原的过程，都有可能改变上下两部分的逆序对数奇偶性。但是，这两个过程中，对上下两部分逆序对数奇偶性的改变一定是同时的，且变化量仅仅与 m 的值有关。这意味着逆序对数奇偶性的变化是可以提前计算得到的，我们只需要适当地调整初始排列，使得上下两部分的逆序对数最终均为偶数即可。

事实上，若按照上文所述实现，这两个过程对逆序对数奇偶性的影响完全相同，所以只需要简单地保证初始排列的上下两部分逆序对数均为偶数即可。

算法流程如下：

1. 判断排列每个位置上的数是否与该位置奇偶性相同。
2. 特殊处理 $n = m + 2$ 的情况。
3. 根据 $m \bmod 8$ 的值和排列上下两部分的逆序对数奇偶性判断是否有解。
4. 还原 $m + 4$ 到 n。
5. 调整初始排列，使得上下两部分逆序对数均为偶数。
6. 建立绑定关系。
7. 使用 3.4.5 中的算法还原排列。

还原 $m + 4$ 到 n 的期望操作次数约为（忽略一次项，下同）

$$\frac{n^2}{4(m-1)} + \frac{m \cdot \min(n-m, m)}{4}$$

设 $z = \frac{m-1}{2}$，建立绑定关系的过程和最终还原的过程总期望操作次数约为

$$8\sum_{i=1}^{z}\frac{1}{i}\sum_{x=0}^{i-1}\min(x, z+1-x) \approx \left(\frac{3}{4}-\frac{\ln 2}{2}\right)m^2 \approx 0.403m^2$$

相较 m 为偶数的算法，该算法的操作次数波动较大，限制相对更为宽松。

你也可以在一开始对排列进行若干次随机操作后进行还原，然后重复若干次，取操作次数最少的方案输出。

4 命题过程

笔者是在首先想到这个问题后，逐步解决这个问题并进行完善，从而命制出这一试题的。

4.1 解决问题

最初思考这个问题时，笔者并没有太多好的想法。此时，笔者使用计算机，求出了当 n, m 较小时，有解的排列数量。然后笔者寻找有解的必要条件，在所有排列中排除不满足必要条件的排列，直到数量与有解的排列相等，从而猜测有解的必要条件。

接下来，从解数较少的情况开始分析。分析 $n = m + 1$ 的情况，可以得到 $\text{lshift}_{m+1,2}$ 和 $\text{rshift}_{m+1,2}$ 操作。

对于 $n \geq m + 2$ 且 $m \bmod 4 = 0$ 的情况，充要条件为排列逆序对数为偶数，所以我们可以尝试构造 shift3 操作。此时可以通过朴素的广度优先搜索算法帮助构造，尝试寻找出能够扩展到 m 较大时的 shift3 操作。

在找到一个算法后，可以通过观察算法实现、寻找瓶颈等方式，来优化算法花费的操作次数。

在 m 为偶数时有一个比较优秀的算法后，笔者开始思考 m 为奇数的情况。仍然从解数较少的 $n = m + 2$ 开始思考，套用、扩展 m 为偶数时的一些结论，从而得出 m 为奇数时的算法。

4.2 完善问题

在有了算法后，笔者对以何种形式展现这个问题，也做了许多思考。

如果出成传统题，由于最终操作次数与 n, m 的值以及排列形态有很大关系，如何赋分成了一大难题；如果出成提交答案的形式，虽然赋分问题解决了，但是由于本题存在许多特殊处理的情况，而且有许多无解的情况，可是提交答案题的测试点数量往往较少，会导致无法覆盖到所有需要处理的情况。

最终，笔者将两种方式的优势结合，采用给定精确数据范围、设置评分参数、捆绑测试的形式来出这道题。

5 总结与展望

构造题是近年来信息学竞赛中一类常见的问题，但这类问题没有固定的解题方法，涉及到的领域又十分广泛，往往难以解决。

本文介绍了笔者命制的一道构造题的解题思路及其命制过程。在解决这一类将初始状态通过某些操作变换至目标状态的构造题时，一般可以通过计算机计算有解的状态数量，或输出有解的状态，从而得出有解的充要条件。然后可以从一些简单的情况入手，并通过已有的操作构造出一些新的、更实用的操作。我们也可以通过有解的充要条件反向推出需要构造的操作。在这一过程中，仍然可以使用计算机辅助运算、构造、枚举。

希望《魔术师》与本文介绍的解题思路能让读者有所启发，在解决构造题时能够更加得心应手。同时，也希望本文能够起到抛砖引玉的作用，引发读者创造出更多新颖有趣的构造题。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢父母对我的关心与支持。

感谢浙江省诸暨中学袁荣乐老师、绍兴市第一中学陈合力老师、董烨华老师的培养与教导。

感谢陈威滔同学、楼宇辰同学在命题过程中的帮助。

感谢楼宇辰同学为本文审稿。

参考文献

[1] Wikipedia contributors. Binomial distribution — Wikipedia, the free encyclopedia, 2022. [Online; accessed 26-December-2022].

[2] Wikipedia contributors. Beta function — Wikipedia, the free encyclopedia, 2022. [Online; accessed 26-December-2022].

OI 中的几何

重庆市巴蜀中学 常瑞年

摘 要

几何相关的题目通常具有直观直觉，而 OI 中的直觉通常与组合优化有关，这是几何在 OI 中不温不火的原因，而在诸如 ACM 等比赛中，因命题均衡而总有一道质量不低的计算几何问题，作者因此认为几何可以当作一类重要的问题在 OI 中出现，而进一步的原因是因为物理空间总是可以看作连续的，而生活中的最优化总是看作几何的。

本文尝试对 OI 中出现的几何问题有浅显的归纳。文章的结构如下：第一章介绍计算几何有关的内容，扩充了三角剖分的概念以及其应用，几何反演的概念。第二章介绍与数据结构有关的几何对象。第三章介绍有限几何，作为最抽象的一部分。

1 计算几何

1.1 半平面交

这里介绍一种较为精确的实现方式以及其引出的点线对偶。

首先不妨假设所有代表半平面的直线均不与 x 轴垂直，故可假设半平面具有形式 $y \leqslant kx - b$ 以及 $y \geqslant kx - b$。

接着我们引入新的平面 $a - b$，考虑 $x - y$ 平面中的直线 $y = kx - b$ 到 $a - b$ 平面中的点的变换：$l : y = kx - b \to (k, b)$，记作 l^*。

并且我们还可以考虑 x–y 平面中的点到 a–b 平面中的直线的变换：$P : (x, y) \to b = xa - y$，记作 P^*。

我们将以上变换合称点线对偶，其具有以下性质：

- 在 $x - y$ 平面中点 (x, y) 在直线 $y = kx - b$ 上，则在 $a - b$ 平面中点 (k, b) 相应在直线 $b = xa - y$ 上
- 在 $x - y$ 平面中三点共线则对偶后三线共点
- 在 $x - y$ 平面中点 p 在直线 l 下方，则 $a - b$ 平面中 p^* 在 l^* 上方
- $P^{**} = P, l^{**} = l$

其中性质 2 可由性质 1 推出，我们称其为 incident 关系的不变性。

而性质 3 有一个直观的理解：$y \leqslant kx - b \Leftrightarrow b \leqslant xk - y$。

在解决问题中，给定点集的凸包问题看起来容易考察，并具有更高的数值稳定性，因此我们考虑半平面交问题的对偶：

考虑所有形式为 $y \leqslant kx - b$ 的直线在 $a - b$ 平面中的对偶，所求交区域的对偶变为了平面上在这些点上方的所有直线，自然由对偶点集的上凸包上所有点的上方直线所表示。

同理所有形式为 $y \geqslant kx - b$ 的直线的对偶的点集的下凸包的下方所有直线的对偶表示了原来的交集。

分别求解两个凸壳后，如何判定这些点所代表的直线的交集不为空呢，注意到交集不为空等价于有一个点在交集里等价于其对偶的直线分离了上下两个凸壳。

于是我们只需要判断两个凸壳是否相交即可，这又只需判定每个点是否在另一个凸壳中。

考虑处理垂直的直线，注意到大多数题目的输入都为整点，我们只需要将平面中每个点记为高斯整数的形式然后乘以复数 $1 + \sqrt{3}i$（这相当于将平面旋转一不易与原直线重合的角度），并记 $w = \sqrt{3}$ 写入结构体进行运算即可。

1.2 平面网格化

我们以一道例题来引入这种简单且有用的思想。

例题 1.2.1 (平面邻点对)**.** 给定平面中一点集 $S, |S| = n$，求欧氏距离下最近的一对点

考虑随机增量法，随机打乱后按顺序加入所有点，若当前已经得到的最小距离为 d，把平面按照 d 细分成网格，即点 (x, y) 将被分到第 $(\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor)$ 格中。

注意到，此时每格仅有 $O(1)$ 个点分配其中，否则答案会取得更小。

考虑加入新的一个点 (x, y)，其在格子 (a, b) 中，则考虑其引起的答案的更新，只需要考虑所有 $(a \pm 1, b \pm 1)$ 格子中的所有 $O(1)$ 个点即可，访问只需要 hash 表和链表。

若此时更新了答案，则重构网格划分，将当前的所有点重新添加进对应的 *hash* 表。

考虑时间复杂度，第 i 次更新答案的概率应为 $O(\frac{1}{i})$（考虑一张 n 个点的图，两两之间有形成排列的边权，第 i 次加入的点与之前的点连边更小的概率可以考虑将这一个点依次替换为之前的 i 个点进行平均），故期望为 $\sum_{i=1}^{n} O(\frac{1}{i} \cdot i) \cdot O(1) = O(n)$。

回顾我们的思路，对于距离这样的问题，最好的刻画其实是圆，但是圆既不好表示，也不能完整的划分整个平面，于是我们采用正方形去近似的表达我们的刻画，这样既划分了平面，又得到了某种几何性质，然后依据这样的较为连续的性质我们又能使用数据结构去维护。

在实际解题中，先套上一个网格化的思路去思考实际是非常有用的。

下面再举几道例题来彰显其效用：

例题 1.2.2 (BJWC2011 最小三角形)**.** 给定平面上一点集 $S, |S| = n$, 求其中三个点组成的周长最小的三角形

与上面的例子一样, 网格化随机增量后，仍然只需要考虑四周 $O(1)$ 个点来更新答案。

例题 1.2.3 (正方形覆盖问题)**.** 给定平面上一整点集 $S, |S| = n$, 你需要给出若干边长相等的正方形, 使得每个正方形至少盖住三个点, 且边与坐标轴平行, 并且顶点为整点, 并且所有点都至少被一个正方形盖住, 求正方形最小边长

考虑确定网格化的方式，除了随机增量确定边长，我们还可以通过二分的方式确定边长及答案。二分网格化之后，我们可以先把其中至少有三个点的正方形网格用正方形盖住，而后只需考虑未被盖住的点，考虑这个点能够形成的正方形数量，以这个点为某个角的正方形共计 4 个，其中每个都有不多于 1 个点（否则可直接覆盖这个点），那么总共可能包含的三个点的数量也就只有 $O(1)$ 种，至于找到那 4 个正方形中的点，其只需暴力检查其四周的网格中的点即可，容易证明这样做是 $O(n)$ 的，加上二分总时间复杂度是 $O(n \log n)$。

例题 1.2.4 (CF1641F.Coverring Circle)**.** 给定 n 个点组成的序列和参数 K, L, 找到最小的半径 r 使得存在一个半径为 r 的圆盖住了某个长度为 L 区间中至少 K 个点

直接二分网格化，考虑最终的 r，不妨让网格化的边长 $d = \frac{\sqrt{2}}{2}r$，这样某个方格内部就可以直接被一个圆覆盖，并且这样取只有相邻九个方格可以有贡献，那么处理完方格内部之后，任一方格按照下标处理的任意长度 L 的区间内部都只有不多于 K 个点，考察相邻九格，将其中的点按照下标进行排序后考察其中每一段长度为 L 的连续段，则其中至多有 $9K$ 个点，对这 $9K$ 个点作 r 固定的问题，这只需要枚举其中的一个点在圆的边界上然后枚举圆的角度作扫描线即可，总时间复杂度为 $O(-\log \varepsilon \cdot n \cdot K^2 \log K)$（$\varepsilon$ 是二分的精度）。

1.3 三角剖分

对三角剖分及其相关考察是 OI 中不足的，这里的三角剖分特指 Delaunay 三角剖分。

普通的三角剖分指的是一张以给定的点集为点集的平面图，并且除了无界面，每一张面都是一个三角形，为了方便起见，假设三点不共线，四点不共圆，这只需在程序的开头对所有点作扰动即可。

Delaunay 三角剖分指满足以下性质的三角剖分：

- 空圆性质，每个三角形面的外接圆都不包含剩下的任意一个点，这样的圆被称作这个点集的内圆，Delaunay 三角剖分与内圆一一对应，特别的，某条边存在于剖分中当且仅当存在以这条边为弦的不包含其他点的圆
- 空圆性质的推论，每个点到最近的点之间一定有连边

- 最小角最大，Delaunay 三角剖分所得的三角形面的最小角度是所有三角剖分中最大的，更进一步，将其中所有角度按照大小排序所得到的一个角度序列是最大的，这比较类似于最大生成树

- 最小外接圆半径，所有三角形面的外接圆半径的最大值是所有三角剖分中最小的

其中空圆性质可以视作 Delaunay 三角剖分的定义，我们按照这个定义去求解 Delaunay 三角剖分，其通常的方法有对偶法，分治法和随机增量法，其中随机增量法最为简单好记，我们选用这种方法进行引入。

首先我们先在所有点集之外加上一个大三角形面作为求解的开始，然后我们按随机顺序依次加入点维护当前点集的三角剖分，不妨设当前加入的点为 P，考虑包含它的一个三角面 f，直接将 P 与三角形面的三个顶点连接我们会得到一个新的三角剖分，只不过可能不满足空圆性质。

考虑当前三角剖分可能出现违反空圆性质的三角形 ABC，其外接圆包含了另一个点 D，并且 DBC 组成了一个三角形面，这样就构成了一个四边形，其中我们可以进行翻转操作：删去边 BC，插入边 AD，这样做之后在这个四边形内就全是空圆了，不过这样可能会出现更多的边需要翻转（具体来说是这个四边形本身的边），我们记录一个队列以处理。

考虑到这样操作之后角度序列字典序变大，于是不可能无限进行下去。

由于每次加入顶点之初需要知道点在哪个面，即动态的点定位过程，我们考虑随时维护每个面里都有哪些点（具体来说，每个面开一个 vector 存其中的点），以同时维护每个点都对应到哪张面上，并且在边翻转和加入新边的时候进行维护（暴力枚举其中的点分到新的 vector 中）。

接下来考虑时间复杂度问题，这需要我们对过程有一个更详尽的刻画：注意到每次处理的四边形有一个顶点总是 P，也就是说我们总是在处理这样的一条边 AB，PAB 的外接圆包含某个 C 的问题，考虑我们检查过并且判定是否其与 P 组成的外接圆包含另一个点的边 AB，当最终算法的执行过程停下来时，我们检查过的边要么违反了空圆性质被翻转，一端连向 P，要么组成了某个星形多边形的外轮廓（星形是指从某一点，这里是 P，能够看到多边形中的每一点）。

这样我们便可以度量当前每个点作为最后一个点插入时的翻转次数，即这个点在算法停止后的度数，由于图是平面图，故度数之和是线性的，随机化后当前点的期望翻转边数是 $O(1)$ 的。

再继续考虑我们维护每个面中含有哪些没被加入过的点的时间复杂度，同上，我们在做完之后的当前的每个面考虑其中的点，只有在当前最后插入的点为这个面的三个顶点之一的时候才会对这些点进行维护，也就是说对于后面的 $n-i$ 个点其只有 $\frac{3}{i}$ 的概率对其进行 $O(1)$ 次维护，求和后总时间复杂度为 $O(n\log n)$。

考虑是否做到了最优，我们将 Delaunay 三角剖分的问题归约到简单的排序问题上，取单位圆周上点 $(0,-1)$，并取一序列 $p_i, p_i \leqslant n$，取点 $(\cos p_i \cdot \frac{\pi}{n}, \sin p_i \cdot \frac{\pi}{n})$，则三角剖分问题等价于最后连接成一扇形，相当于给序列排序，则时间复杂度下界 $O(n\log n)$ 达到最优。

接下来我们以两道例题简述三角剖分的应用：

例题 1.3.1 (最小欧几里得生成树)**.** 给定平面上 n 个点，两两连边，边权为这两个点的欧氏距离，求最小生成树

依 Prim 算法考察最小生成树的过程，则每一条边的直径应该对应一个空圆，则这条边应该在 Delaunay 三角剖分上，故保留三角剖分的边做最小生成树即可。

例题 1.3.2 (最小权匹配)**.** 平面上给定 n 个圆心 X_i，确定它们的半径 r_i 使得 $|X_iX_j| \geqslant r_i + r_j$，求 $\max(\sum r_i)$

略去对偶与一些最优的考量，最终我们需要对这样的点对连边建图：设每个点最近点的距离 d_i，需要给 $|X_iX_j| \leqslant d_i + d_j$ 的点连边，在此我们可以使用三角剖分这一工具对边数进行考察。

考虑 i, j 连边，设其最近点各自为 p, q，若 $p \neq q$，则 i, j, p, q 形成一四边形且要么 i, j 之间的边在 Delaunay 三角剖分上要么 p, q 在 Delaunay 三角剖分上且 i, p, q 形成一个面，再考虑 $p = q$ 的情形，则对最近点同为 p 的点对于 p 来说只有 $O(1)$ 个（网格化进行估计）。

由这两个例子说明对 Delaunay 三角剖分进行的考察都是在空圆性质上进行的，而其大多数时候带来了一个简单有力的结构以解决问题。

与 Voronoi 图的对偶关系则是 Delaunay 三角剖分的另一大用途。按照 Delaunay 三角剖分建立的平面图，其对偶图（平面图意义下）被叫做 Voronoi 图，具有以下性质：

Voronoi 图中每个原点集中的点所在的面是距离这个点最近的点组成的，即求出平面上每个点在原来的点集中的最近点设为其管辖点，则每个原点集中的点的管辖区域的划分形成了 Voronoi 图。

Voronoi 图有特殊的算法可以求解（Fortune），不过其原理复杂并不如三角剖分对偶来得简单。

由于 Voronoi 图的以上性质，其通常用来求解某种距离相关的问题，容易发现沿着 Voronoi 图的边走就是沿着一条距离最近点最远的路线走，这就催生了以下题目：

例题 1.3.3 (SDOI2012)**.** 给 n 个点，以及一 S, T，要求找到 S 到 T 的路径使得其距离最近的关键点最远

可以二分最近距离，之后可以断掉 Voronoi 图的边，或者等价的，在 Delaunay 三角剖分中连上相应的边，检查连通性。

Voronoi 图也可以在一些算测度的计算几何问题中提供对平面的合适的划分：

例题 1.3.4 (自选)**.** 给平面上 n 个点，以及一矩形区域，求在矩形区域中均匀随机一个点，距离最近关键点的期望

按照 Voronoi 图划分区域后积分即可。

1.4 圆反演

几何问题总是伴随着更多几何变换的提出而变换着形式，好的变换能够使人拥有对几何的直观看出问题的本质。

圆反演就是这样的一种变换，经典的几何变换还有仿射变换，莫比乌斯变换，球极投影以及上文提到的点线对偶等等。

圆反演的定义如下：首先选定一反演中心 O，为方便起见我们将其挪至原点，然后选定一反演半径，采取适当的缩放我们设反演半径为 1，然后将平面中的点 $P:(x,y)$ 反演为一新点 $P^*:(x',y')$ 使得 O,P,P^* 共线，并且 $|OP|\cdot|OP^*|=1$

由于直线和圆是点的集合，类似可对一族点定义反演即直线和圆的反演为其上每一个点的反演组成的几何对象。

使用一个几何变换只需要了解其性质，圆反演的性质有：

- 不过反演中心的直线和过反演中心的圆互为反演，不过反演中心的圆被反演成另一不过反演中心的圆
- 相切，相交，相离的两对象反演之后仍旧相切，相交，相离（圆与圆，直线与圆）

由于对几何对象的反演的计算是机械且容易的，所以我们在思考题目时可以关注到题目中复杂的圆对象，直接套用圆反演观察得到的问题，大量的圆的出现，这也是圆反演名字的由来，比如这道例题：

例题 1.4.1 (HNMTC.cyc)**.** 给平面上不交且不过原点的 n 圆，试找一过原点的圆与最多圆相交

直接圆反演之后转化为找一直线与最多的圆相交，则其至少可与一圆相切，枚举此圆，只需钦定直线的角度，则对其余每个圆均有一个可贡献的角度区间，扫描线即可。

如同组合优化的对偶方法一样，出题人可以先出题再进行一系列地几何变换以隐藏题目的直观，选手也可以通过几何变换来重新发掘这样的直观。

1.5 三维凸包

对于三维空间的计算几何，我们主要依赖于向量进行计算，因为其数值稳定性和线性代数的方便性。我们可以通过叉积计算某一个面（其由三个点定义，其方向由三个点的顺序定义）的法向量，并通过与法向量的点积计算另一个空间中的点处于这个面的正面还是反面。

这就构成了求解三维凸包的全部基础知识，接下来我们仍然按照增量求解凸包，考虑在之前的凸包中加入一个新的点 P，则加入 P 点后，凸包有一些面需要被删去，对于之前的凸包我们可以给每张面规定外面和里面的区别，如果新的点 P 在所有面的里面，则其在

凸包内部，不需考虑，那么 P 点在其正面的面就是要被删去的面，对于凸包上的边，若其相邻两面分别里外有点 P，则这条边与 P 将新形成一张面，则我们每次加入新点需遍历一遍所有面，时间复杂度 $O(n^2)$。（传统的二维凸包亦可这样做）

对于经典的二维问题，我们也可以通过某些几何变换将其提升为三维更有直观，比如这样的变换：$(x,y)\to(x,y,x^2+y^2)$，则" 提升" 后的点全部落在一抛物面上，考察圆的方程 $x^2+y^2+ax+by+c=0$，则显然其为抛物面 $z=x^2+y^2$ 与平面 $ax+by+z+c=0$ 的交，这就是说平面中的圆被" 提升" 之后会处于同一平面上，这可以导出 P 在 ABC 外接圆内相当于是说 P' 在平面 $A'B'C'$ 下，其中 P',A',B',C' 均为变换之后的点，这就把一个较为复杂的问题转为了更加数值稳定的问题（这可以用在 Delaunay 三角剖分的检查中），具体来说，混合积的正负，只需求解一行列式。

考虑平面一点集在" 提升" 之后的三维凸包，根据其面的朝向我们可以把面分为两种，一种是朝下的，所有点均在其上，这种面对应的是内圆也就是 Delaunay 三角剖分的每一个三角面，那么朝上的就相应对应另一种圆：外圆，所有点都在这三个点的外接圆内部，内圆和外圆在三维凸包上得到了统一。

关于此有一道例题，其中光是题面就结合了圆反演的知识，我们用这样一道结合了很多算法知识的题目作为本节的收尾：

例题 1.5.1 (ZJOI2018. 保镖)**.** 给定 n 个点和一矩形区域，求在区域中均匀随机一个点 P 之后依 P 为反演中心，1 为反演半径将所有点作反演之后的凸包期望点数

题目留给读者当作练习。

2 几何问题衍生的数据结构

对数据结构的研究始于计算几何，一些数据结构的提出最开始也是来自计算几何问题，例如平衡搜索树的出现，以及 fractional-cascading 之类的数据结构技巧，本章先传统地以两个经典命题开始，并在第三部分引入较为现代的研究内容。

2.1 点定位

对于静态的点定位问题，我们已经再熟悉不过，不过对于动态的问题，我们只需一些数据结构来维护。

对于支持动态插入删除边，在线求出询问点在哪一个面中的问题，只需注意到我们只需求解询问点向上第一条边是哪条边。

我们首先使用线段树划分给定线段的 x 坐标，对于线段树区间 $[l,r]$，x 坐标完整包含 $[l,r]$ 的线段且不包含其父区间的会被保留其中，按照相应的端点的 y 坐标排序。

则查询所有包含询问点 x 坐标的区间，可以求得其 y 的前驱，假使每个区间查询花费 $O(\log n)$，总共就会花费 $O(\log^2 n)$ 的时间进行查询。

加入和删除线段只需要进入相应的 $O(\log n)$ 个区间并以 $O(\log n)$ 每次的时间在相应的平衡树中删去线段。

如果改变想法，我们使用静态的扫描线去维护静态的平面图，并以可持久化平衡树维护对应位置的 y 排序，我们可以以单 log 的时间复杂度解决静态在线询问的问题，这是可持久化数据结构在历史上第一次被提出用来解决的问题。

类似地，如果把可持久化数据结构换为可追溯化数据结构，我们可以支持维护所有边均为水平或者垂直的动态点定位问题，这是可追溯化数据结构第一次被提出时用以解决的问题。

2.2 正交查询

正交查询需要解决这样的问题：给定 d 维空间中的 n 个点，每次询问给定 d 维空间中一矩形 $[l_1, r_1] \times [l_2, r_2] \times ... \times [l_d, r_d]$，得到有关这个矩形中的点的答案，可能是有没有点，点的个数以及得到里面的所有点。

正交一词来源于询问的矩形，其边缘互相正交，我们将其称作正交范围。

一个简单的想法是使用 d 维线段树，这样答案就以 $\log^d n$ 的时间被表示为 $\log^d n$ 棵子树（在一维的情况，这是线段树第一次被提出的目的，range trees）。

接下来我们先考虑 $d = 2$ 的情况，我们为每个 x 线段树区间排序而不是建立另一棵有序的搜索树，这样我们就得到了经典的划分树结构，我们将在这棵树上查询时进行搜索的复用。

如果继续暴力地做，我们需要在 $\log n$ 个线段树区间上的数组进行二分搜索，容易发现其实我们只需在线段树的根上进行二分搜索，并对线段树的每个区间的每个元素保留其在两个线段树儿子的区间中的前驱后继，我们就可以通过父亲的搜索结果跳转 $O(1)$ 个前驱获得儿子的搜索结果，并把答案表示为 $O(\log n)$ 个区间，将同样的方法推广到 d 维，我们只在最后的维度上执行这样的记录，可以做到 $O(\log^{d-1} n)$ 每次询问，$O(n \log^{d-1} n)$ 预处理和空间。

接下来，我们将这个问题变成动态的，这意味着插入节点，并询问正交范围。

显而易见的，我们的线段树必须被改造为重量平衡树（推荐使用旋转的 Treap，便于实现），仍然先考虑 $d = 2$ 的情形，到儿子的指针这时需要特殊维护，不妨考虑外层平衡树的每层的节点，将其依据来自哪棵子树分为黑点和白点，则在层内相当于查询异色的前驱后继，我们考虑在插入一个节点时启发式分裂，在其异色部分的短的那一边暴力修改，长的那一边与之前的指针交换标号，时间复杂度对于更新来说是均摊 $O(\log^d n)$ 每次，证明这一点只需考虑对黑色节点的插入（白色节点同理），不妨设这一层在重构前的白色节点全部在黑色节点前插入，这样重构前的时间复杂度达到顶，而看作笛卡尔树上的 dsu on tree 可以得到这一复杂度。查询时间不变。

在更高的维度中，一些更有趣的数据结构可以被作用在几何中（它们因此提出），在 $d \geqslant 3$ 的时候，存在一种使用空间换取询问时间的 fractional-cascading 做法。

接下来我们只考虑静态的问题，不妨设 $d = 3$（更高的维度可以套取线段树，动态问题可以套取重量平衡），首先假设询问是 $[l_1, r_1] \times [l_2, r_2] \times [l_3, r_3]$，通过对第二维和第三维做 OI 中称之为" 猫树" 的分治，我们可以将第二维和第三维割裂成 4 个 2-sides 的问题即 $[l_1, r_1] \times [-\infty, r_2] \times [-\infty, r_3]$ 以及其他三个方向，转换的代价则是最初的预处理部分增加两个 $\log n$。

然后考虑 $[l_1, r_1] \times [-\infty, r_2] \times [-\infty, r_3]$ 的问题如何做到单 log（以及附加一个正比于答案的部分），对于 $[l_1, r_1]$ 我们使用线段树进行划分，对于线段树每个区间内部，自然是以第二维排序，那么怎么在这一区间所代表平面上的点进行 2-sides 的查询呢。

考虑平面上的点 (x, y)，其向 y 轴正无穷的方向引了一条射线，$[-\infty, r_2] \times [-\infty, r_3]$ 的查询相当于是在 $(x, y) = (r_2, r_3)$ 处向左（x 坐标负无穷的方向）引一条射线，并把这条射线所交的向上的射线所代表的关键点取出即为答案。

那么怎么取出那些与询问射线有交的射线呢，我们首先对原来的所有射线做处理，把关键点再向左向右发射射线直到碰到其他朝上的射线，以此划分出一些网格，如图 1，其中蓝色点为关键点，橘色的点为询问点，绿色射线穿过的蓝色射线即为相交的部分。

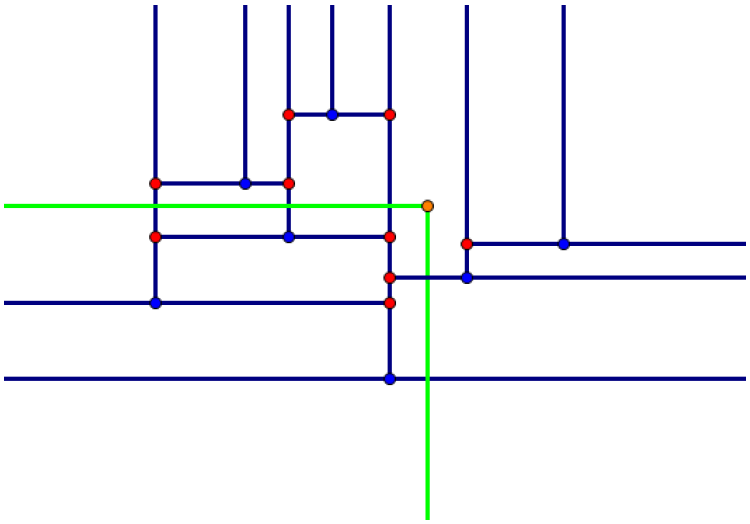


图 1: 对点集的预处理

我们现在需要考虑从左至右找到绿色射线穿过的每一块（上文所写的网格，其中每一张面，可以类比砌墙的砖块）直到 x 坐标到达 r_2 就停止，而每一块到其右边的所有块之间又需要一次二分检索 r_3 所在的块，容易想到从右至左进行 fractional-cascading 的合并以确定 x 增加时从当前块到其右边的哪一块，而在最开始我们也需要进行一次二分找到 r_3 所属的第一块（这一块比较特殊，其左边界为 $x = -\infty$），因此这样的第一次搜索（在无界块中的搜索）相当于是在当前这个线段树节点（最初对第一维的线段树划分, 因此内层二维平面无界块的第一次搜索实际上有 log 个，是在我们当前正处于的线段树节点上做的）上有一个列表，需要进行搜索，我们对线段树上的所有列表再进行 fractional-cascading 即可（从儿子

合并 $\frac{1}{3}$ 的列表）。于是我们对 r_3 处于哪一个块的搜索从线段树的根便开始了，总共的询问复杂度为 $O(\log n)$（之后我们就获得了每个线段树节点对 r_3 在其中哪一块的定位，之后便以 $O(1)$ 每个块的跳转速度获取相交的每个节点，其中 $O(1)$ 由 fractional-cascading 的跳转指针做到）。

由于最初的两次分治，预处理的时间和空间复杂度达到了 $O(n\log^3 n)$。

2.3 separator

随着机器学习的发展，聚类和划分算法层出不穷，人们探索一些有关 seperator 的组合问题时，研究出了很多有用的理论。

我们先引入一个简单的定理：

定理 2.1 (separator)**.** 给定一张平面图 G，其点集可以被分为三部分 A, B, C 其中 A, B 之间的点无边相连，$|A|, |B| \leqslant \frac{2n}{3}$，$|C| \leqslant 2\sqrt{2n}$

对于一张已经给定了嵌入的平面图，即知道了一种画在平面上的方法的平面图，我们直接给出一个线性的算法，以形成对定理的证明。

第一步，我们先把整张平面图补全为一张三角剖分，然后我们得到补全之后的图的 bfs 树，不妨设层分别为 $0, 1, ..., r$，其中第 0 层为根，设每一层的点集分别为 $L(0), L(1), ..., L(r)$。

第二步，找到最小的 i 使得 $0, ..., i$ 之间的点数第一次 $\geqslant \frac{n}{2}$，设这个点数为 p，找到一 $j \leqslant i$ 使得 $|L(j)| + 2(i-j) \leqslant 2\sqrt{p}$，找到一 $k > i$ 使得 $|L(k)| + 2(k-i-1) \leqslant 2\sqrt{n-p}$，通过反证列出不等式，我们可以证明这样的 j, k 一定存在（k 有可能大于 r，此时 $L(k)$ 为空）。

第三步，考虑深度在 $[0, j-1], [j+1, k-1], [k+1, r]$ 中的点，将它们分为这三部分，如果其中最大的部分的大小 $\leqslant \frac{2n}{3}$，则令 $C = L(j) \cup L(k) \leqslant 2\sqrt{2n}$，再令 A 为其中最大的部分，B 为剩下两部分的并，这样 $|A|, |B| \leqslant \frac{2n}{3}$。

否则，根据我们的定义，仅有中间的这一部分有可能会大于 $\frac{2n}{3}$，我们要证明其中存在进一步的 separator 进行分割，考虑将 $L(j)$ 中的所有点缩成一个节点，删去在 $[0, j-1], [k, r]$ 中的节点，我们得到了一张直径不超过 $k-j-1$，点数多于 $\frac{2n}{3}$ 的小图，不妨设这张小图的点数为 n'，接下来，我们找到这张小图的任一生成树，则任取一非树边都能形成一个环，这个环分离了其中的点和其外部的点，因此是一个 separator，找到所有的这样的环中，分成的两部分中较大部分最小的那一个环，这就是我们要找的小图的 separator。

为了继续证明这一点，不妨设这个圈对应的非树边为 (x, z)，且较大的那一部分为圈的内部（否则我们可以通过一些变换使内外翻转，具体来说是先对应到三维凸包，再将凸包翻转映射回平面），若内部的点数已经 $\leqslant \frac{2n'}{3}$，则已证毕，否则考虑 (x, z) 这条边对应的圈内的三角面的第三个点 y，我们用 y 导出矛盾。

若 $(x, y), (y, z)$ 在圈上，则圈就只由 (x, y, z) 组成，这是不可能的。

若 (x, y) 在圈上 (y, z) 不在圈上，则 (y, z) 必为非树边（否则树边成环），取非树边为 (y, z) 的这个环的较大部分会更小，与 (x, z) 的取法矛盾。

若 $(x,y),(y,z)$ 均不在圈上，这时又有两种情况，因为 $(x,y),(y,z)$ 不可能同时为树边，若其中一条为树边，不妨设为 (x,y)，则取 (y,z) 这个圈内部（此时不包含 y 了）更小，与 (x,z) 取法矛盾。

若 $(x,y),(y,z)$ 都不为树边，由生成树的定义及 Jordan 曲线定理，一定存在 y 连向圈上除了 x,z 之外某个点 w 的路径 p，这样 $(x,y),(y,z)$ 构成的俩环都是 (x,y,p,x) 或者 (z,y,p,z) 的形式，选取其中一个内部较大的环，不失一般性设为 (x,y)，由于 (x,z) 内部的点多于 $\frac{2n'}{3}$，则其中大的一部分加上 $|p|$ 就一定大于 $\frac{n'}{3}$，因此此时 (x,y) 的外部的点一定不超过 $\frac{2n'}{3}$，如果其内部的点没有外部多，那么我们就找到了一个合适的 separator，否则其内部的点数仍然比外部多，则这个环内外较大的部分比 (x,z) 组成的环来得少，矛盾。

综上我们对于小图找到了一个 separator S，且其大小不超过 $2(k-j-1)+1$，取 $C=S\cup L(j)\cup L(k)$ 则 $|C|\leqslant 2(k-j-1+1)+|L(j)|+|L(k)|\leqslant 2(\sqrt{p}+\sqrt{n-p})\leqslant 2\sqrt{2n}$，且 C 将图分成的四部分均不超过 $\frac{2n}{3}$，由经典的分析可知可以组成 A,B 使得每部分不超过 $\frac{2n}{3}$。

接着，我们的一个想法是定理中的 $\frac{2}{3}$ 是否可以改进，结论如下：

定理 2.2 (advanced)**.** 平面图 G 的点集，可以被分为三部分 A,B,C 其中 A,B 之间无边相连，$|A|,|B|\leqslant\frac{n}{2}$，$|C|\leqslant\frac{2\sqrt{2n}}{1-\sqrt{\frac{2}{3}}}$

我们使用一点技巧来完成改进的 separator 的构造，我们维护当前的 A,B,C，并且维护一个剩余的部分 D，并且保证时刻都有 $|A|\leqslant|B|$ 且 A,B,C 组成 separator 的构造，初始时 $A=B=C=\varnothing,D=V$。

每一次，我们都把 D 划分为 A',B',C' 其满足前一个定理，不妨设 $|A'|\leqslant|B'|$，我们令 $A=A\cup A'$，$C=C\cup C'$，$D=B'$，并在 $|A|>|B|$ 的时候交换 A,B，这样每一次 D 至少减少 $\frac{1}{3}$，且最终得到了一个 separator，满足改进后的定理。

在 $2\sqrt{2}$ 这个因子的改进上，我们亦有结论：

定理 2.3 (Djidjev)**.** $2\sqrt{2}$ 可被改进到 $\sqrt{6}$，并有下界 $\frac{\sqrt{4\pi\sqrt{3}}}{3}$

作为一个强力的划分的工具，其自然会与图上的分治算法所结合，例如以下问题：

例题 2.3.1 (平面图独立集计数)**.** 给定一张平面图，求它的独立集的数量

对于给定的平面图，我们找到其 $O(\sqrt{n})$ 大小的 separator C，之后枚举 C 中节点的状态，然后递归到分离出来的子图，总时间复杂度为 $T(n)=2T(\frac{n}{2})\cdot 2^{O(\sqrt{n})}$，解递归式得到 $2^{O(\sqrt{n})}$。

对于如此有力的工具，我们自然要问，对于更多类似的图是否有有效的 separator，答案是肯定的，我们给出下面的定理：

定理 2.4 (亏格版本)**.** 对于一张具有亏格 g 的图 G，存在一个点集 C，且 $|C|\leqslant 6\sqrt{gn}+2\sqrt{2n}+1$，当我们将 C 从 G 中移除之后，最大的连通块大小不超过 $\frac{2n}{3}$，并且 $O(\sqrt{gn})$ 是下界。

对于这个定理，我们先引入一些必要的拓扑知识，并给出一个线性求解的算法。

定义 2.1 (基础知识)**.** 图的节点被定义为空间中的点，其边被定义为空间中的曲线（一维流形），则 G 到可定向曲面 S 的嵌入为一个 G 上的点到 S 上的点的对应，使得对应的边无交。

图的面（*faces*）被定义为 $S-G$ 的连通部分，通俗的理解就是在 S 中删去 G 的边，从而把 S 划分了很多块，这些块就是面。

我们称一个嵌入为 *2-cell embedding*，当其中所有的面都可以与某个二维平面中的开盘同胚。

如果一张图能够嵌入在亏格为 g 的表面上，而不能够嵌入在亏格为 $g-1$ 的表面上，我们就称这张图的亏格为 g，同时称这样的表面为这张图的亏格表面。

其中亏格的概念简单来讲就是曲面的洞的数量，也是在一颗球面上所粘接的环柄的数量，拓扑学中经典的咖啡杯与甜甜圈等价说的就是亏格相等。

对于图在亏格表面中的嵌入，我们有如下经典结论：

定理 2.5. 任一张图在其亏格表面中的嵌入都是 *2-cell embedding*

定理 2.6 (Euler)**.** 对于一张点数为 n，边数为 m 的图，其有一个在亏格为 g 的表面里的 *2-cell embedding*，且面数为 f，则有 $n-m+f=2-2g$

定理 2.7. 定义图 G 上的收缩操作 $Con(u,v)$ 为将之间有边的点 u,v 合并为同一节点 w，且 w 的邻居就是 u,v 各自的邻居，收缩操作不会使图的亏格增加。

限于作者的拓扑水平，以及篇幅限制，此处不对这三个定理进行证明。

在 separator 定理的基础上，我们可以进行一些预处理，使得删去一个至多 $6\sqrt{gn}+1$ 个点的部分之后，剩下的图为平面图，具体操作如下：

首先找到图 G 的一棵 bfs 树，找到其中间的一层 i 使得前 $i-1$ 层和 $i+1$ 到最后一层都只有不超过 $\frac{n}{2}$ 个节点。

找到 i 前的最后一层 j，使得其有不超过 $\sqrt{gn}$ 个节点，找到 i 之后的第一层 k，使得其节点数不超过 $\sqrt{gn}$（k 可能不存在，此时令其为所有层数 +1），易知 $k-j\leqslant\sqrt{\frac{n}{g}}$。

将第 j 层的点收缩为一个，并删去所有 j,k 层之外的点，我们得到一张图 H，接下来我们对 H 进行讨论。

首先我们找到 H 的一棵生成树，将 H 划分为树边和非树边，我们将删去某些非树边直到最后仅剩下一张面，并且这张面还是 2-cell 的，即同胚于某一平面上的开盘（其实就是同胚于一张平面上的面）。

假设当前有超过一张面，任意挑一张面 F，我们可以通过沿着其边走的左转法得到这张面上的所有边，与平面图所不同的是，有的边会将某一条边的正反都经过一次（假想在一张 torus 中嵌入 K_5，存在一张面发生这样的情形）。

对于任何一张面，都存在一条非树边，使得其仅被这张面经过一次，然后我们将删去这条边，接着重复这样的操作。而如果我们预先对所有面都找到其经过的环，就像在平面图中找到所有面那样使用最小左转法，再一条一条删去这些非树边使得最后只剩下一张面，

那么其实就相当于建立这张图的对偶图的生成树（此前以及此后的生成树都是原图的生成树，对偶图的生成树仅在此出现一次），然后从叶子开始删。

之后我们继续删去当前图中的一度点，即生成树的叶子，最后我们得到的是一张面，剩下的生成树和非树边形成这张面的边界。

而在这样操作之后，由欧拉定理可知，$n' - m' + 1 = 2 - 2g$，其中 n', m' 为当前的点数与边数，即 $m' = n' - 1 + 2g$，观察图的结构发现，生成树的叶子即对应非树边的一端，那么点数减去 1 就是树边的数量，可以得到非树边的端点数量为至多 $4g$。

又因为这棵生成树可以看作是由很多条路径并成的，每一条路径都是一个非树边的端点到生成树的根形成的路径，由之前的过程，如果我们确定生成树的根为 G 中第 j 层收缩得到的节点，则其高度不超过 $\sqrt{\frac{n}{g}}$，那么总共的节点数就是 $4\sqrt{gn} + 1$（算上根）。

加上之前图 G 的第 j, k 层节点，一共是 $6\sqrt{gn} + 1$ 个点，删去这些节点之后，剩下的 H 中的部分是一个剩下的唯一的面中的嵌入，即一平面嵌入，是平面图。

对剩下的平面图使用 separator 定理进行分解，我们就得到了整张图不超过 $6\sqrt{gn} + 2\sqrt{2n} + 1$ 个节点的 separator。

这就意味着，我们可以对一些较为有规律的图进行手动的嵌入，然后在上面进行各种分治合并的算法，笔者暂时没有找到这方面的题目，但是更多类型的图以及不同程度简化的 separator 以及特殊设计的分治算法一定可以进行考察。

上述算法都需要依赖一些特定的嵌入，我们自然又要问，是否存在一些统一有效的算法，不需要借助图在表面的嵌入，内蕴地给出图的一个 separator，这样的算法也是存在的。

在给出算法之前，我们仍然需要一些前置知识：

定义 2.2. 对于图 G，定义其 *Laplacian* 为矩阵 $L(G) = D(G) - A(G)$，其中 $D(G)$ 为 G 的度数矩阵，$D(G)_{i,i} = deg_i$，$A(G)$ 为 G 的邻接矩阵，即对边 (i, j)，$A(G)_{i,j} = A(G)_{j,i} = 1$。

对于 $L(G)$，有性质：

- $L(G)$ 是对称半正定矩阵，有特征值 0
- $L(G)$ 的第二大特征值 λ_2，被称为 G 的代数连通度，如果其为 0，那么图不连通

其中，证明较为显然，在此略去。

我们还可以定义图 G 的 Fiedler 向量为 λ_2 对应的特征向量 v_2，在此我们给出被称为谱划分的启发式算法：

找到 v_2 的中位数，将中位数及左边的点分到 A 中，右边的点分到 B 中，这样我们得到了一个点数相差不超过 1 的划分，属于特殊的 separator，可以称作 bisection。

令 $C(A, B)$ 表示集合 A, B 之间的边集，我们定义一个分割 A, B 的割率如下：

定义 2.3 (cut ratio). $\phi(A) = \frac{|C(A,B)|}{\min(|A|,|B|)}$

显然，更小的割率具有更好的效果，那么谱划分算法的效果如何呢，我们只需以下定理进行估计：

定理 2.8 (Fiedler's inequality)**.** $\lambda_2 \leqslant O(\frac{g}{n})$

定理 2.9 (Courant-Fischer)**.** 令 A 是一埃尔米特矩阵，即其共轭转置等于其自身，则对于其第 k 大特征值，有：$\lambda_k = \min_{U,\dim U=k} \max_{x\in U, x\neq 0} \dfrac{x^T A x}{x^T x}$

定理 2.10. 设 G 有最大度数 Δ，对于任何与全 1 向量正交的向量 x，存在一 s 使得 $A = \{i|x_i < s\}$ 与 $\overline{A}$ 形成的割的割率不超过 $\sqrt{2\Delta \frac{x^T L(G) x}{x^T x}}$

因此对于度数有界的图，我们可以找到其割率不超过 $\sqrt{\frac{g}{n}}$ 的割（当然不一定对应 bisection），对于度数方面的限制，大多数的问题允许我们进行三度化，比方说最大生成树，最短路等等，我们都可以正确确定边权，而且图的亏格不发生变化。

而对于图的 Laplacian 其有最优化方面的更多变形，在求解 λ_2 方面，选手可以采用最基础的迭代法，以实现简单的算法达到不错的效果的目的，因此谱划分算法不失为考场上拿分的一种良好的选择。

综上，我们对 separator 这一对象进行了较为深入的讨论，其比较符合平面图存在可以切一刀，且能把图分成均匀两半这样的几何直觉，并实际地给出了构造，是某种直觉的具象化，其还可以结合一种新兴的，高效的被称为" 核化" 方法的技巧解决一些看起来困难的问题（网格化也是一种方法，能否联系其进行一些算法之间的联合，发挥更大的作用），能够在 OI 中出大量的题目进行考察。

3 有限几何

有限几何是最抽象的一类几何，在这里我们只关心 incident 的关系了。即直线和直线是否相交，点是否在直线上这样的关系，并将以公理化的形式抛弃原来的欧式几何。有限几何的作用，是用来刻画一些组合，密码学问题的结构，为大量的 OI 中出现的诡异构造题目提供背景与基础，降低了这些构造问题表面上的难度。

我们假想一个具有以下公理的几何结构：

公理 1.

(a) 过任意两点都有唯一的一条直线

(b) 任何两条直线都有唯一的交点

(c) 每条直线都有至少三个点

这样的几何显然是不同于欧式几何的，因为平行的直线就没有交点，实际上我们可以找到一些例子，它们被称作射影平面，例如较为经典的在欧式平面上添加一条无穷远直线。

最简单的射影平面是 Fano 平面，其具有 7 个点和 7 条直线，在有限集合里，平面的概念如下：

定义 3.1. 对于平面 P，其具有点集 S 和其上的子集族 L，被称为直线集，其中每一个子集代表一根直线，点在直线上当且仅当点在对应的子集里，直线的交点是对应集合的交集。

对于射影平面（即满足公理 (a)，(b)，(c) 的平面），我们可以推出以下事实：

推论 3.0.1.

(d) 所有的直线所含点一样多

(e) 对于每一个点均有同样多的直线经过它

(f) 所有的点是不可区分的，所有的直线是不可区分的，即存在一种重新分配标号的方式使得任何两个点都在重命名标号下等价

其中 (d) 的证明可以考虑对于直线 l_1, l_2，找到其交点之间的对应，这只需要利用 (c) 和 (b) 找到一些点和直线即可。这三条推论同样可以成为公理，其揭示了射影平面的某种对称性（以及，还跟拟阵有关系）。

接下来，我们在射影平面的基础上，改动一些引理，引入平行的概念，得到仿射平面：

公理 2.

(a) 过任意两点都有唯一的一条直线

(c) 每条直线都有至少三个点

(g) 对于任何一条直线 l 以及一个不在其上的点 p，都有唯一一条与 l 平行的直线经过 p，此处平行所指即没有交点

仿射平面的一个有限的例子即 Tictactoe 平面，最经典的莫过于我们的欧式几何的笛卡尔平面，那么这样就有一个显而易见的转化射影平面和仿射平面的方法：

对于仿射平面，我们将所有直线按照平行关系的自反闭包分为等价类（显然平行关系补上自反关系是等价关系，传递性由公理 (g) 推出），然后为每个等价类添加一个新的无穷点，把所有得到的新的无穷远点添加进同一条直线，即无穷远直线，这样我们就得到了一张射影平面，如同我们在欧式几何到射影几何所假设的那样，反之，对于射影平面，我们只需去掉其中任何一条直线都能得到一张仿射平面。

接下来，我们可以设计各种各样的结构，并为他们赋上一些公理，例如下面的几何：

公理 3.

(A) 任何三个点都决定了一张唯一的面

(B) 任何两张面要么平行，要么交出了一条线

(C) 对于任何一张面 F 和不在其上的一个点 p，都有恰好一张面经过 p 且与 F 平行

(D) 任何两个点都不可区分，任何两张面都不可区分

这样的一个接一个几何对象，大多数时候都存在于构造题之中，作为构造对象的结构，因此，我们需要将其实际的表示出来，以便构造，在此之前我们先看更多这些几何所具有的性质：

定理 3.1. 假设有限的射影平面上每条直线都有 $n+1$ 个点，我们将其称作 n 阶射影平面，n 为这个射影平面的阶，则这个射影平面有 n^2+n+1 个点和 n^2+n+1 条线。

证明可由 (d)，(e) 推出。

类似的，对于仿射平面我们也有一样的定理，并且根据前述的构造方法，我们可以由射影平面构造出仿射平面：

定理 3.2. 对于一个 n 阶射影平面，其删掉一条直线得到的对应仿射平面，我们称作 n 阶仿射平面，其具有 n^2 个点和 n^2+n 条线。

接下来我们可以从另一种射影几何（不是欧式几何改的那个射影几何）的定义中得到射影平面的构造：

定义 3.2. 给定一有限域 F（我们知道其具有素数幂大小 $q=p^k$），在其上有一个 $n+1$ 维的向量空间，射影几何 $PG(n,q)$ 定义在这个空间上，它的点，线，面被定义为这个空间的 *1,2,3* 维子空间，它们有一个统一的名字：*variety*，一个 $k+1$ 维的子空间被叫做 *k-flat*，所以一个点就是一个 *0-flat*，我们说两个 *variety* 有关系当且仅当其中一个包含另一个，也就是点在直线上之类。

由此，我们可以构造出射影平面 $PG(2,q)$，并且我们知道 k-flat 的数量由高斯系数确定：$\#k-flat=\begin{bmatrix}n\\k\end{bmatrix}_q=\prod_{i=0}^{k-1}\frac{q^n-q^i}{q^k-q^i}$，数量是一种很重要的特征（数学关系），在之后的例题中我们会见到。

在描述问题的结构中，我们有时会得到一些非常平均的结果，对于这些结果我们难以表达，这时我们还需要借助另外一种组合的描述：

定义 3.3. 块设计（*block-design*），是一个集合 S 和一个其上的集合族 F，其中 $|S|=v,|F|=b$，F 中的每个集合具有相同的大小 k，每个 S 中的元素都恰在 r 个集合中，且对于任意不同的 t 个元素，它们恰好同时在 λ 个集合中。我们简记这样的块设计为 $t-(v,k,\lambda)$ 设计。

对于块设计，我们有一些显然的结果：

定理 3.3. $bk=vr$，$\lambda(v-1)=r(k-1)$

现在，我们可以用块设计描述更多的问题，比如 Steiner 三元组问题，可以被描述为 $2-(n,3,1)$ 设计。而如果我们把射影平面中的点集看作 S，线集看作 F，我们也能得到独特的块设计。

接下来，我们用有限几何的结构去解决一些问题：

例题 3.0.1. 找到具有渐进最优大小的一个 $n\times n$ 网格的一个子集，使得其中不含一个子矩形上的四个顶点。

这道题目极为经典，具有多种构造，如果我们把 $\{1,2,...,n\}$ 视作点集，其上的集合视作直线，并且每行都视为一条直线，那么我们可以得到任何两条直线都至多交于一点的限制，如果 $n = q^2 + q + 1$，按照射影平面去构造直线集，我们就能得到 $(q^2 + q + 1)(q + 1)$ 个点的最优解，对于 $n = q^2$，我们同样得到 $(q^2 + q)q$ 个点的解，其中类似 $q^2 + q + 1$ 的数字可以让我们想到对应的有限几何对象。

类似于此题的做法，通过有限几何的思考我们还可以解决诸如 ARC140E 这样的题目。

例题 3.0.2 (JV. 不败的无尽兵团)**.** 找到一个渐进最优的 $3-(529,k,1)$ 设计或者 $3-(625,k,1)$ 设计，有关 k 的描述此处并不准确，集合大小不需相同。

对于具有有限集合背景的题目，我们首先是要找到数量关系，例如此题中的 $529 = 23^2$ 以及 $625 = 5^4$，再看几何（结构）关系，例如此例中我们可以用块设计来进行描述，接下来是分析量级，选择合适的代数结构，根据对应的有限几何结构构造具有相应特征的解。

在此例中，我们有：

公理 4 (Mobius plane)**.**

(A1) 对于任意三个点都有恰好一个环经过这三个点

(A2) 任何一个环 C 及其上的一个点 P，和不在圆上的点 Q，都有恰好一个经过 Q 的环切 C 于 P，相切即集合的交为 1

(A3) 至少有一个环，每个环上至少有三个点

因此，对于点和环的系统，其恰好构成了 3-design，这正是我们需要的，对于 Mobius 平面，我们有构造：

构造 1.

首先找到一仿射平面 A，我们加入一个无穷远点以及新的 *cycles*，即环的集合由以下组成：

(1) 平面 A 中的直线并上无穷远点

(2) 平面 A 中满足公理的环

因此我们套用 Mobius 平面的构造，先构造一个仿射平面 A（而这又需要先构造一个射影平面），加入其中的直线并上无穷远点。至于剩下的环的构造，由 Mobius 平面具有的特征，比较容易联想到常见的二次曲线椭圆，我们让具有一定参数的椭圆得到的集合作为 A 中自身的环，在此基础上，读者应有域构造的相关知识，以完成构造，对于满足公理的正确性，则应当依靠数论知识（有限几何提供最初的思路），读者在相应构造时，可以多尝试几种椭圆的离心率以及不同的参数得到合适的集合。

实际上通过计算，我们可以得到 Mobius 平面对应 $3-(n^2+1,n+1,1)$ 设计，经过对集合的截断（即抛掉多余的点），我们可以得到此题的解。依此，对题目数据的猜测有的时候并不需要准确，而是需要一个量级作为参考。

通过更多的有限几何知识我们还知道，如果给定的 $n = 2^k$ 我们还可以通过 $AG(n, 2^k)$ 来构造（AG 是 PG 对应的仿射几何），以及有限几何中更加丰富的各种公理和平面，例如 Minkowski 平面，Laguerre 平面（因此可以对应的平面为背景制造大量高质量的构造题目）。

例题 3.0.3. 在左右均有 $n = 343 = 7^3$ 个点的二分图中连尽可能多的边使得无 $K_{3,3}$ 子图

将右部点集看成点，左部点看成其上的集合族，即邻点为一集合，考察这一集合的性质，亦即，任意三个点都至多有两个集合包含它们。我们可以尝试先构造一些基础的部分，例如任意三个点都至多有一个 cycle 经过它们，再让题目中的集合满足任意一个 cycle 都有至多两个集合包含它，这又可以转化成对于一个集合和其上的一个 cycle，至多（转化成恰有比较平均）有另一个集合包含这个 cycle，不难由 7^3 的特征想到三维空间，以集合的性质想到半径固定的球，构造相应的代数方程即可。

例题 3.0.4. 在 $[1, n]$ 中找出具有渐进最优大小的子集，使得其中两两异或值不同。

先观察此题，最大的集合显然具有 $O(\sqrt{n})$ 大小，异或的特征不免让人想到域 $GF(2^k)$，这些是基本的数学特征，至于几何结构特征，我们可以采取有限几何给我们的思路，不过我们可以不局限于某些具体的平面的构造，$\sqrt{n}$ 便让人联想到 $\sqrt{n} \times \sqrt{n}$ 平面上的一条代数曲线，因此我们按照曲线的次数从低到高依次试一试即可（对应于一个方程的零点集）。

综上，在构造中使用有限几何，一方面需要我们有丰富的基础知识，另一方面需要做一些基本的观察，并且有丰富的几何联想，将构造同二次曲线，代数曲线联系起来（这便成为代数几何）。

本章介绍的有限几何仅仅只是一个开头，对于有限几何的真正研究，其后还有有限几何的基本群，buildings 以及一些著名的结果，可以解决更多的问题，希望本章能够起到一个抛砖引玉的结果，吸引更多同学研究有限几何（与有限几何并称的还有块设计和超图）。

4 总结

全文的内容，重点部分有对偶变换，网格化，三角剖分，separator，有限几何，希望读者能够掌握其中的内容，在几何上拥有直观，将几何的直观与其他的一些问题结合起来，也希望能够帮助读者在 OI 的道路上解决更多的题目，收获更多有意思的理论。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢退役 OIer 选手交流群群友提出的几何的题材，将我想写的部分串在了一起。

感谢陶立宇，唐绍轩，许庭强同学为本文验稿，唐绍轩同学提出的无数建议极大地丰富了本文。

感谢江城同学与我有关 separator 定理应用方面的讨论。
感谢重庆市巴蜀中学的黄新军老师给予的关心和指导。
感谢家人对我的关心，支持与鼓励。

参考文献

[1] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. SIAM Journal on Applied Mathematics, 36(2):177–189, 1979.

[2] John R Gilbert and Joan P Hutchinson and Robert Endre Tarjan. A separator theorem for graphs of bounded genus.Journal of Algorithms, 391-407, 1984.

[3] Kelner, Jonathan A. Spectral Partitioning, Eigenvalue Bounds, and Circle Packings for Graphs of Bounded Genus.SIAM Journal on Computing 35(4):882-902, 2006.

[4] https://en.wikipedia.org/wiki/Delaunay_triangulation

[5] Handbook of Combinatorics Volume 1

[6] Lovasz. Discrete Mathenlatics

[7] 詹兴致，矩阵论

浅谈回文串问题的相关算法及其应用

杭州第二中学 徐安矣

摘 要

回文串相关问题是信息学竞赛中的一类常见问题。本文介绍了与回文串问题有关的常用算法——字符串 Hash、Manacher 算法、回文树和用等差数列的形式维护所有回文后缀，还介绍了用等差数列的形式维护所有回文后缀的信息的扩展应用。

引言

回文串有很多优美的性质，但是处理回文串问题相关算法并不是很多。作者梳理了回文串问题相关算法和例题，并介绍了作者在研究回文串的性质时发现的用等差数列的形式维护所有回文后缀的一些优美性质，可以支持信息的合并，解决一些动态修改的字符串问题。

1 记号与定义

令 S 是一个字符串，记 $|S|$ 为字符串 S 的长度，记 $S[i]$ 为字符串 S 的第 i 个字符，记 $S[l\dots r]$ 为将 S 的第 l 个到第 r 个字符顺次排列形成的字符串，如果 $l>r$ 则表示空字符串，记 Σ 表示字符集大小。

定义 1.0.1. 满足 $\forall 1\le i\le |S|, S[i]=S[|S|+1-i]$ 的字符串 S 是回文串。

定义 1.0.2. 对于 S 的一个回文子串 $S[l\dots r]$，称它的回文中心是 $\frac{l+r}{2}$。

定义 1.0.3. 字符串 S 中，位置 i 的回文半径是最长的 x 满足 $S[i-x+1\dots i+x-1]$ 回文。

2 常用算法

2.1 字符串 Hash

定义一个把字符串映射到整数的函数 f，称这个 f 是 Hash 函数。

若两个字符串 S 和 T 满足 $f(S) \neq f(T)$，则 $S \neq T$，否则我们认为 $S = T$。若 $S \neq T$ 且 $f(S) = f(T)$，我们称为 Hash 碰撞。

我们可以选择合适的 Hash 函数减小 Hash 碰撞概率。

通过 Hash 函数我们可以 $O(1)$ 判断两个字符串是否相同。

2.1.1 Hash 函数的选取

可以选取 Hash 函数 $f(S) = \left(\sum_{i=1}^{|S|} B^{|S|-i}\text{val}(S[i])\right) \bmod P$，这里 val 是一个字符集到 $[1, \Sigma]$ 的映射，且 $\forall a \neq b, \text{val}(a) \neq \text{val}(b)$，$P$ 取大于字符集大小的质数，B 取任意大于字符集大小小于 P 的正整数，P 越大正确率越高。

这样选取的 Hash 函数的好处在于通过 $f(S[1 \ldots r])$ 和 $f(S[1 \ldots l-1])$ 可以快速查询子串 $S[l \ldots r]$ 的 Hash 值，即 $f(S[l \ldots r]) = \left(f(S[1 \ldots r]) - f(S[1 \ldots l-1])B^{r-l+1}\right) \bmod P$。

2.1.2 Hash 函数的正确率

考虑两个不同的 S 和 T，满足 $f(S) = f(T)$。记 $h(B) = f(S) - f(T) = \sum_{i=1}^{l}(s[i] - t[i])B^{l-i}$，其中 $l = max(|S|, |T|)$。可以发现 $h(B)$ 是关于 B 的不超过 $l-1$ 次非零多项式。

如果 S 和 T Hash 碰撞，那么 $h(B) = 0$，而 $h(B)$ 最多有 $l-1$ 个根，如果我们保证 B 是从 $[0, P)$ 均匀选取，那么 f(S) 与 f(T) 碰撞的概率不超过 $\frac{l-1}{P}$，实际应用中，可以将 f 看成是一个随机映射，因此正确率是 $\frac{1}{P}$。

为了提高正确率，可以通过选取多个 P 和 B 降低碰撞的概率。

2.2 Manacher 算法

在处理回文子串问题时，有时要求出字符串的回文半径，Manacher 算法可以做到线性时间处理出字符串每个位置的回文半径。

为了避免分奇偶讨论和边界问题，我们在字符串相邻字符之间加入一个特殊字符，再在字符串开头和末尾添加一个不同的特殊字符。如 aabbacab 变成 $#a#a#b#b#a#c#a#b#@。

由此我们得到了一个新的字符串 S，我们要求出对每个 i 求出 $R[i]$ 表示位置 i 的回文半径。

表 1: S 和对应的 R

S	#	a	#	a	#	b	#	b	#	a	#	c	#	a	#	b	#
R	1	2	3	2	1	2	5	2	1	2	1	6	1	2	1	2	1

2.2.1 算法流程

我们从小到大枚举 i 并计算 $R[i]$，维护两个辅助变量 r 和 pos，表示已求出的回文串覆盖到的最右边界和对应的回文中心。

计算 $R[i]$ 时，我们可以给 $R[i]$ 一个下界，分以下两种情况讨论：

1. $r < i$ 时，有 $R[i] \geq 1$。
2. $r \geq i$ 时，有 $R[i] \geq \min(R[2pos - i], r - i + 1)$。因为 i 和 $2pos - i$ 关于 pos 对称，如果只考虑 $S[pos - R[pos] + 1 \ldots pos + R[pos] - 1]$，它们的回文半径是相同的，而在这个子串中，$2pos - i$ 的回文半径是 $\min(R[2pos - i], r - i + 1)$。

有了 $R[i]$ 的下界，我们可以不断增加 $R[i]$，直到 $S[i - R[i] \ldots i + R[i]]$ 不回文，由此得到正确的 $R[i]$。

再用当前的 $i + R[i] - 1$ 和 i 更新 r 和 pos。

2.2.2 算法复杂度

考虑得到 $R[i]$ 的下界，然后不断增加 $R[i]$ 的过程。$R[i]$ 每增加 1，r 就增加 1，并且 r 不超过 n，说明 $R[i]$ 的增加次数是 $O(n)$ 次。而算法其它部分复杂度也是 $O(n)$，因此，算法的复杂度是 $O(n)$。

2.3 回文树

2.3.1 结构

一个字符串 S 的回文树由两棵树组成，设两棵树的根分别为奇根和偶根，回文树上除了根每个节点表示一个回文串，每个节点对应的字符串是父亲对应的字符串两端添加边对应的字符。特别的，偶根对应的字符串是空串，奇根对应的字符串是长度为 -1 的不存在的字符串，它的孩子对应的字符串是边对应的字符。同时，回文树每个节点有失配指针 $fail_i$，指向这个节点对应字符串的最长回文后缀对应节点。

2.3.2 构造方法

我们使用增量构造的方法构造回文树，假设现在已经构造出 S 的回文树，现在我们要在 S 的末尾增加一个字符 c，并维护出 Sc 的回文树。

引理 2.3.1. Sc 中不在 S 中出现的最多只有一个回文串，且是最长回文后缀。

证明. Sc 比 S 多的回文串肯定含有 c，因此必然是回文后缀，考虑 Sc 的一个回文后缀，若这个回文后缀不是最长的，那么这个回文后缀以最长回文后缀的中心对称必然出现在 S 中。

因此我们只要找最长回文后缀，如果没有这个回文串对应节点，为这个回文串新建一个节点，并维护出其回文树的父亲和失配指针。

我们沿着 S 最长回文后缀对应节点的失配指针走，直到找到节点 u 满足它到父亲边对应字符是 c。如果 u 节点没有字符 c 对应孩子，新建节点 v，它在回文树上的父亲是 u，父亲边对应字符是 c，$fail_v$ 是 w 其中 w 是沿着 u 的失配指针走找到的第一个节点使得 w 到父亲边对应字符是 c。

2.3.3 前端加入

前端加入我们只需要找到最长回文前缀对应节点，由于每个节点对应的都是一个回文串，$fail_u$ 指向这个节点对应字符串的最长回文后缀对应节点，同时也指向最长回文前缀对应节点，并且最多只会新增一个本质不同的回文串。因此与后端加入过程几乎相同。

2.4 用等差数列的形式维护所有回文后缀

引理 2.4.1. 字符串 S 的所有回文后缀按照长度排序后，可以划分成 $O(\log|S|)$ 段等差数列。

证明. 见参考文献 *[1]*。

有了这个性质，我们可以用等差数列的形式维护出 S 的所有回文后缀。

2.4.1 末尾加字符

我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。

新串的 S' 的回文后缀将会是：

1. 单独一个字符 c。
2. 原先的回文后缀在两端同时增加字符 c。

第一种情况新增的回文后缀就是单独一个字符 c。

第二种情况，考虑原先的一个极长的等差数列，定义 S^k 表示 k 个串 S 拼接形成的串，将这个等差数列表示的串表示成：$A, BA, BBA, B^3A, \ldots B^kA$。此时如果 B 的最后一个字符是 c，那么这个等差数列除了 B^kA 都出现在新串的回文后缀中；如果不是 c，那么这个等差数

列除了 B^kA 都不出现在新串的回文后缀中。而最后一项只需要检验它左边的字符是否是 c 即可判断它是否出现在新串回文后缀中。

做完这个，再把相邻的公差相同的等差数列合并成一个等差数列，以保证等差数列是 $O(\log|S|)$ 段。

2.4.2 扩展应用

见例题分析——集训队互测 2022 回文。

3 例题分析

3.1 APIO2014 回文串

3.1.1 题目大意

给定一个字符串 S，求 S 中出现次数乘长度最大的回文串。

$|S| \le 3 \times 10^5$。

3.1.2 算法分析

我们建出 S 的回文树，要维护每个节点代表的回文串的出现次数，我们对每个位置 i，找到 i 最长回文后缀对应节点，然后将这个节点沿着失配指针到根路径上节点的出现次数加一。将这个过程用树上前缀和优化即可。最后遍历每个不同的回文串，更新答案。

3.1.3 总结

回文树，有每个节点唯一对应一个本质不同的回文串的性质，是解决回文串问题的有力工具。

3.2 codeforces 1738H Palindrome Addicts

3.2.1 题目大意

你要维护一个字符串 S，初始为空，有 Q 次操作，每次可以：

1. push c：往 S 末尾插入字符 c。

2. pop：删除 S 的第一个字符，保证 S 非空。

每次操作后询问 S 有多少本质不同的回文串。

$1 \le Q \le 10^6$。

3.2.2 算法分析

我们发现一次操作答案改变量不超过 1，对于 push 我们只需要找到最长回文后缀，查询是否在加入前的字符串中出现过。对于 pop 我们只需要找到最长回文前缀，查询是否在删除后的字符串中出现过。

找最长回文前缀和最长回文后缀可以将 S 离线，每次末尾加，用等差数列的形式维护所有回文后缀，得到最长回文后缀。前缀同理。将字符串的 Hash 值放入哈希表，可快速判断是否出现过。

也可以不用等差数列维护，直接在回文树的 $fail$ 链上倍增，找到最长回文前后缀。

还有更加简单的做法，直接维护当前最长回文后缀，由于末尾加入字符，最长回文后缀增加长度不超过 1，且如果不回文肯定会减少直到找到新的最长回文后缀。

3.2.3 总结

对于同一个问题，可以有多种方法解决。对于有多种方法解决的问题，可以多加思考，选择代码难度低的写法。

3.3 Internal Longest Palindrome Queries

3.3.1 题目大意

给出一个长度为 n 的字符串 S，有 Q 个询问，每次给定区间 $[l,r]$，查询 $S[l\dots r]$ 中的最长回文串。

$1 \le n \le 2\times 10^5, Q \le 2\times 10^5$

3.3.2 算法分析

引理 3.3.1. 找到 $S[l\dots r]$ 的最长回文前缀 $S[l\dots i]$ 和最长回文后缀 $S[j\dots r]$。

证明. 考虑一个回文串，不妨假设它的中心 x 在 $[l,\frac{l+i}{2})$，那么这个回文串的长度是 $2(x-l)$，而由于 $x<\frac{l+i}{2}$，所以长度 $2(x-l)$ 不超过最长回文前缀长度 $i-l+1$。

引理 3.3.2. S 的任意一个中心在 $[\frac{l+i}{2},\frac{j+r}{2}]$ 的回文子串完全在 $[l,r]$ 之间。

证明. 不妨假设有一个回文子串 $S[l\dots r]$ 中心在 $[\frac{l+i}{2},\frac{i+r}{2}]$，且 $\frac{L+R}{2}\le\frac{l+r}{2}$，且 $L<l$，那么 $S[l\dots L+R-l]$ 必然是回文前缀，且长度比 $S[l\dots r]$ 的最长回文前缀长，矛盾。

有了这两条性质，我们只需要求出 $S[l\dots r]$ 的最长回文前缀和最长回文后缀，用 Manacher 算法预处理出回文半径 $R[i]$，查询 $R[i]$ 对应区间最大值即可。

求最长回文前后缀可以离线回文树，然后倍增。或者树上线性并查集做到线性时间复杂度。

查询区间 $R[i]$ 最大值是经典的 RMQ 问题，可以直接使用 ST 表做到 $O(n\log n)$ 预处理 $O(1)$ 查询，也可以按照 $\log n$ 大小分块，然后外层使用 ST 表，内层状压单调栈做到 $O(n)$ 预处理，$O(1)$ 查询。

3.4 集训队互测 2022 回文

3.4.1 题目大意

给定一个只含小写字母的字符串 S，有 Q 次操作，操作有以下两种：

1. 给出两个整数 i 和 c：将 S_i 修改为 c（$1\le i\le|S|$，c 是小写字母）。
2. 给出两个整数 l 和 r：查询 $S[l...r]$ 有多少回文后缀（$1\le l\le r\le|S|$）。

$1\le|S|\le2\times10^5,1\le Q\le2\times10^5$。

3.4.2 算法分析

我们建线段树，线段树的节点 $[l,r]$ 用等差数列的形式维护 $S[l\dots r]$ 的所有回文前缀和回文后缀。

关键的问题在于合并线段树的左孩子和右孩子的信息。如果能快速合并，那么操作 1 是线段树单点修改，操作 2 是线段树区间查询。

现在知道左孩子字符串 S，右孩子字符串 T。由于回文前缀和回文后缀的维护是几乎相同的，因此只需要讨论 ST 的回文后缀。

新串的回文后缀将会：

1. 不经过 S。
2. 经过 S，且中线在 T。
3. 经过 S，且中线在 S 和 T 的中线。
4. 经过 S，且中线在 S。

第一种情况，直接从 T 里继承即可。

第二种情况，发现这个新串回文后缀的中线肯定也是 T 的某个回文前缀的中线，也就是这种情况新串回文后缀肯定是从 T 中回文前缀扩展得到的。枚举 T 中的回文前缀，考虑现在有一个极长的回文串的等差数列：$A, AB, ABB, AB^3, \ldots, AB^k$。那么 $T = AB^kC$，C 是剩余部分，且 B 不是 C 的前缀。如果 C 是 B 的前缀，那么这个等差数列的一个后缀会对新串回文后缀贡献形成一个新的等差数列，这个等差数列的长度取决于 S，可以二分得到。否则 C 不是 B 的前缀，那么最多只有一项会成为新串的回文后缀。将 AB^k 往左扩展，扩展成 $B'^{k'}AB^k$ 的形式，这里 B' 是 B 的反串，k' 是尽可能大的向左扩展次数，此时只要判断以 $B'^{k'}AB^k$ 为回文中心，右端点为新串右端点的串是否是回文后缀，如果是，加入新串回文后缀中。

第三种情况最多只会新增一个回文后缀，直接判断有无新增即可。

第四种情况与末尾加字符相似，新串的回文后缀会从 S 的回文后缀中扩展得到。原先是头尾增加单个字符 c，现在是头加 T 的反串，尾加字符串 T，解决思路基本相同。

合并过程中需要快速判断 S 的两个子串是否相同，可以使用分块维护字符串 Hash 做到 $O(\sqrt{|S|})$ 修改，$O(1)$ 查询。

合并的时间复杂度是 $O(\log(|s| + |t|))$，因为当前等差数列会被完全包含在下一个等差数列的 A 中，且 B 的长度比 A 大，因此 k 之和是 $O(\log(|S| + |T|))$ 的，而第二种情况中 k' 是不会超过 k 的，因为中线在 t，k' 比 k 大中线不可能在 T。因此二分 k' 的复杂度也是 $O(\log(|s| + |t|))$ 的。

算上线段树维护的复杂度，总的复杂度是 $O(n \log n + Q(\log^2 n + \sqrt{n}))$。可以通过这道题目。

3.4.3 总结

对于带修的回文串查询问题，可以尝试使用线段树配合等差数列维护回文前后缀解决。这种可以合并的性质是普通回文树没有的。

感谢

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彭思进、杨耀良的指导。
感谢父母对我的培养和教育。
感谢学校的栽培，李建老师的教导和同学们的帮助。
感谢王相文同学与我交流讨论、给我启发。
感谢王相文同学同学为本文审稿。
感谢其它所有本文所参考过的资料的提供者。
感谢各位百忙之中抽出宝贵的时间来阅读本文。

参考文献

[1] oi-wiki，《回文树》，https://oi-wiki.org/string/pam/。

[2] 翁文涛，2017 年 IOI 国家候选队论文集，回文树及其应用

[3] 陈孙立，2019 年 IOI 国家候选队论文集，子串周期查询问题的相关算法及其应用

[4] Kazuki Mitani,Takuya Mieno,Kazuhisa Seto,and Takashi Horiyama，arXiv:2210.02000v2 [cs.DS] 6 Oct 2022， Internal Longest Palindrome Queries in Optimal Time

浅谈有限域在 OI 中的一些应用

威海实验高级中学 戚朗瑞

摘 要

本文主要介绍了有限域，并介绍了特征为 2 的有限域在 OI 中下列问题的一些应用：

- 最长简单路径问题
- Graph Motif 问题

1 有限域相关基本概念

1.1 域

定义 1. 域：域可以定义为一个含有加法和乘法两种操作的集合 F，要求含有以下性质：

- 封闭性：任意两个域 F 中的元素进行加法 *(+)* 或乘法 *(·)*，得到的结果仍在 F 中。
- 交换律：$a+b=b+a, a\cdot b=b\cdot a$。
- 结合律：$(a+b)+c=a+(b+c), (a\cdot b)\cdot c=a\cdot(b\cdot c)$。
- 乘对加分配率：$a\cdot(b+c)=a\cdot b+a\cdot c$。
- 存在加法单位元 *(0)* , $a+0=a$。
- 存在乘法单位元 *(1)* , $a\cdot 1=a$。
- 所有元素存在加法逆元，$\forall a\in F, \exists b\in F, a+b=0$。
- 所有非零元素存在乘法逆元，$\forall a\in F\setminus\{0\}, \exists b\in F, a\cdot b=1$。

1.2 有限域

定义 2. 有限域：若一个域中元素个数（称为域的阶）有限，则称该域为有限域。

有限域的元素个数（阶数）必为 p^n，其中 p 为质数。我们可以把这个有限域记为 $\mathbb{F}_{p^n}$，把 p 记为这个有限域的特征。

显然，有限域满足域的所有性质。

1.3 有限域 $\mathbb{F}_p$ 中的四则运算

简单来说，这部分的运算和常规运算很相似，只是需要把结果对 p 取模。

特别的，对于除法，则等价于乘上除数在模 p 意义下的逆元。

(一个数 a 在模 p 意义下的逆元 a^{-1} 满足 $a \cdot a^{-1} \equiv 1 \pmod{p}$)

1.4 有限域 $\mathbb{F}_{p^n}$ 中的四则运算

我们可以将 $\mathbb{F}_{p^n}$ 中的元素视为小于 n 次的 $\mathbb{F}_p$ 上的多项式。

我们首先选取一个 $\mathbb{F}_p$ 上的次数为 n 的不可约多项式 (指不可在 $\mathbb{F}_p$ 中分解为两个非常数多项式的乘积) 作为模数。记这个多项式为 R 。

加法、减法两个操作只需要最终把每一项对 p 取模即可。

对于乘法，我们最终还需要将结果的多项式对多项式 R 取模。

对于除法，则等价于乘上除数在模 R 意义下的逆元。

(一个多项式 A 在模 R 意义下的逆元 A^{-1} 满足 $A \cdot A^{-1} \equiv 1 \pmod{R}$)

2 特征为 2 的有限域

当 $p = 2$ 时，还有一个重要的性质：对于任意元素 a ，$a + a = 0$ 。

利用这个性质，对于很多题目中“不能有重复元素”的要求，我们可以尝试通过构造一些特征为 2 的有限域多项式，利用重复元素使具有重复元素的部分贡献为 0 。

这样，我们可以根据题意构造一个 $\mathbb{F}_{2^n}$ 中的多项式，并通过判断是否为 0 来解决一些判定性问题。

另外，$\mathbb{F}_{2^n}$ 中的很多操作可以通过位运算来快速完成。当 n 较小时，其中加法、减法都等同于二进制异或，乘法也可以通过若干异或及左右移运算在 $O(n)$ 内完成。

下文将给出两个较为典型的例题，来阐述特征为 2 的有限域在 OI 中的应用。

注：下文中部分“特征为 2 的有限域多项式”将简写为“多项式”。

3 最长简单路径问题

3.1 问题描述

给定一张简单图（有向或无向均可），你需要找出一条最长路径，使得该路径中不重复经过某一相同点。这里保证答案 k 远小于图中点数 n 。

3.2 初步分析

一种比较简单的思路是，使用状态压缩动态规划等方法记录经过的所有点集合等信息，做到不重复经过同一点。

但这样做的复杂度为 $O(\binom{n}{k} \cdot poly(n))$ 。(这里用 $poly(n)$ 表示关于 n 的多项式复杂度，即 $n^{O(1)}$ 。)

显然，由于该问题可规约到哈密顿路径，它属于 NP-Complete 问题。我们无法完全做到多项式复杂度。但由于 k 远小于 n ，我们能否将复杂度做到 $O(2^k \cdot poly(n))$ ？

下一步，我们可以从小往大枚举 k ，并将问题转化为确定是否存在一条长度为 k 的简单路径。

由于 $\sum_{i=1}^{k} O(2^i) = O(2^k)$ ，这步转化不会改变复杂度。

3.3 构造多项式

现在，问题转化成了判定是否存在长度为 k 的简单路径。

我们希望构造一个多项式，满足当且仅当图中存在一条长度为 k 的简单路径，多项式（以极大概率）不为 0 ，否则多项式为 0 。

如何构造这一多项式？

首先，我们希望不同路径间互不冲突，则我们先给一条路径所对应的多项式构造为 $\prod_{i=1}^{k-1} X_{v_i,v_{i+1}}$ ，其中 $v_1, v_2, \ldots, v_k$ 表示路径所经过的 k 个点，X 为一个预先设定的 $n \times n$ 的矩阵，其中每个元素均在 $\mathbb{F}_{2^v}$ 的所有元素中均匀随机。

但此时存在一个问题：我们无法保证路径不包含重复点。因此我们必须再将多项式乘上一些东西来去掉含有重复点的路径。

再次回忆特征为 2 的有限域的性质：$a + a = 0$ 。那么我们可以尝试使经过重复点的路径的所有贡献出现偶数次，抵消为 0 。

我们再设定一个 $n \times k$ 的矩阵 Y ，每个元素也均在 $\mathbb{F}_{2^v}$ 的所有元素中均匀随机。

我们将每一条路径对应的多项式乘上：

$$\sum_{p \in permutation(k)} \prod_{i=1}^{k} Y_{v_i,p_i}$$

其中 $permutation(k)$ 表示所有 1 到 k 的排列的集合。

我们观察上面这个式子，可以发现当 v_i 互不相同时，枚举的所有排列后面的乘积都互不相同。上面式子的值（以极大概率）不为 0 。

而如果路径中存在重复点，即存在两个数 i, j ，满足 $i < j, v_i = v_j$ 。

那么我们可以对于一个排列 P ，交换 P_i, P_j 两项，得到一个排列 Q 。

显然有

$$\prod_{i=1}^{k} Y_{v_i,P_i} = \prod_{i=1}^{k} Y_{v_i,Q_i}$$

也就是说，

$$\prod_{i=1}^{k} Y_{v_i,P_i} + \prod_{i=1}^{k} Y_{v_i,Q_i} = 0$$

如果对 Q 做上述操作，那么我们又能得回 P 。这样的话，所有的 $P \leftrightarrow Q$ 构成了一个双射。

因此，对于一个存在重复点的路径，其对答案的贡献恰好为 0 。我们也就做到了去除含有重复点的路径。

最终构造的多项式为：

$$\prod_{i=1}^{k-1} X_{v_i,v_{i+1}} \sum_{p \in permutation(k)} \prod_{i=1}^{k} Y_{v_i,p_i}$$

我们将所有路径对应的多项式相加，并判断和是否为 0 即可。

3.4 设计算法求多项式

我们希望求出

$$\sum_{v_{1,2,\ldots,k} \in Path(G)} \prod_{i=1}^{k-1} X_{v_i,v_{i+1}} \sum_{p \in permutation(k)} \prod_{i=1}^{k} Y_{v_i,p_i}$$

接下来将给出若干个解法。

3.4.1 状态压缩动态规划

一个可行方案是，使用状态压缩动态规划，记录 $f(pos, S)$ 表示所有 (起点为 pos，并且排列 p 中已经填了 S 集合中的所有数) 的贡献之和。

转移也很简单，直接枚举从起点走的第一条边以及在 p 中填的值即可。

dp 部分转移操作复杂度为 $O(2^k km)$ ， m 为图的边数。

如果再算上计算多项式乘法的话，在使用位运算优化的情况下，总时间复杂度为 $O(2^k kmv)$ 。(有限域多项式阶数为 2^v)

该算法空间复杂度为 $O(2^k n)$ 。

3.4.2 容斥原理转化式子

如果想得到多项式空间复杂度，可以采取另一种方法：容斥。

我们观察上述式子的这部分：

$$\sum_{p\in permutation(k)}\prod_{i=1}^{k}Y_{v_i,p_i}$$

比较棘手的是要求 p 是排列的这部分，即要求 p 每种元素都出现了一次。

我们尝试容斥，每次钦定若干个元素没出现，得到下列等式：

$$\sum_{S\subseteq\{1,2,\ldots,k\}}(-1)^{k-|S|}\sum_{p|\forall i\in 1,2,\ldots,n,p_i\in S}\prod_{i=1}^{k}Y_{v_i,p_i}$$

又由于特征为 2 的有限域的性质，$-a=a$，我们可以直接去掉 $(-1)^{k-|S|}$ 这部分。

$$\sum_{S\subseteq\{1,2,\ldots,k\}}\sum_{p|\forall i\in 1,2,\ldots,n,p_i\in S}\prod_{i=1}^{k}Y_{v_i,p_i}$$

再给式子变形一下:

$$\sum_{S\subseteq\{1,2,\ldots,k\}}\prod_{i=1}^{k}\sum_{j\in S}Y_{v_i,j}$$

这样，我们只需要对每个点 i 先算出 $\sum_{j\in S}Y_{i,j}$。记 f(pos,l) 表示当前在点 pos，要走 k 步的所有方案贡献总和。直接 $O(mkv)$ dp 即可。

时间复杂度和状压 dp 相同，但空间复杂度是关于 n 的多项式级别的。

3.5 算法正确性

上文提到的算法中，如果不存在长度为 k 的简单路径，那么多项式肯定等于 0。而如果存在，则“极大概率”不为 0。那这种算法的正确率到底是多少？

我们首先引入一个引理:

引理 1 (Schwartz-Zippel 引理)**.** 令 $P(x_1,x_2,\ldots,x_n)$ 为一在域 $\mathbb{F}$ 中关于 $x_1,x_2,\ldots,x_n$ 的，总度数为 d $(d\geq 0)$ 的非零多项式。

令 S 为 F 的一个非空有限子集，然后从 S 中独立随机选取 n 个元素 $r_1,r_2,\ldots,r_n$。

那么有结论: $P(r_1,r_2,\ldots,r_n)=0$ 的概率 $\leq\frac{d}{|S|}$。

证明由于和本文主题无关，这里略去。

回到这个问题中，构造的多项式为：

$$\sum_{v_{1,2,\ldots,k}\in Path(G)}\prod_{i=1}^{k-1}X_{v_i,v_{i+1}}\sum_{p\in permutation(k)}\prod_{i=1}^{k}Y_{v_i,p_i}$$

如果问题有解，其次数为 $2k-1$ 。

而域的大小为 2^v ，X,Y 中的元素都在整个域中随机选取。那么我们也能得到 $|S|=2^v$ 。

那么该算法出错的概率不超过：$\frac{2k-1}{2^v}$ 。

当 2^v 远大于 k 时，在 OI 内可基本忽略其出错概率。

3.6 输出方案

下面介绍一个输出可行方案的简单方法：

我们发现，上述算法可以求出以每个点为起点的多项式。

那么我们只需要每次选取一个合法的多项式非零的点作为起点，然后删掉这个点，并将 k 减小 1 。并将下一轮合法的点集标记为该点所有出边指向的点。

最终我们将 k 轮所选取的点拼接起来就是一个合法路径。

时间复杂度也并未改变，由于每轮 k 都减小 1 ，2^k 会除 2 。

3.7 进一步优化

在无向图中，上述算法还可以优化至 $O(1.66^k poly(n))$ 的复杂度。

有兴趣的话可以查阅 参考文献 [1] 进一步了解，这里不再展开。

4 Maximum Graph Motif 问题

4.1 问题描述

给定一张无向图 $G=(V,E)$ ，其中每个点都染有一种颜色。

给定一个颜色的可重集合 M ，设在 M 中颜色 i 出现了 m_i 次。

你需要选取 V 的一个子集，满足以下条件：

- 构造一张新图 G' ，其中 G' 的点集为选取的子集，边为 E 内两端点都在选取的点集中的所有边。需要保证 G' 为连通图。
- 记选取的点集中颜色为 i 的点出现了 c_i 次。需要保证 $c_i \le m_i$ 。

你需要求出在满足以上两个条件中的所有点集内，所含点数最多的含有多少个点。

保证 $|M|$ 远小于 n 。(换句话说，给出的算法指数上只能含有 $|M|$ 。)

4.2 初步分析

显然，我们仍可以把原问题转化为一个判定性问题，即对于所有 k，判断是否存在一个 k 个点的子图合法。

受到上面一个问题启发，我们仍希望能构造一个多项式，使得当且仅当多项式不为 0，该判定性问题有解。

我们做一步转化。一个点集的导出子图连通等价于可以找出一棵树，恰好覆盖点集中的每一个点，且边都在原图中出现过。

因此，我们可以将问题为尝试找到一棵满足题中第二个条件的树。

为了方便起见，我们令树均为有根树。

我们从大往小枚举 $|T_V|$，并确定是否能找出一个大小为 $|T_V|$ 的树 T。

4.3 构造多项式

首先，我们还是希望不同的有根树间互不影响，所以我们还是把每条边 $u \to v$ 对应到一个多项式 $X_{u,v}$。

X 为一 $n \times n$ 的矩阵，其中每个元素均在 $\mathbb{F}_{2^v}$ 中均匀随机。

那么我们首先将一棵有根树的多项式设为:

$$\prod_{e=(u,v)\in T_E} X_{u,v}$$

其中，T_E 表示选取的有根树的所有边，$e = (u, v)$ 表示从有根树中选取的一条 $u \to v$ 的有向边。

接下来我们考虑如何处理第二个限制条件。

我们发现，特征为 2 的有限域多项式不太容易处理 $c_i \le m_i$ 这种条件。我们不妨转化一下，变为再给每一个有根树中颜色为 i 的点打一个 1 到 m_i 的编号，要求颜色相同的点编号两两互不相同。

我们记第 i 个点选的编号为 a_i。

对于编号，我们可以再建立一个矩阵 Y，$Y_{pos,id}$ 表示点 pos 选取的编号为 id 的贡献。Y 中的元素也均在 $\mathbb{F}_{2^v}$ 中均匀随机。

至此，有根树的贡献为:

$$\prod_{e=(u,v)\in E_T} X_{u,v} \sum_{a} \prod_{i\in T_V} Y_{i,a_i}$$

其中 $\sum_a$ 表示枚举所有编号方案，T_V 表示有根树的点集。

下一步，我们还需要限制颜色相同的点选的编号互不相同。

我们回想最长简单路径中限制点互不相同的方法，尝试是否能类似地使用在本题中。

我们再将每个有根树上的点写上一个 1 到 $|T_V|$ 中的数字，满足其构成一个排列，记点 i 上的数字为 p_i 将贡献乘上:

$$\sum_{p}\prod_{i\in T_V} Z_{(col_i,a_i),p_i}$$

其中 col_i 表示点 i 的颜色。Z 为一个第一维下标为一个 (颜色，编号) 的二元组，第二维下标为一个整数。元素也均在 $\mathbb{F}_{2^v}$ 中均匀随机。

显然，如果有两个点颜色和编号均相同，根据特征为 2 的有限域性质，上式为 0，否则以极大概率不为 0。

最后整理一下，构造出对于一棵固定的有根树的整个多项式为:

$$\prod_{e=(u,v)\in E_T} X_{u,v} \sum_{a}\prod_{i\in T_V} Y_{i,a_i} \sum_{p}\prod_{i\in T_V} Z_{(col_i,a_i),p_i}$$

4.4 求多项式的值

仍然考虑容斥:

$$\begin{aligned}&\prod_{e=(u,v)\in E_T} X_{u,v} \sum_{a}\prod_{i\in T_V} Y_{i,a_i} \sum_{p}\prod_{i\in T_V} Z_{(col_i,a_i),p_i}\\ =&\prod_{e=(u,v)\in E_T} X_{u,v} \sum_{a}\prod_{i\in T_V} Y_{i,a_i} \sum_{S\subseteq\{1,2,\ldots,|T_V|\}}\sum_{p,p_i\in S}\prod_{i\in T_V} Z_{(col_i,a_i),p_i}\end{aligned}$$

原理和最长简单路径问题中的容斥部分相同，这里不再展开。

我们首先枚举 S，并固定。

整理一下剩余部分:

$$\begin{aligned}&\prod_{e=(u,v)\in E_T} X_{u,v} \sum_{a}\prod_{i\in T_V} Y_{i,a_i} \sum_{p,p_i\in S}\prod_{i\in T_V} Z_{(col_i,a_i),p_i}\\ =&\prod_{e=(u,v)\in E_T} X_{u,v} \prod_{i\in T_V}(\sum_{a_i=1}^{m_{col_i}} Y_{i,a_i} \sum_{p_i\in S} Z_{(col_i,a_i),p_i})\end{aligned}$$

括号内的式子可以预处理出来，记 i 的贡献为 W_i。

$$=\prod_{e=(u,v)\in E_T} X_{u,v}\prod_{i\in T_V} W_i$$

我们尝试使用动态规划求出结果。记录 $dp_{pos,size}$ 表示考虑根节点为 pos，节点个数为 $size$ 的所有有根树中，上述式子之和。

转移的话，可以直接枚举 pos 的所有出边，并考虑将这条边指向的点接入到 pos 内作为子树，并枚举这个子树的节点个数。

可能有一个疑问：上面的转移并没有限制接入中的子树内不含有接入子树前的原有根树中已有的点，是否可能会错误地把这部分不合法情况计入答案？

但仔细分析一下，这种情况是不会对答案产生贡献的。假如最终产生的“树”中含有两个相同的节点。那么我们发现交换这两个节点的编号后，产生的贡献是相等的。它们可以一一抵消，最终总贡献为 0 。

这样，我们直接用上述方法，就能算出固定 S 时的贡献。最后把所有贡献相加，并找出最大的一个不为 0 的多项式对应的点数即可。

最终总复杂度为 $O(2^{|M|}|M|^2mv)$ 。

4.5 算法正确性

该问题构造的多项式和上一问题形式类似，也可以用相同方法证明当 2^v 远大于 $|M|$ 时出错概率几乎为 0 。

由于方法和上一问题类似，具体细节就不加以展开了。

4.6 输出方案

我们考虑怎么输出该问题的一个合法选取点集方案。

首先，我们先求出答案大小，设其为 k ，并找到一个合法根节点，设其为 $root$。

接下来若干轮，我们另外记录在每一次尝试加入边后，对于所有枚举的 S 的 $dp_{root,k}$ 之和。记尝试加入第 i 条边后的值为 x_i 。(我们令 $x_0 = 0$)

我们找到一个最小的 i ，满足 $x_{i-1} = 0 \wedge x_i \neq 0$ 。

这样的话，显然存在一个以 $root$ 为根的合法方案，包含第 i 条边指向的点。

那么我们可以将该点和根节点缩起来 (缩成的点保留根节点的颜色，连边为原先两点连边的并集)，并将 $|M|$ 中删掉一个该点的颜色，然后将 k 减一。

这样，我们可以递归下去，直到 $k = 1$ 为止。

该算法时间复杂度和仅求答案相同，为 $O(2^{|M|}|M|^2mv)$ 。

5 总结

本文主要简单介绍了有限域的相关性质，并讲解了一些特征为 2 的有限域在 OI 中应用的例子。

但有限域的应用，特别是对于一些 NP-Complete 问题的优秀复杂度算法，远不止本文介绍的这些。希望能通过本文让更多的同学了解有限域相关内容，并更深入地探究相关问题。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队杨耀良教练的指导。
感谢唐绍轩同学、俞越同学的启发及为本文审稿。
感谢山东省学会及威海实验高中教练们的教导。
感谢家人、朋友对我的支持与鼓励。

参考文献

[1] Łukasz Kowalik, Algebraic techniques in parameterized algorithms - Part II: Polynomials over finite fields of characteristic two, 2014

[2] Andreas Björklund, Petteri Kaski, Łukasz Kowalik, Probably Optimal Graph Motifs, 2013

[3] 李超, 阮威, 张翔, 张龙, 计算机代数系统的数学原理, 2009

[4] WikiPedia, 有限域算术, (https://zh.m.wikipedia.org/zh-cn/有限域算术)

[5] WikiPedia, Schwartz-Zippel lemma, (https://en.wikipedia.org/wiki/Schwartz-Zippel_lemma)

浅谈一类树上统计相关问题

浙江省诸暨中学 朱羿恺

摘 要

本文根据笔者为 IOI2023 中国国家集训队作业命制的一道试题，引出了一类有关树上统计相关的问题，介绍了一种在该问题上具有一定通用效果的算法，之后又给出了几道例题，并依次采用了本文中提到的算法解决了这些问题。

1 引言

树相关问题在各种算法竞赛中一直是一个考察的热门，而相关问题的种类也是纷繁复杂，本文对其中一种类型进行了介绍，并给出了例题和讲解，希望能对读者有所启发，在遇到相关问题时能更加得心应手。

如果没有特殊说明，在接下来的问题中，我们总是认为树的规模和询问次数（如果存在多次询问）的大小为 $O(n)$。

2 预备知识

为方便文章讨论与读者阅读，本节中将给出本文中用到的一些相关知识，已经具备相关知识的同学可以跳过对应的部分。

2.1 重链剖分

树链剖分用于将树分割成若干条链的形式，以维护树上路径的信息。而重链剖分则是树链剖分的一种形式。

先给出一些定义：

定义 2.1.1 (重儿子)**.** 一个节点的子节点中子树最大的子结点为重儿子。如果有多个子树最大的子结点，任取其一。如果没有子节点，就无重子节点。

定义 2.1.2 (轻儿子)**.** 一个节点的子节点中非重儿子的子节点都为轻儿子。

定义 2.1.3 (重边)**.** 一个节点和其重儿子之间的边为重边。

定义 2.1.4 (轻边)**.** 一个节点和其轻儿子之间的边为轻边。

定义 2.1.5 (重链)**.** 若干条首尾衔接的重边构成重链。

把落单的结点也当作重链，那么整棵树就被剖分成若干条重链。
如图：

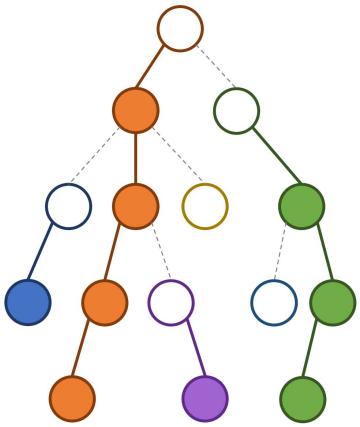


图 1: 虚线表示轻边，实线表示重边。每条重链的链头用空心表示。

2.2 重链剖分的性质

性质 2.2.1. 树上任意一条路径可以拆分成 $O(\log n)$ 条重链。

证明. 可以发现，当我们向下经过一条轻边时，所在子树的大小至少会除以二。

因此，对于树上的任意一条路径，把它拆分成从两个端点的最近公共祖先分别向两边往下跳重链，分别最多跳 $O(\log n)$ 次就会到达两端，因此，树上的每条路径都可以被拆分成不超过 $O(\log n)$ 条重链。 □

性质 2.2.2. 所有点的所有轻儿子的子树大小之和为 $O(n \log n)$ 级别。

证明. 每个点的贡献为其到根的路径的轻边数，由上一条性质，这个数量为 $O(\log n)$ 级别，对 n 个点求和，自然是 $O(n \log n)$。 □

3 试题

3.1 题目描述

给出一棵大小为 n 的树。

定义 $x \to y$ 表示树上从 x 到 y 的简单路径，树上 x, y 两点的距离 $\mathrm{dis}(x, y)$ 为 $x \to y$ 包含的边数。

定义树上一个点 u 到树上一条简单路径 $x \to y$ 的距离为 $\min\limits_{v \in x \to y} \mathrm{dis}(u, v)$。

有 q 次询问，第 i 次询问给出三个整数 x_i, y_i, d_i，满足 $1 \le x_i, y_i \le n, 0 \le d_i < n$，你需要求出距离 $x_i \to y_i$ 小于等于 d_i 的点数。

3.2 数据范围与限制

对于全部的数据满足，$1 \le n, q \le 2 \times 10^5$。

时间限制：3 s

空间限制：1024 MB

4 试题分析

4.1 初步转化

我们假设以 1 为根，令 z_i 为 x_i 和 y_i 的最近公共组先，考虑把问题拆成 $z_i \to x_i$ 和 $z_i \to y_i$ 两个部分。

我们先求出距离 z_i 小于等于 d_i 的点数，这是一个经典的问题，可以通过点分治在 $O(n \log n)$ 的时间复杂度内解决。

注意到此时 $x_i \to z_i$ 还需要的统计的点为 $\sum_{u \in z_i' \to x_i} f_{u,d_i}$，其中 z_i' 表示 $z_i \to x_i$ 上的第二个点的编号，$f_{u,i}$ 表示 u 子树中距离 u 为 i 的点数。$z_i \to y_i$ 也是同理。

于是我们现在的问题在于如何快速计算 $\sum_{u \in z_i' \to x_i} f_{u,d_i}$。一个经典的处理是把问题进行差分，转为计算 $\sum_{u \in 1 \to x_i} f_{u,d_i} - \sum_{u \in 1 \to z_i} f_{u,d_i}$。

4.2 一个稍劣的解法

现在的问题在于快速计算 $\sum_{u \in 1 \to x} f_{u,d_i}$。为了方便，我们转而计算距离 $1 \to x$ 小于等于 d 的点数，不难发现这样转化后答案依然是正确的（因为有差分，多出来的部分会被减掉）。

我们先算下 $1 \to x$ 的大小，接下来只要考虑 $1 \to x$ 上每个点其他子树对询问的贡献即可。

考虑对树重链剖分，先算一下每个点的重儿子子树对于其他子树内的询问的贡献，这个部分就是 $O(n\log n)$ 次询问一个子树内，距离子树的根小于等于 d 的节点个数，可以使用长链剖分在 $O(n\log n)$ 的时间复杂度内解决。

考虑下每个询问还有哪些东西没算。

用语言描述的话就是 x 到根上的每个点 u 还需要计算的子树就是除了重儿子子树和包含 x 的子树，即下图中蓝色节点的子树，而第一次计算的则是黄色节点的子树。

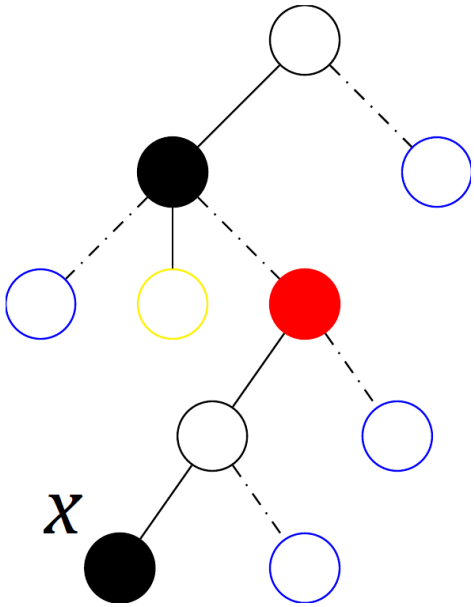


图 2: 虚线表示轻边，实线表示重边。

注意到蓝色节点（即所有点的轻儿子构成的集合）的子树的大小之和为 $O(n\log n)$ 级别。

我们把 x 这个询问在每个右图涂黑的节点（即从 x 往上跳，如果是从重儿子跳上来的，则说明该节点是涂黑的节点）里塞一份，然后对每条重链做就行，这依然是一个二维数点问题。注意最后还需要减去图上涂红节点子树对于询问的重复贡献。

时间复杂度是 $O(n\log^2 n)$ 的，已经可以通过此题。

4.3 最终解法

可以发现我们的算法复杂度瓶颈在于计算轻儿子子树对于答案的贡献。

接下来直接给出做法。

我们对于一条重链从上往下扫描，依次加入每个点的所有轻子树，维护一个前缀和数组，以轻儿子深度的时间代价更新轻儿子深度那么多的前缀和数组。具体来说，就是给前缀和数组的第 i 位加上当前加入的轻儿子子树中深度小于等于 i 的点数。

对于询问，扫到其时就直接在前缀和数组上 $O(1)$ 查询，这样做时间复杂度是 $O(n\log n)$，但是容易发现，我们少计算了一些东西。

稍加分析不难得到，假设当前询问的 d_i 为 d，那么少算的部分就是已经被扫到的深度小于 d 的轻子树的大小之和。

对于这一部分，考虑将这些子树按照深度分组，具体来说，把深度在 $[2^i, 2^{i+1})$ 之中的归到第 i 组。

对于组内和整组分别维护一个前缀和数组，对于整组的前缀和数组，每次加入一个轻子树更新的时间代价为 $O(\log n)$，而每个点只会加入一次，所以这个部分的时间复杂度为 $O(n\log n)$；对于组内的前缀和数组，加入第 i 组更新的时间代价为 $O(2^i)$，即对于第 i 组，假设加入的总大小为 A，更新的时间代价不会超过 $\frac{A\times 2^i}{2^i} = A$，而 A 之和是 $O(n\log n)$ 级别的，所以这个部分的时间复杂度也为 $O(n\log n)$。

对于查询来说，直接 $O(1)$ 查询两个前缀和数组即可。

综上所述，本题即可在 $O(n\log n)$ 的时间复杂度内解决。

5 试题总结

经过一些转化，原问题转化为了计算所有点和询问给出的一个点满足相关的一些限制的个数，而这个限制又和两者的最近公共祖先有关。

而对于这类问题，我们往往可以分成两个部分来计算。

- 对树重链剖分后，当前询问的点到根的路径上的点的重儿子子树。
- 对树重链剖分后，当前询问的点到根的路径上的点的轻儿子子树，当然还需要减去询问给出的点所在的一些轻子树的贡献。

如果修改和询问都是 $O(1)$ 的，那么该算法的复杂度则为 $O(n\log n)$ 的。

6 例题

6.1 例题 1

CodeForces1098F Ж-function [1]

[1]题目来源：http://codeforces.com/problemset/problem/1098/F

6.1.1 题目大意

给出一个字符串 S，定义 $S[l\dots r]$ 为由 S 的第 l 个到第 r 个字符组成的子串。

再定义 $F(S)=\sum_{i=1}^{|S|}\mathrm{lcp}(S,S[i\dots|S|])$，其中 $\mathrm{lcp}(S,T)$ 为字符串 S 和字符串 T 最长公共前缀的长度。

给出一个长度为 n 的字符串 T，有 q 次询问，每次询问给出两个正整数 l,r，表示询问 $F(T[l\dots r])$ 的值。

6.1.2 数据范围与限制

对于全部的数据满足，$1\le n,q\le 2\times 10^5$。

时间限制：6 s

空间限制：512 MB

6.1.3 题目解法

先对 T 的反串建后缀树。

后缀 i,j 的 lcp 即为后缀树上 i,j 两点最近公共祖先对应的长度，于是我们可以得到 $\mathrm{lcp}(T[l\dots r],T[i\dots r])=\min(r-i+1,len_{\mathrm{lca}(i,l)})$。

这与上文提到的问题类型非常相像，考虑套用上文提到的算法。

那需要考虑两部分的贡献：

- 第一部分是每个点重儿子子树对于其他轻儿子子树内的询问的贡献。这个部分考虑分类讨论（假设当前点代表的长度为 len）：
  - $i\in[l,r-len)$，此时对答案的贡献为 len，统计子树内 $[l,r-len)$ 内的个数即可。
  - $i\in[r-len,r]$，此时对答案的贡献为 $r-i+1$，统计子树内 $[r-len,r]$ 内的和与个数即可。

  两个部分都是简单的二维数点问题，可以用线段树合并解决，这一部分的时间复杂度为 $O(n\log^2 n)$。
- 第二部分是每个点轻儿子子树对于重儿子子树内的询问的贡献。类似上一部分的考虑，这部分可以转化为三维偏序，一维是要在区间 $[l,r]$ 内，一维是在重链上的深度，还有一维则是 lcp 需要考虑的取 min。

  对于三维偏序问题，我们对按一维排序，另外两维使用 cdq 分治套树状数组即可，该部分的时间复杂度为 $O(n\log^3 n)$，常数很小。

当然本题还可以通过把树链剖分换成全局平衡二叉树来去掉第二个部分的一个 log，不过实际表现却不如该算法，因此此处略去不讲。

6.2 例题 2

LuoguP4482 [BJWC2018]Border 的四种求法 [2]

6.2.1 题目大意

给出一个长度为 n 的字符串 S，有 q 次询问，每次询问给出两个正整数 l, r，表示询问 $S[l \dots r]$ 的 border。

定义一个串 S 的 border 为最大的 i，满足 $i < |S|$，并且 $S[1 \dots i] = S[|S| - i + 1 \dots |S|]$。

6.2.2 数据范围与限制

对于全部的数据满足，$1 \le n, q \le 2 \times 10^5$。

时间限制：5 s

空间限制：512 MB

6.2.3 题目解法

先对 S 建后缀树。

问题等价于要找到最大的 i，满足 $i < r$ 且 $i - len_{\text{lca}(i,r)} + 1 \le l$。如果得到的结果小于 l，则表示不存在 border。

依旧考虑两部分的贡献：

- 第一部分相当于查询子树内在原串中位于 $[1, \min(r-1, l+len-1)]$ 的最大值，依旧可以线段树合并解决，时间复杂度 $O(n \log^2 n)$。
- 第二部分则是需要满足两个限制，分别是 $i < r$ 和 $i - len_{\text{lca}(i,r)} + 1 \le l$。考虑维护一棵线段树，每次把 $i - len_{\text{lca}(i,r)} + 1$ 插入到第 i 个位置，线段树上每个节点则维护当前区间 $i - len_{\text{lca}(i,r)} + 1$ 的最小值，这样询问的时候在线段树上二分即可。注意该题的信息不可减，需要特殊处理一下。

  该部分的时间复杂度依旧是 $O(n \log^2 n)$。

7 总结

通过上述例题，可以体会到该算法在解决这类树上问题时的通用性。

希望本文能起到抛砖引玉的作用，对 OI 中相关的树上的问题产生更多的启发。

[2]题目来源：https://www.luogu.com.cn/problem/P4482

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队杨耀良教练的指导。
感谢董烨华、陈合力、袁荣乐老师对我的培养与指导。
感谢父母对我的理解与支持。

参考文献

[1] OI Wiki，“树链剖分”.

[2] CodeForces Round 530 Tutorial

浅谈矩阵在信息学竞赛中的应用

四川省成都市第七中学 杨宁远

摘 要

矩阵在信息学中有着极为广泛的应用。本文从矩阵的基本概念和性质出发，针对 LGV 引理、特征多项式等问题给出了优秀的算法，并讨论了若干道与之相关的信息学竞赛例题。本文还介绍了余子式及其高效的计算方法，探讨了余子式在矩阵树定理中的应用。在最后，本文介绍了计算特征值数值解有关的方法，例如 Jacobi 算法和 QR 迭代法等。

1 引言

矩阵是线性代数的重要部分，与之相关的线性代数工具有行列式、余子式、特征多项式、特征值与特征向量等等。这些工具可以有效帮助我们解决信息学竞赛中的很多问题，为我们提供更为本质的视角。

笔者在本文中梳理和总结了矩阵在信息学中的应用，希望能借此机会，向选手们普及矩阵相关的知识和算法，吸引选手们进行更进一步的研究。笔者也希望在将来的算法领域里，能看到更多与线性代数有关的题目及其优美的解法。

本文的第 2 节为矩阵相关的记号约定；第 3 节介绍了行列式以及由此推导出来的 LGV 引理；第 4 节介绍了余子式、伴随矩阵及其求法和矩阵树定理；第 5 节介绍了矩阵的线性递推及其应用；第 6 节介绍了矩阵的特征值、特征多项式和相关的应用；第 7 节讨论了特征值的一些求解方式。

2 记号约定

2.1 矩阵的定义

一个由 $n \times m$ 个数排列为 n 行 m 列的数表被称为一个 n 行 m 列的矩阵，简称为一个 $n \times m$ 的矩阵。

对于一个 $n \times m$ 的矩阵 A，它可以写作：

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,m} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,m} \end{bmatrix}$$

其中矩阵第 i 行 j 列的元素被记为 $a_{i,j}$。

特殊地，若 $n = m$，那么我们可以称 A 为 n 阶矩阵或方阵。

2.2 矩阵的基本运算

定义 2.1 (加法)**.** 两个 $n \times m$ 的矩阵 A 与 B 的加法定义为其元素对位相加。令 $C = A + B$，那么有 $c_{i,j} = a_{i,j} + b_{i,j}$。

定义 2.2 (减法)**.** 两个 $n \times m$ 的矩阵 A 与 B 的减法定义为其元素对位相减。令 $C = A - B$，那么有 $c_{i,j} = a_{i,j} - b_{i,j}$。

定义 2.3 (乘法)**.** 一个 $n \times s$ 的矩阵 A 乘上另一个 $s \times m$ 的矩阵 B 会得到一个大小为 $n \times m$ 的矩阵 C，并且有：

$$c_{i,j} = \sum_{k=1}^{s} a_{i,k} \times b_{k,j}$$

定义 2.4 (数乘)**.** 一个 $n \times m$ 的矩阵 A 乘上另一个数 x 的结果 $B = xA$ 有*:*

$$b_{i,j} = a_{i,j} x$$

定义 2.5 (转置)**.** 对于一个 n 阶矩阵 A，其转置 A^T 被定义为将 A 中的元素按对角线翻转。也就是说如果 $B = A^T$，那么有 $b_{i,j} = a_{j,i}$。

3 行列式

3.1 行列式的定义与计算

定义 3.1 (行列式)**.** 对于一个 n 阶矩阵 A，其行列式被定义为：

$$|A| = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}$$

其中 S_n 为所有 n 阶排列，$sgn(\sigma)$ 为 $(-1)^{\sigma\text{的逆序对个数}}$。

显然，如果我们按照定义直接枚举排列计算行列式的复杂度是 $O(n!)$ 的。然而这种方法效率过于低下，大部分时候都不能满足我们的要求。

定义 3.2 (矩阵的初等变换)**.** 矩阵的初等变换定义为：

1. 将第 i 行加上第 j 行的 k 倍（$i \neq j$）
2. 将第 i 列加上第 j 行的 k 倍（$i \neq j$）
3. 将第 i 行乘上 k（$k \neq 0$）
4. 交换 i, j 两行。

定理 3.1. 对矩阵进行第 $1, 2$ 种初等变换不改变行列式的值，第 3 种会将行列式乘上 k，第 4 种会将行列式乘上 -1。

证明. 将行列式展开后容易发现第 $1, 2$ 种的贡献会被抵消掉，而第 3 种会使得所有排列的答案全部乘上 k。

对于第 4 种，我们发现如果交换排列中的两个数，那么逆序对数奇偶性会变化。 □

综上，我们可以通过矩阵的初等变换，利用解方程时常用的高斯消元法来求解行列式。这种方法的时间复杂度为 $O(n^3)$。

也可以把高斯消元的过程看作左乘行变换矩阵或者右乘列变换矩阵来得到上三角矩阵。即 $P_1P_2P_3 \cdots P_kA = A' = AQ_1Q_2Q_3 \cdots Q_k$。

3.2 行列式的一些性质

定理 3.2.

$$|A| = |A^T|$$

证明. 由于排列求逆后不改变逆序对个数，而转置后的行列式可以看作把原矩阵的排列求逆，于是转置后的行列式不变。 □

定理 3.3.

$$|AB| = |A||B|$$

证明. 令 A', B' 分别为 A, B 进行高斯消元后的上三角矩阵。

将 A, B 分别表示为 $P_1P_2P_3\cdots P_kA'$，$B'Q_1Q_2Q_3\cdots Q_k$，那么也就有：

$$|AB| = |P_1P_2P_3\cdots P_kA'B'Q_1Q_2Q_3\cdots Q_k| = |A'B'| = |A'||B'|$$

□

3.3 LGV 引理

定理 3.4 (LGV 引理)**.** 对于一张有向无环图 G，边有边权。考虑一个大小为 n 的起点集合和一个大小为 n 的终点集合，定义 $e(a, b)$ 为所有从 a 到 b 的路径的边权乘积之和。定义矩阵

$$M = \begin{bmatrix} e(a_1,b_1) & e(a_1,b_2) & e(a_1,b_3) & \cdots & e(a_1,b_n) \\ e(a_2,b_1) & e(a_2,b_2) & e(a_2,b_3) & \cdots & e(a_2,b_n) \\ e(a_3,b_1) & e(a_3,b_2) & e(a_3,b_3) & \cdots & e(a_3,b_n) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ e(a_n,b_1) & e(a_n,b_2) & e(a_n,b_3) & \cdots & e(a_n,b_n) \end{bmatrix}$$

定义一组不相交路径为 n 条路径 $P_1, P_2, \cdots P_n$，使得存在一个排列 σ，其中第 i 条路径为从 a_i 出发，到 b_{σ_i} 结束，且路径两两不存在公共点。这组路径的带符号权值被定义为

$$f(P_1, P_2, \cdots P_n) = sgn(\sigma)\prod_i e(i, \sigma_i)$$

那么有：

$$|M| = \sum_{P_1,P_2,\cdots,P_n} f(P_1, P_2, \cdots, P_n)$$

证明. 假如说一组路径有某两条存在交点，那么我们找到最小的 i，并在 i 最小的基础上找到最小的 j 使得 P_i, P_j 有交。那么我们找到它们最靠前相交的点，并将之后的路径交换一下，容易发现这会使得 $sgn(\sigma)$ 改变，而路径组的权值不变。将得到的新路径组与原来的路径组进行匹配。容易发现所有有交的路径组都能一一匹配，又由于路径的权值前面有一个 $sgn(\sigma)$，于是有交路径组的权值恰好能够两两抵消。

而对于不相交的路径，行列式中 $sgn(\sigma)$ 正好与路径权值中的相符。于是得证。 □

3.4 例题

给定 n, m, k, r, c, V，求出满足如下条件的 $n \times m$ 矩阵的数量模 998244353 的值：

1. 对所有 $1 \le i \le n, 1 \le j \le m$，有 $1 \le a_{i,j} \le k$
2. 对所有 $1 \le i \le n, 1 \le j \le m-1$，有 $a_{i,j} \le a_{i,j+1}$
3. 对所有 $1 \le i \le n-1, 1 \le j \le m$，有 $a_{i,j} \le a_{i+1,j}$
4. $a_{r,c} = V$

$n, m \le 200, k \le 100$。

3.5 问题分析

题目来源：XXI Open Cup, Grand Prix of Tokyo, Problem A Ascending Matrix。

首先发现题目相当于要把 A 分成恰好 k 层，层可以为空。并且还规定了 $a_{r,c}$ 属于哪一层。要求层的轮廓路径只能向右或向上走，且层的轮廓路径不能交叉。

再转化一下，变成：有 $k-1$ 条从 $(n, 0)$ 走到 $(0, m)$，要求只能向右和向上走的路径，路径之间有顺序，后面的路径不能走到前面的路径左边。并且 $a_{r,c}$ 左边要有 $v-1$ 条路径。

我们将第 i 条路径向右平移 $i-1$ 格，再向下平移 $i-1$ 格，现在限制就变成了路径不交了。而第 i 条路径的起点变成了 $n+i-1, i-1$，终点变成了 $i-1, m+i-1$。容易发现如果 σ 的逆序对数 >0，那么路径一定有交。因此如果不考虑 $a_{r,c}$，我们就可以直接使用 LGV 引理求出方案数。

现在我们需要保证 $a_{r,c}$ 左边恰有 $v-1$ 条路径。我们将一条路径的权值改写为：

$$\begin{cases} 1, & \text{路径不经过 } a_{r,c} \text{ 的左边} \\ x, & \text{路径经过 } a_{r,c} \text{ 的左边} \end{cases}$$

那么容易发现答案恰为 $[x^{v-1}]|A|$。而显然 $|A|$ 最高为一个 n 次多项式，因此带入 n 个点值后使用拉格朗日插值即可在 $O(n^3 k)$ 复杂度内解决。

4 余子式与伴随矩阵

4.1 余子式的定义和性质

定义 4.1 (余子式)**.** n 阶矩阵 A 中一个元素 $a_{i,j}$ 的余子式 $M_{i,j}$ 定义为 A 删掉第 i 行和第 j 列后的矩阵的行列式。

定义 4.2 (代数余子式)**.** n 阶矩阵 A 中一个元素 $a_{i,j}$ 的代数余子式 $A_{i,j}$ 定义为 $(-1)^{i+j}M_{i,j}$。

定理 4.1 (拉普拉斯按行展开)**.**

$$\forall i, |A| = \sum_j a_{i,j}A_{i,j}$$

证明. 我们枚举 $\sigma_i = j$，那么不难发现

$$\sum_{\sigma_i=j} sgn(\sigma)\prod_{i=1}^{n} a_{i,\sigma_i} = a_{i,j}A_{i,j}$$

□

定理 4.2 (拉普拉斯按列展开)**.**

$$\forall i, |A| = \sum_j a_{i,j}A_{i,j}$$

证明. 由于实际上对 $|A|$ 按列展开就是对 $|A^T|$ 按行展开，而又有 $|A| = |A^T|$，于是证毕。 □

定义 4.3 (伴随矩阵)**.** 矩阵 A 的伴随矩阵定义为：

$$A^* = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} & \cdots & A_{n,1} \\ A_{1,2} & A_{2,2} & A_{3,2} & \cdots & A_{n,2} \\ A_{1,3} & A_{2,3} & A_{3,3} & \cdots & A_{n,3} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ A_{1,n} & A_{2,n} & A_{3,n} & \cdots & A_{n,n} \end{bmatrix}$$

定理 4.3. 当 $|A| \neq 0$ 时，有 $AA^* = |A|I$

证明. 将式子展开后会发现其实就是拉普拉斯展开。 □

利用上面的性质我们可以在 $O(n^3)$ 的时间复杂度内求出伴随矩阵。

如果 $rank(A) = n$，那么会有 $A^* = |A|A^{-1}$。直接使用矩阵求逆就可以求出伴随矩阵。

如果 $rank(A) \leq n-2$，那么删掉一行一列后一定仍然线性相关，也就是说余子式一定为 0。

否则如果 $rank(A) = n-1$，我们找到一组不全为 0 的列向量 $\vec{p}$ 和行向量 $\vec{q}$，使得 $A\vec{p} = 0, \vec{q}A = 0$。那么对于 $q_r \neq 0, p_c \neq 0$，有 $A_{i,j} = \frac{q_i p_j}{q_r p_c}A_{r,c}$。

证明. 观察一下 (r,i) 的余子式以及 (r,c) 的余子式，我们发现可以利用初等变换将 $A_{r,c}$ 变为 $A_{r,i}$。不妨令 $i<c$，令 (r,c) 的剩余矩阵为 B，(r,i) 的为 C。那么我们将 B 的第 $i+1\sim c-1$ 列向左平移一格，将第 i 列交换到第 $c-1$ 列上。由于有 $\sum_i p_i A_i=0$，其中 A_i 是 A 的第 i 列，于是我们将 B 的第 $i-1$ 列乘上 p_i，将剩余的列按 p 的系数加到第 $i-1$ 列上，最后除以 $-p_c$ 就可以得到 C 的第 $i-1$ 列。于是我们就有 $A_{r,i}=\frac{p_i}{p_c}A_{r,c}$。

对行来说也有类似的结论。将两者结合起来即得证。 □

4.2 矩阵树定理

定理 4.4 (矩阵树定理)**.** 对于一个无向图 G，它的生成树个数为：度数矩阵减去邻接矩阵删去第 i 行 i 列的行列式。

定理 4.5 (矩阵树定理的有向形式)**.** 对于一个有向图 G，它的以 i 为根的外向生成树个数为：入度矩阵减去边矩阵删去第 i 行 i 列的行列式。

类似地，它的以 i 为根的内向生成树个数为：出度矩阵减去边矩阵删去第 i 行 i 列的行列式。

4.3 例题

给定一张大小为 n 的有向图。对于每个点 i 求出以 i 为根的外向树的个数。

$n\le 500$。

4.4 问题分析

题目来源：HDU 多校第 10 场 J 题。

这道题就是要求出所有的 $A_{i,i}$，直接用我们上面所讲的方法求出伴随矩阵即可。

复杂度：$O(n^3)$。

5 矩阵的线性递推

5.1 问题引入

给定一个 n 阶矩阵 A，求出 A^k。所有运算均在模 10^9+7 意义下进行。

数据范围：$n \le 100$，$k \le 10^{10000}$。

5.2 问题分析

题目来源：BZOJ 4162, shlw loves matrix II。

直接矩阵快速幂复杂度是 $O(n^3 \log k)$ 的。而在这道题当中，k 高达 10^{10000}，显然矩阵快速幂并不够快。

定义 5.1 (特征多项式)**.** 矩阵 A 的特征多项式定义为：

$$p_A(\lambda) = |A - \lambda I|$$

定理 5.1 (Cayley-Hamilton 定理)**.**

$$p_A(A) = a_0 + a_1 A + a_2 A^2 + \cdots + a_n A^n = 0$$

我们先求出 A 的特征多项式以及 $A^i (i \le n)$。那么利用 Cayley-Hamiliton 定理，我们可以将 A^k 表示为 $A^i (i \le n)$ 的线性组合。具体的系数就是 $x^k \mod p_A(x)$。

$x^k \mod p_A(x)$ 可以利用多项式快速幂在 $O(n^2 \log)$ 的复杂度内计算得到。现在我们来思考一下，在模质数意义下，如何求解特征多项式。

5.2.1 基于维护多项式的解法

将矩阵 A 改造为：

$$A' = \begin{bmatrix} a_{1,1}-x & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2}-x & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3}-x & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n}-x \end{bmatrix}$$

然后对矩阵在模 x^{n+1} 意义下做高斯消元，最后把对角线上多项式乘起来就是答案。

由于需要使用多项式乘法，复杂度为 $O(n^5)$。

5.2.2 基于拉格朗日插值的解法

我们对 λ 分别带入 $0,1,2,\cdots n$，然后算出 $|A-\lambda I|$。最后根据点值使用拉格朗日插值求出答案。复杂度为 $O(n^4)$。

5.2.3 基于上海森堡矩阵的解法

定义 5.2 (上海森堡矩阵)**.** 如果一个 n 阶矩阵 A 满足：$\forall i > j+1, a_{i,j} = 0$，那么 A 是一个上海森堡矩阵。

假如说我们能将 A 通过如下形式变为一个上海森堡矩阵 A'：

$$A' = P_1 P_2 \cdots P_k A P_k^{-1} \cdots P_2^{-1} P_1^{-1}$$

其中 A_i 为初等变换的矩阵。那么就有：$|A-\lambda I| = |A'-\lambda I|$。因为：

$$\begin{aligned}|A-\lambda I| &= |P_1 P_2 \cdots P_k (A-\lambda I) P_k^{-1} \cdots P_2^{-1} P_1^{-1}| \\ &= |P_1 P_2 \cdots P_k A P_k^{-1} \cdots P_2^{-1} P_1^{-1} - \lambda P_1 P_2 \cdots P_k P_k^{-1} \cdots P_2^{-1} P_1^{-1}| \\ &= |A'-\lambda I|\end{aligned}$$

我们考察左乘初等变换的矩阵，发现：将第 i 行加上第 j 行的 k 倍的逆矩阵，如果右乘到 A 上，恰好就是将第 j 列减去第 i 列的 k 倍。

也就是说对 A 的一次行列变换可以被描述为：将第 i 行加上去第 j 行的 k 倍，并且将第 j 列减去第 i 列的 k 倍。那么我们从 2 到 n 枚举 i，每次用第 i 行去消所有 $> i$ 行的第 $i-1$ 列，就可以将 A 变成 A'。

而上海森堡矩阵求行列式是可以使用 $O(n^2)$ 的 dp 来解决的。我们仍然带入 $1 \sim n$ 的点值进行拉格朗日插值。于是我们就可以在 $O(n^3)$ 的时间复杂度内解得特征多项式。

5.2.4 基于 Berlekamp-Massey 算法的解法

Berlekamp-Massey 算法能在 $O(n^2)$ 的时间复杂度内从长为 n 的序列中找到最短递推式。

我们可以直接对矩阵套用 Berlekamp-Massey 算法，但是由于矩阵的加法是 n^2 的，总复杂度就会变为 $O(n^4)$。我们可以随机选择一个行向量 a 和一个列向量 b，然后对 aA^kb 进行递推，这样求递推式的复杂度就为 $O(n^3)$ 了，瓶颈在于求向量乘矩阵。由 Schwartz-Zippel 引理，这样求出正确递推式的概率为 $1-\frac{2n}{p}$。

这种做法求出的最短递推式长度实际上是矩阵的不同特征值个数。

5.3 例题

给定一张 n 个点，m 条边的有向图，边有边权。第 i 个点的权值为 a_i。每一时刻所有点的权值更新为 $a'_u = \sum_{(v,u)\in E} a_v w(v,u)$。问 k 时刻后所有点的权值。答案对 10^9+7 取模。

数据范围：$n \le 5000, k \le 10^{18}$。

5.4 问题分析

暴力矩阵快速幂复杂度为 $O(n^3 \log)$。如果使用上海森堡来计算特征多项式，预处理时仍然有一个 $O(n^3)$，无法接受。

我们考虑使用 Berlekamp-Massey 算法。这个时候题目实际上已经给出了列向量 a，我们只需要再随机找一个行向量 b 相乘就行。预处理前 $n+1$ 时刻的所有 a，然后乘上 b 用于进行 Berlekamp-Massey。最后用多项式快速幂求出线性组合。

6 特征值与特征向量

定义 6.1 (特征值与特征向量)**.** 我们称，如果有非零 v，使得

$$Av = \lambda v$$

那么 λ 称为矩阵的特征值，v 则称为特征向量。

我们将式子变为：$(A - \lambda I)v = 0$。也就是说 $A - \lambda I$ 要线性相关，即 $|A - \lambda I| = 0$。因此 $p_A(\lambda) = 0$，也就是 λ 为特征多项式的根。

6.1 矩阵的对角化

如果一个 n 阶矩阵的所有特征值都互不相同，那么显然其有 n 个线性无关的特征向量。我们将所有特征向量 v_i 拼接起来，并把 λ_i 放到对角线上：$v_{i,j} = (v_i)_j$，$b_{i,j} = [i = j]\lambda_j$。

那么由 $Av_i = \lambda_i v_i$，得到 $AV = VB \Leftrightarrow V^{-1}AV = B$，其中 B 只有对角线上有值。

我们又发现：

$$A^k = P(P^{-1}AP)^k P^{-1}$$

也就是说，通过矩阵对角化，我们将求矩阵的幂转化为了矩阵乘法和对角线上值的幂。

6.2 例题 1

有 n 个编号为 $1, 2, \ldots, n$ 的厨师为国王准备菜肴, 第 i 个厨师的能力值为 i。最初第 i 个厨师的菜肴有美味值 a_i，其中 $|a_i| \le i$。每个厨师有一个允许他复制的厨师名单，国王确保厨师 i 只能从技能更强的厨师那里复制。

在每一天，厨师会进行两个阶段的操作来改变菜肴的美味值：

1. 可以选择令他的菜肴的美味值乘上他的能力值，也可以不乘。

2. 在所有厨师进行完操作 1 后，对于所有在第 i 个厨师的列表中的厨师 j，厨师 i 可以选择将自己的菜肴的美味值加上厨师 j 的菜肴的美味值或者不加。

所有的厨师都尽量令自己菜肴的美味值最大化。

最后有 q 个操作，每种操作是以下两种中的一个：

1. 1 k l r 询问 k 天后 l 到 r 的厨师的菜肴的美味值之和。

2. 2 i x 将第 i 个厨师菜肴初始的美味值 a_i 加上 x，其中 $x > 0$。

$1 \le n \le 300, 1 \le q \le 2 \times 10^5, k \le 10^3, -i \le a_i \le i$。

6.3 问题分析

题目来源：CF1540E, Tasty Dishes。

首先厨师一定是把列表中的所有厨师 $a_j > 0$ 的全部加一遍。由于每个人 $a_i > 0$ 的时间不一样，因此我们考虑对于每个人单独考虑其贡献。我们假设第 i 个人最早 $a_i > 0$ 的时间为 d_i。容易发现 d_i 只与 $a_i > 0$ 的集合有关，并且至多改变 $O(n)$ 次。

我们令 $\vec{e_i}$ 为只有第 i 位为 1 的向量，假设转移矩阵为 T 那么问题变为求

$$\sum_{d_i \le k} T^{k-d_i} \vec{e_i} a_i + \sum_{d_i > k} \vec{e_i} a_i$$

注意到特征向量满足 $Tv_i = \lambda_i v_i$。我们将 e_i 拆为特征向量的线性组合，这个可以 $O(n^3)$ 预处理。然后对于每个特征向量，求出它对于询问 $l \sim r$ 的贡献系数。于是我们现在只需要维护每个 $\lambda_i^k v_i$ 的系数。

我们可以使用 n 个树状数组来维护 $d_i \le k$ 的 e_i 对于 $\lambda_i^k v_i$ 的贡献系数和。于是最后的复杂度为 $O(n^3 + qn \log n)$。

6.4 例题 2

现在有一个随机变量 x。对于 $1 \le i \le n$，x 有 $a_i \times 10^{-9}$ 的概率为 i。

每一时刻 x 有 $\frac{x}{n}$ 的概率减少 1，有 $\frac{n-x}{n}$ 的概率增加 1。问最后 $x = i$ 的概率。答案对 998244353 取模。

$n \le 10^5$。

6.5 问题分析

题目来源：ARC133F, Random Transition。

我们先写出 $n + 1$ 阶转移矩阵：

$$A=\begin{bmatrix}0 & \frac{1}{n} & 0 & \cdots & 0 & 0\\ 1 & 0 & \frac{2}{n} & \cdots & 0 & 0\\ 0 & \frac{n-1}{n} & 0 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \cdots & \vdots & \\ 0 & 0 & 0 & \cdots & 0 & 1\\ 0 & 0 & 0 & \cdots & \frac{1}{n} & 0\end{bmatrix}$$

令 A 的第 $i+1$ 小的特征值为 λ_i，对应的特征向量为 v_i。我们可以发现：$\lambda_i=\frac{2i}{n}-1$。而特征向量 v_i 为多项式 $(1+x)^i(1-x)^{n-i}$ 的系数，即 $(v_i)_j=[x^j](1+x)^i(1-x)^{n-i}$。并且如果将 v_i 拼接为 P，那么 P^{-1} 的第 $i+1$ 列为 $n^{-2}v_{n-i}$。

对于 $b=Pa$，有

$$b_i=\sum_{j=0}^{n}a_j[x^i](1+x)^j(1-x)^{n-j}$$

那么，我们只需要求出

$$\sum_{i=0}^{n}a_i(1+x)^i(1-x)^{n-i}$$

直接分治 NTT 可以在 $O(n\log^2 n)$ 的时间内求出。

但实际上，我们可以做得更优。令 $t=1+x$，问题变为求 $\sum_{i=0}^{n}a_it^i(2-t)^{n-i}$。发现 a_i 对于 $[t^j]$ 的贡献为一个组合数，可以拆开以后利用 NTT 求出。然后我们需要利用 $f(t-1)$ 还原得到 $f(x)$，直接将 $f(t-1)$ 展开，贡献仍为组合数，不再赘述。

于是，我们在 $O(n\log n)$ 的时间复杂度内计算出了 $b=Pa$。对于 $b=P^{-1}a$ 我们也能类似地计算。而中间计算 A'^k 只需要 n 次快速幂，因此整个问题的复杂度为 $O(n\log n)$。

6.6 小结

在上面两道例题中，我们利用了矩阵的特征值以及特征向量，将原本难以计算的矩阵快速幂转化为了对角线上元素的快速幂，从而降低了复杂度。实际上，第二道题的官方做法并不是矩阵快速幂，而是观察性质后求生成函数。但是在我们无法观察出性质的时候，将转移矩阵的特征向量进行打表后往往能有新的发现。

7 特征值的计算

第 6 节中讨论的情况基本都是能直接求解出特征值的，在本节我们将针对更为一般的情形，介绍三种求数值解的方法以及一种求模意义下解的方法。这些方法也可以帮助我们进行打表的工作。

7.1 迭代幂法

对于 n 阶矩阵 A，如果其绝对值最大的特征值唯一，那么我们可以利用迭代幂法求出 A 绝对值最大的特征值以及对应的特征向量。

我们先随便找一个非零列向量 x，然后重复以下步骤：

1. 计算 $x' = Ax$。
2. 将 x' 缩放使得 x' 的模长为 1。
3. 将 x 赋为 x'。

重复若干步后 x 将会收敛于 $|\lambda|_{max}$ 的 v。

证明. 我们先将 x 分解为特征向量 v_i 的线性组合：$x = \sum_i p_i v_i$。那么迭代 k 次之后，有 $x = \sum_i \lambda_i^k p_i v_i$。那么经过若干次迭代后，其他向量的 λ_i^k 相对于 $|\lambda|_{max}^k$ 可以忽略不计，也就是说 x 里几乎只剩下了最大特征向量。

□

类似地，将整个过程反过来可以求出 $|\lambda|$ 最小的特征值以及特征向量。

7.2 Jacobi 方法

定义 7.1 (Givens 旋转矩阵)**.** *Givens* 旋转矩阵 $R(p, q, \theta)$ 定义为：

$$R(p,q,\theta)_{i,j} = \begin{cases} 1, & i = j, i \neq p, i \neq q \\ \cos\theta, & i = j = p \text{或} i = j = q \\ \sin\theta, & i = p, j = q \\ -\sin\theta, & i = q, j = p \end{cases}$$

对于实对称矩阵 A（$A = A^T$），求出 $B = R(p,q,\theta)AR^T(p,q,\theta)$，那么有：

$$\begin{cases} b_{i,j} = a_{i,j}, & i,j \neq p,q \\ b_{p,i} = b_{i,p} = \cos\theta a_{p,i} + \sin\theta a_{q,i}, & i \neq p,q \\ b_{q,i} = b_{i,q} = \cos\theta a_{q,i} - \sin\theta a_{p,i}, & i \neq p,q \\ b_{p,q} = b_{q,p} = \sin\theta\cos\theta(a_{q,q} - a_{p,p}) + (\cos^2\theta - \sin^2\theta)a_{p,q}\theta & \\ b_{p,p} = \sin^2\theta a_{p,p} + \cos^2\theta a_{q,q} - 2sin\theta\cos\theta a_{p,q} & \\ b_{q,q} = \cos^2\theta a_{p,p} + \sin^2\theta a_{q,q} + 2sin\theta\cos\theta a_{p,q} & \end{cases}$$

容易发现有 $\sum_{i,j} b_{i,j}^2 = \sum_{i,j} a_{i,j}^2$。

我们注意到 $b_{p,q} = b_{q,p} = \frac{1}{2}\sin 2\theta(a_{q,q} - a_{p,p}) + \cos\theta a_{p,q}$。那么令 $\tan 2\theta = \frac{2a_{p,q}}{a_{q,q}-a_{p,p}}$，就有 $b_{p,q} = b_{q,p} = 0$。此时发现 B 的对角线元素的平方和相比 A 增加了 $2a_{p,q}^2$。

我们每次都找到 $|a_{p,q}|$ 最大的 p,q 进行操作，那么在经过足够次数的迭代后，A 的非对角线元素会趋近于 0。此时 A 对角线上元素的就是特征值，而之前操作过的 $R(p,q,\theta)$ 按顺序依次相乘就可以得到特征向量。

令 $\mathrm{T}(A)$ 表示非对角线元素的平方和。那么显然会有 $\mathrm{T}(B) \leq (1 - \frac{2}{n(n-1)})\mathrm{T}(A)$。为了使 $\mathrm{T}(B) < \epsilon$，则要进行至少 $\frac{\ln \mathrm{T}(A) - \ln \epsilon}{\ln(n(n-1)) - \ln(n(n-1)-2)} \approx \frac{n^2}{2}(\ln \mathrm{T}(A) - \ln \epsilon)$ 次操作。如果使用数据结构维护平方最大值，则可以实现 $O(n^3(\ln \mathrm{T}(A) - \ln \epsilon)\log n)$ 的复杂度。而实际上，我们发现，如果我们对于每行维护一个最大值所在位置，那么每次操作期望只会改变 $O(1)$ 行的最大值，于是可以实现 $O(n^3(\ln \mathrm{T}(A) - \ln \epsilon))$ 的复杂度。

7.3 QR 分解

定理 7.1 (QR 分解)**.** 任何实数非奇异矩阵 A 都存在 QR 分解，即：

$$A = QR$$

其中若限定 R 对角线元素为正，则该分解是唯一的。这里 Q 为正交矩阵，也就是 $Q^TQ = I$，而 R 为上三角矩阵。

7.4 Householder 变换

Householder 变换可以用于求解矩阵的 QR 分解。其具体流程为：

1. 令 x 为矩阵 A 第一列的向量，令 e_1 为 n 维列向量 $[1,0,0,\cdots,0]^T$。
2. 计算 $v = x - |x|e_1$，其中 $|x|$ 为 x 的模长。
3. 计算 $\omega = \frac{v}{|v|}$。

4. 计算 $Q = I - 2\omega\omega^T$。

容易发现这样计算出的 Q 满足 QA 只有第一列只有第一行处有值。这启发我们像高斯消元那样递归下去，得到最终的 R。也就是 $R = Q_n \cdots Q_2 Q_1 A$，那么我们便可以得到：$A = Q_1^T Q_2^T \cdots Q_n^T R$，也就是 $Q = Q_1^T Q_2^T \cdots Q_n^T$。

7.5 QR 迭代法

对于 n 阶非奇异矩阵 A，进行 QR 分解：$A_1 = Q_1 R_1$。然后按照如下公式迭代：

$$A_{k+1} = R_k Q_k = Q_k^T A_k Q_k = Q_{k+1} R_{k+1}$$

那么若 A 的特征值的绝对值互不相同，则有 $\lim_{k\to\infty} A_k = T$。其中 T 是一个上三角矩阵。

而又有 $T = Q_k^T \cdots Q_2^T Q_1^T A_1 Q_1 Q_2 \cdots Q_k$，于是得到 $|A| = |T|$。而上三角矩阵的特征值就是对角线上元素。得到特征值后可以通过高斯消元解方程解出特征向量。

由于篇幅有限，此处不给出此方法收敛的证明，感兴趣的读者可以下来自己阅读参考文献。

7.6 多项式求根

在实数域上由于精度有限，使用多项式求根来求出特征值往往误差会比较大，因此一般都不会使用这种方法。然而如果是在模质数意义下求特征值时就可以采用这种方法。

在模 p 意义下对多项式进行分解有许多种做法，这里作者提供一种：每次将 $F(x)$ 的系数随机平移 d，求出 $\gcd(F(x+d), x^{\frac{p-1}{2}})$。显然，如果 $x - w | x^{\frac{p-1}{2}} - 1$，那么 w 在模 p 意义下有二次剩余。那么每次求 $\gcd(F(x+d), x^{\frac{p-1}{2}})$ 会期望得到 $F(x)$ 一半的根，可以将 $F(x)$ 分为两部分分别递归求解。

那么我们只需要使用 3.1.3 中提到的算法求出特征多项式，然后求出所有的根就可以得到特征值。再跑 n 次高斯消元就可以解出特征向量。

8 总结

本文探讨了矩阵的行列式、特征方程、特征值以及特征向量在信息学竞赛中的应用。信息学中的动态规划（即 dp）与矩阵也有着非常紧密的联系，许多 dp 都可以通过矩阵快速幂进行优化。在一些计数题中，如果难以优化 dp，那么可以尝试列出转移矩阵并且求出特征向量后打表观察规律，或许就会有不小的收获。

矩阵特征值与特征向量的数值解计算还有其他的许多算法，碍于笔者的水平有限，并没有进行进一步介绍。感兴趣的读者可以自行研究一下参考资料中的内容。

9 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢父母对我的关心和支持。
感谢林鸿老师、蔺洋老师和叶诗富老师对我的培养。
感谢所有伴我走过 OI 之路的人。

参考文献

[1] Characteristic Polynomial, https://en.wikipedia.org/wiki/Characteristic_polynomial

[2] Berlekamp-Massey algorithm, https://en.wikipedia.org/wiki/Berlekamp%E2%80%93Massey_algorithm

[3] Eigenvalues and eigenvectors, https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors

[4] QR decomposition, https://en.wikipedia.org/wiki/QR_decomposition

[5] Householder transformation, https://en.wikipedia.org/wiki/Householder_transformation

[6] 基于 QR 分解的特征值迭代法, https://blog.csdn.net/weixin_44246009/article/details/115263946

[7] Jacobi eigenvalue algorithm, https://en.wikipedia.org/wiki/Jacobi_eigenvalue_algorithm

[8] 非对称特征值问题的计算方法, https://www.cnblogs.com/Bluemultipl/p/15913516.html

[9] 经典 Jacobi 方法用于求解矩阵特征值, https://zhuanlan.zhihu.com/p/262879394

浅谈一些二分图匹配相关问题

华东师范大学第二附属中学 柯绎思

摘 要

二分图匹配问题是信息学竞赛中的一类常见问题。本文先总结了几种特殊的二分图匹配问题，之后研究了可匹配集上的最优化问题。

1 引言

二分图匹配问题是信息学竞赛中的一类常见问题，许多问题背后都有匹配相关的结构。本文主要总结了几类相关的问题。

第二节中，介绍二分图匹配的基础定义。

第三节中，总结几种特殊的二分图匹配问题。

第四节中，介绍匹配的拟阵性质。

第五节中，给出一个求字典序最小的最大可匹配集的算法。

2 前置知识

定义 2.1 一张无向图 $G=(V,E)$ 为**二分图**，当且仅当 G 的点集 V 能划分成 V_1,V_2 满足 $V_1\cap V_2=\varnothing, V_1\cup V_2=V$，且 $\forall(u,v)\in E$ 均有 $u\in V_1,v\in V_2$ 或 $u\in V_2,v\in V_1$。

定义 2.2 二分图 G 的一组**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。

若存在 M 使 $|M|=|V_1|$，称存在 V_1 的**完美匹配**。

若 $|M|$ 是所有匹配中最大的，则称 M 为**最大匹配**。

对于 $S\subseteq V_1$，记 $N(S)=\{v:\exists(u,v)\in E,u\in S,v\in V_2\}$。

定理 2.1（Hall 定理） 二分图 $G=(V_1,V_2,E)$ 存在 V_1 的完美匹配当且仅当 $\forall S\subseteq V_1,|S|\le|N(S)|$。

求二分图的最大匹配的经典算法是**匈牙利算法**，即依次加入每个 $u\in V_1$ 寻找增广路，复杂度为 $O(|V||E|)$。需要注意只要一个点加入匹配，那么它就会一直留在匹配中。

3 与区间相关的匹配问题

有一类常见问题为给定一些区间 $[l_i, r_i]$ 以及一些点 x_i，询问相关的匹配问题。此类问题通常能通过贪心算法或 Hall 定理解决。

本节总结了四种经典的问题。

3.1 问题 3.1

给定 n 个区间 $[l_i, r_i]$ 以及 m 个点 x_i，区间 $[l_i, r_i]$ 与点 x_j 能匹配当且仅当 $x_j \in [l_i, r_i]$，求最大匹配。

该问题有非常经典的贪心算法：将区间按**右端点**从小到大排序，依次考虑每个区间，若存在与其能匹配且还未匹配的点，则选择坐标最小的点与其匹配。

考虑其正确性：若区间 $[l, r]$ 能匹配 x_i 和 x_j 且 $x_i < x_j$，则剩余区间中能匹配 x_i 的一定能匹配 x_j，故 $[l, r]$ 匹配 x_i 更优。

3.2 问题 3.2

考虑**问题 3.1** 的一个推广。

给定 n 个区间 $[l_i, r_i]$ 以及另外 m 个区间 $[x_i, y_i]$，区间 $[l_i, r_i]$ 与 $[x_j, y_j]$ 能匹配当且仅当 $[x_j, y_j] \subseteq [l_i, r_i]$，求最大匹配。

类似**问题 3.1**，将 $[l_i, r_i]$ 按**右端点**从小到大排序，依次考虑每个区间 $[l_i, r_i]$，找到未匹配的包含于其中的**左端点**最小的区间 $[x_j, y_j]$ 与其匹配。

该算法的正确性与**问题 3.1** 类似。

3.2.1 例题 Clique[1]

给定环上的 n 段弧，建一张 n 个点的图，i、j 之间有边当且仅当弧 i、j 交非空，求该图的最大团。

$n \leq 3000$。

注意到将任意一个团中经过位置 1 的弧删掉后，剩余弧一定经过一个公共点。枚举该公共点 i，所有弧可以分为四类：

1. 同时经过 1 和 i，它一定在此时的最大团中。
2. 只经过 1。

[1] https://qoj.ac/problem/1178

3. 只经过 i。

4. 既不经过 1，也不经过 i，它一定不在此时的最大团中。

只保留 2、3 类弧，考虑求出其反图的最大独立集。注意到反图为二分图，最大独立集 = 总点数 – 最大匹配。将一边的弧同时取反后就转化为**问题 3.2**。求 $O(n)$ 次最大匹配，复杂度为 $O(n^2 \log n)$。

3.3 问题 3.3

给定 n 个区间 $[l_i, r_i]$ 以及 $n-1$ 个点 x_i，有 q 次询问，每次新加一个点 y，询问当前有没有完美匹配，询问间独立。

还是考虑**问题 3.1** 中的贪心，将区间按右端点从小到大排序后依次遍历，找到第一个没有点与其匹配的区间 $[l_i, r_i]$，则 y 应该满足 $y \le r_i$。忽略掉该区间后继续遍历，如果又存在一个区间没有点能与其匹配，则对于任意 y 都不存在完美匹配。遍历完后就能得到合法 y 的一个上界，同理将该过程反过来做一遍就可以得到合法的 y 的一个下界。

记求出的上下界分别为 R、L，下面证明任意位于 $[L, R]$ 中的 y 都存在完美匹配。

引理 3.1 n 个区间 $[l_i, r_i]$ 和 n 个点 x_i 存在完美匹配，当且仅当对于任意区间 $[L, R]$，$\#\{i : [l_i, r_i] \subseteq [L, R]\} \le \#\{i : x_i \in [L, R]\}$。

证明 必要性显然，考虑充分性。由 **Hall 定理**，只需满足 $\forall S \subseteq \{1, 2, \cdots, n\}$，$|S| \le \#\left\{i : x_i \in \bigcup_{j \in S} [l_j, r_j]\right\}$。若 $\bigcup_{j \in S} [l_j, r_j]$ 由多个区间构成，则由于每个区间已经满足条件，其必然满足条件。因此，不妨设 $\bigcup_{j \in S} [l_j, r_j]$ 为一个连续区间 $[L_0, R_0]$。设 $S' = \{i : [l_i, r_i] \subseteq [L_0, R_0]\}$，则 $|S| \le |S'| \le \#\{i : x_i \in [L_0, R_0]\}$，得证。

记 $F(l, r) = \#\{i : [l_i, r_i] \subseteq [l, r]\} - \#\{i : x_i \in [l, r]\}$，则需要 $\forall l \le r$，$F(l, r) \le 0$。由于当前匹配数已为 $n-1$，故不存在 l, r 使 $F(l, r) > 1$。若 $F(l, r) = 1$，则包含于其中的区间没有完美匹配，因此贪心过程到 r 时已经有未匹配的区间出现，故一定有 $R \le r$。同理 $L \ge l$。所以 y 属于所有这样的 $[l, r]$ 的交中，加入 y 后 $F(l, r) \le 0$，得证。

3.3.1 例题 Permutation for Burenka [2]

称两个长度相等的数组类似，当且仅当对于每个子区间，它们最大值的下标都相等。给定一个 1 到 n 的排列 p 和一个数组 a，a 中有 k 个元素未知。现在给定一个大小为 $k-1$ 的集合 S，有 q 次询问，每次给出一个数 d，问能否把 $S \cup \{d\}$ 中的数填入 a 中的未知位置，使 a 与 p 类似。

保证 a、S、d 的元素互不相同，$n, q \le 3 \times 10^5$。

[2] https://codeforces.com/problemset/problem/1718/D

两个数组类似，当且仅当它们的笛卡尔树完全一样。故 a 的笛卡尔树的形态已知，因此可以确定出每个未知位置的取值范围。问题转化成 k 个区间与 k 个点是否有完美匹配，用**问题 3.3** 的解法即可。

3.4 问题 3.4

考虑**问题 3.1** 的一种更加特殊的情况：对于所有 i，$r_i = +\infty$，即只有 l_i 的限制，此时有更简洁的判定是否有完美匹配的方法。维护一个值域上的数组，对于每个 l_i，把 $[l_i, +\infty)$ 加 1；对于每个 x_i，把 $[x_i, +\infty)$ 减 1。有完美匹配当且仅当数组的每个位置都非负。由 **Hall 定理**易得其正确性。

3.4.1 例题 1 Examination[3]

给定两个长度为 n 的数组 A_i 和 B_i，要求选择一些位置，将上面的 A_i 重排，使得 $\forall i = 1, 2, \cdots, n$，有 $A_i \geq B_i$。求最少选择的位置数量。

$n \leq 3 \times 10^5$。

显然所有 $A_i < B_i$ 的位置都要被选中。现在还要再选择一些其他的位置。加入位置 j 的效果就是在维护的值域数组上把 $[B_j, A_j)$ 区间加 1。问题转化成选择最少的区间加 1 使得数组上所有位置非负。该问题可以贪心解决：每次取出值小于 0 的最靠左的位置，在剩余区间中选择包含它的右端点最大的区间加 1。该过程最多重复 n 次，复杂度为 $O(n \log n)$。

3.4.2 例题 2 Two Faced Cards[4]

给定 n 张卡片，第 i 张正面是 A_i，反面是 B_i，再给定 $n+1$ 个数字 C_i。有 q 次询问，每次新给一张正面是 D，反面是 E 的卡片，要求将一些卡片翻转，使得 $n+1$ 张卡片与 $n+1$ 个数字能完美匹配，满足卡片正面的数字 $\leq$ 匹配的数字，求最小的翻转次数。询问间独立。

$n, q \leq 10^5$。

先钦定所有卡片不翻转，则翻转卡片 i 的效果就是把 $[b_i, a_i)$ 加 1，而新加一张正面为 x 的卡片的效果就是把 $[x, +\infty)$ 加 1。故问题转化成，对于每个 x 求出最少选多少区间加 1 使得 $[0, x)$ 都 ≥ 0，$[x, +\infty)$ 都 ≥ -1。

该问题可以贪心解决：首先，每次找到小于 -1 的最靠右的位置，选择包含它的左端点最小的区间加 1。重复这个过程知道所有数都 ≥ -1。然后，每次找到小于 0 的最靠左的位

[3]https://atcoder.jp/contests/arc147/tasks/arc147_e

[4]https://atcoder.jp/contests/agc013/tasks/agc013_f

置，选择包含它的右端点最大的区间加 1，并时刻维护 x 的答案。

该贪心算法的正确性与本文关系不大，故略去。

4 可匹配集的拟阵性质

定义 4.1 记 $M=(S,\mathcal{I})$ 表示一个定义在有限集 S 上，独立集的集合为 $\mathcal{I}$ 的**拟阵**，其中 $\mathcal{I}$ 为 S 的非空子集族。拟阵需要满足：

1. 遗传性：$\forall I\in\mathcal{I},J\subset I$，有 $J\in\mathcal{I}$。

2. 交换性：$\forall I,J\in\mathcal{I},|J|>|I|$，$\exists z\in J\setminus I$ 满足 $I\cup\{z\}\in\mathcal{I}$。

$\mathcal{I}$ 中的元素被称为**独立集**。

定义 4.2 在拟阵 $M=(S,\mathcal{I})$ 中，对于 $I\in\mathcal{I}$，若 $\forall x\in S\setminus I$，$I\cup\{x\}\notin\mathcal{I}$，则称 I 为 M 的**基**。

定义 4.3 在拟阵 $M=(S,\mathcal{I})$ 中，对于 $I\notin\mathcal{I}$，若 $\forall x\in I$，$I\setminus\{x\}\in\mathcal{I}$，则称 I 为 M 的**环**。

定义 4.4 在二分图 $G=(V_1,V_2,E)$ 中，对于 $S\subseteq V_1$，若存在一个匹配覆盖 S 中所有点，则称 S 为**可匹配集**。

对于二分图 $G(V_1,V_2,E)$，记 $\mathcal{I}$ 为 V_1 上的子集族，$S\in\mathcal{I}$ 当且仅当 S 为可匹配集。

定理 4.1 $(V_1,\mathcal{I})$ 为拟阵。

证明 遗传性显然，下面证明交换性。对于任意 $S_1,S_2\in\mathcal{I},|S_1|<|S_2|$，设它们对应的匹配分别为 $M,N,|M|<|N|$。设 E' 为 M 与 N 的对称差，由于每个点与 M、N 都至多有一条边相连，E' 中的联通分量只有：

1. 孤立点。

2. 长度为偶数的环，且环上的边交替地属于 $M\setminus N$ 和 $N\setminus M$。

3. 路径，且路径上的边交替地属于 $M\setminus N$ 和 $N\setminus M$。

又由于 $|N|>|M|$，故至少存在一条路径 E'' 同时以 $N\setminus M$ 中的边开头和结尾。该路径为 S_1 的一条增广路，所以 M 与 E'' 的对称差也为一个匹配，且覆盖 S_1 和一个 $x\in S_2\setminus S_1$。所以 $\exists x\in S_2\setminus S_1$ 使得 $S_1\cup\{x\}\in\mathcal{I}$，交换性得证。

因此，拟阵的相关结果，如最优化问题的贪心算法、基交换定理 [3]，都可以应用在可匹配集上。

4.1 例题 日程管理[5]

给定 q 次操作，每次加入一个截至日期为 t_i，收益为 p_i 的任务，或删除一个已有的任务。如果在不晚于第 t_i 天完成任务 i，能获得 p_i 收益。每个任务要花一天完成，每天最多完

[5]CTSC2015

成一个任务。每次操作后求出能获得的最大收益。

$t_i, q \le 3 \times 10^5$。

如果任务集合已经确定，相当于求拟阵的带权最大独立集，可以直接按照收益从大到小加入。

进一步考虑加入操作如何处理，若加入后依然为独立集，则可以直接加入；否则一定包含恰好一个环，此时可以去掉环上权值最小的元素。再考虑删除操作，只需将剩余元素中能加入的权值最大的元素加入即可，显然最多只能加入一个元素。由拟阵的带权最大独立集的贪心算法易得其正确性。

至于判定一个集合是否为独立集，可以发现该问题与**问题 3.4** 一致，维护值域上的数组即可。找环上权值最小的元素时，可以找到小于 0 的最靠左的位置 p，则所有 $t_i \le p$ 的任务都是环上的元素。找能加入的元素时，找到等于 0 的最靠右的位置 p，则所有 $t_i > p$ 的任务都可以加入。

用线段树维护，复杂度为 $O(q \log T)$。

可匹配集的拟阵性质在二分图最大匹配中可能没有直接应用，但是能使一些问题系统化，方便地得到一些贪心策略。

5 字典序最小的最大可匹配集

定义 5.1 对于集合 A、B，$|A| = |B|$，称 A 的**字典序**小于 B 当且仅当 $A \ne B$ 且 A 与 B 的对称差的最小元素属于 A。

对于一般二分图，按编号从小到大对 V_1 中的点进行增广，所得的即为字典序最小的最大可匹配集。但是可以做到更优秀的复杂度。

下文称字典序最小的最大可匹配集为**最优匹配**。

定义 5.2 对于图 $G = (V, E)$，称点集 $S \subseteq V$ 为 G 的**点覆盖**，当且仅当 $\forall (u, v) \in E$，$u \in S$ 或 $v \in S$。

定理 5.1 如果可以在 $T(|V_1|, |V_2|)$ 的时间复杂度内计算出二分图 $G = (V_1, V_2, E)$ 的最小点覆盖以及方案，则可以在 $T(|V_1|, |V_2|) \log |V_1|$ 的时间复杂度内计算出 V_1 的最优匹配。

下面直接给出做法。

考虑过程 $solve(S_1, S_2, S_3)$ 表示求解 $V_1 = S_1 \cup S_2$，$V_2 = S_3$ 的二分图的最优匹配，并且 S_2 中所有元素的编号小于 S_1 中所有元素且 S_2 一定在最优匹配中。答案就是 $solve(V_1, \varnothing, V_2)$。

将 S_1 的元素按照编号大小分成两部分，编号较小和较大的部分分别为 X、Y。先求二分图 $(S_2 \cup X, S_3)$ 的最小点覆盖 C。记 $B_l = (S_2 \cup X) \cap C$，$A_l = (S_2 \cup X) \setminus B_l$，$A_r = S_3 \cap C$，

$B_r = S_3 \setminus A_r$。

引理 5.2 二分图 (B_l, B_r) 的最大匹配大小为 $|B_l|$。

证明 最大匹配中每条边恰好有一个点在最小点覆盖里，且最小点覆盖中的点都在匹配边中，所以 B_l 中每个点的匹配点都不在最小点覆盖里，即都在 B_r 中。

引理 5.3 二分图 $(B_l \cup Y, B_r)$ 的最优匹配一定包含 B_l。

证明 由**引理 5.2**，二分图 (B_l, B_r) 的最优匹配为 B_l。又因为 Y 中点的编号都大于 B_l 中的点，由匈牙利算法的过程，B_l 一定在二分图 $(B_l \cup Y, B_r)$ 的最优匹配中。

引理 5.4 二分图 $(S_2 \cup X, S_3)$ 的最优匹配由 (A_l, A_r) 和 (B_l, B_r) 的最优匹配构成。

证明 由于 A_l、B_r 都不在点覆盖中，故 A_l 与 B_r 之间没有连边。又因为任何一条匹配边都恰好有一个点在最小点覆盖中，故 B_l 与 A_r 之间的边不可能成为匹配边。

引理 5.5 二分图 $(S_1 \cup S_2, S_3)$ 的最优匹配由 (A_l, A_r) 和 $(B_l \cup Y, B_r)$ 的最优匹配构成。

证明 在二分图 $(S_2 \cup X, S_3)$ 的最优匹配的基础上，逐个对 Y 中的点进行增广。类似**引理 5.2**，A_r 中所有点均已匹配且匹配点在 A_l 中。增广路一旦进入 $A_l \cup A_r$ 就无法出来且无法找到未匹配点，所以 (A_l, A_r) 的匹配不会改变。

由**引理 5.5**，可以把问题分成两部分，分别为 $solve(A_l \cap S_1, A_l \cap S_2, A_r)$ 和 $solve(Y, B_l, B_r)$。因为 $|A_l \cap S_1| \le |X|$，令 X、Y 大小相等即可让 $|S_1|$ 每次减半。又因为每个点恰好被分到一侧，因此时间复杂度为 $T(|V_1|, |V_2|) \log |V_1|$。

该算法用伪代码表示如下：

算法 6 求解过程 *solve*

输入： S_1，S_2，S_3

输出： 二分图 $(S_1 \cup S_2, S_3)$ 的最优匹配

if $S_1 = \varnothing$ **then**
  return S_2
else if $|S_1| = 1$ **then**
  if 二分图 $(S_1 \cup S_2, S_3)$ 存在 $S_1 \cup S_2$ 的完美匹配 **then**
    return $S_1 \cup S_2$
  else
    return S_2
  end if
else
  将 S_1 按编号分成大小相等的两部分 X、Y
  $C \leftarrow$ 二分图 $(S_2 \cup X, S_3)$ 的最小点覆盖
  $B_l \leftarrow (S_2 \cup X) \cap C$
  $A_l \leftarrow (S_2 \cup X) \setminus B_l$
  $A_r \leftarrow S_3 \cap C$
  $B_r \leftarrow S_3 \setminus A_r$
  return $solve(A_l \cap S_1, A_l \cap S_2, A_r) \cup solve(Y, B_l, B_r)$
end if

5.1 例题　Insects[6]

给定平面上 n 个黑点 (a_i, b_i)，再依次加入 m 个白点 (x_i, y_i)，黑点 i 与白点 j 能匹配当且仅当 $a_i \le x_j, b_i \le y_j$。每次加入白点后求最大匹配。

$n, m, a_i, b_i, x_i, y_i \le 10^5$。

本题的朴素做法为依次对每个白点进行增广，容易发现这相当于求白点字典序最小的最大可匹配集。求解最大匹配的过程与**问题 3.2** 类似。为了求出最小点覆盖，使用经典的构造最小点覆盖的方法 [4]，使用线段树优化建图可以做到 $O(n \log n)$，所以总复杂度为 $O(n \log^2 n)$。

本节介绍的算法的意义在于，对于一些特殊的二分图，将点按照某个特殊的顺序进行增广能高效地计算最大匹配，而该算法使得按照任意顺序进行增广的结果都能较高效地计算出来。

[6] https://qoj.ac/problem/4219

6 总结

本文总结了贪心算法、Hall 定理在几个特殊的二分图匹配问题上的应用，介绍了二分图可匹配集的最优化问题的相关结果，并最后给出了一个求字典序最小的最大可匹配集的算法，该算法使得特殊二分图上按指定顺序增广的结果能够高效计算。

由于笔者才疏学浅，所以介绍的内容较为简单。二分图匹配问题还有很多值得研究的地方，感兴趣的读者可以自行尝试。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢教练金靖老师对我的指导。
感谢父母对我的培育和教诲。
感谢柯怿憬同学对本文的帮助。
感谢所有帮助我的同学、老师。

参考文献

[1] Wikipedia, Hungarian algorithm.

[2] Wikipedia, Hall’s marriage theorem.

[3] 杨乾澜，《浅谈拟阵的一些拓展及其应用》，IOI2018 中国国家候选队论文

[4] https://oiwiki.org/graph/graph-matching/bigraph-match/# 二分图最小点覆盖 könig-定理

浅谈并查集在 OI 中的特殊应用

杭州第二中学 王相文

摘 要

并查集是一种朴素但有效的算法，具有广泛的应用场合。本文介绍了常用的三种并查集模型，并通过大量例题展示了三种模型的应用。

引言

并查集是一个在初学信息学时就会接触到的算法，但大多数选手在学习时都止步于最基础的并查集。事实上并查集算法具有很强的拓展性，在不少题目中可以比经典解法做得更简洁，更快速。

笔者在本文中梳理和总结了并查集在信息学中的应用，希望能通过本文将不常见的并查集的应用带入大家的视野。

本文的第一节会对并查集算法本身进行介绍，第二节至第四节分别介绍了三种不同的并查集模型与其应用，最后总结全文。

1 并查集简介

并查集是一种用于管理元素所属集合的数据结构，实现为一个森林，其中每棵树表示一个集合，树中的节点表示对应集合中的元素。

常见的并查集支持两种操作：

1. Union(x, y)：合并 x 与 y 的所属集合（合并 x 和 y 所属的树）。
2. Find(x)：查询 x 所属集合（查询 x 所属树的根节点）。

实现时我们只需维护每个点的父亲节点 fa。Find 操作可以不断跳父亲节点直到跳到树根，Union 操作可以找到两个节点对应的根节点，并将一个根节点的父亲节点设为另一个根节点。

朴素实现的并查集复杂度是 $O(nq)$ 的，其中 n, q 分别是点数和操作数。下面将介绍几种常见的优化方法。

1.1 启发式合并

朴素实现的并查集 Find 操作最坏复杂度可能会达到 $O(n)$，与树高相关，我们可以考虑使用启发式合并的方法来降低树高。

我们对每个点额外维护子树大小 size，Union 操作时选择子树大小更小的根节点，将它的父亲设为另一个根节点，并更新另一个根节点的子树大小。

由于每次我们都把 size 小的点合并到 size 大的点上，因此每个点非根节点 u 必然有 $2siz_u \le siz_{fa_u}$，树高降低为 $O(\log n)$，Find 复杂度也降低为 $O(\log n)$。Union 复杂度与 Find 复杂度相同，总时间复杂度为 $O(q\log(n))$。

1.2 路径压缩

重新审视 Find 操作。我们其实只想知道每棵树的树根是什么，因此可以在一次 Find(u) 操作后，将 u 到根节点上的所有节点的父亲都设为根节点。如果原树是一条链，我们 Find 叶子节点后树就变成了一个菊花，可以有效降低树高。

可以证明，在点数和操作数分别为 n, q 且满足 $n \le q$ 时，路径压缩并查集的最坏复杂度为 $O(q\log_{(1+\frac{q}{n})} n)$。由于路径压缩并查集的复杂度证明与本文关系不大，这里不展开详细描述。[1]

1.3 启发式合并 + 路径压缩

我们还可以注意到启发式合并和路径压缩并不矛盾，可以同时使用这两个优化。可以证明，在点数和操作数分别为 n, q 且满足 $n \le q$ 时，同时使用启发式合并和路径压缩的并查集复杂度为 $O(q\alpha(q,n))$。[2] 我们先定义一个函数 $A(i,j)$ 满足：

$$A(i,j) = \begin{cases} 2^j & \text{if } i = 1 \\ A(i-1,2) & \text{if } i \ge 2, j = 1 \\ A(i-1, A(i,j-1)) & \text{if } i,j \ge 2 \end{cases}$$

然后给出 $\alpha(q,n)$ 的定义：$\alpha(q,n) = \min\{i \ge 1 | A(i, \lfloor \frac{q}{n} \rfloor) > \log n\}$。

注意到 $\alpha(n,n) \ge \alpha(q,n)$，我们可以用 $\alpha(n,n)$ 来估算出一个复杂度的界，为了方便，之后将其记作 $\alpha(n)$。注意到 $A(4,1) = 65536$，而 $A(5,1)$ 即使是位数也难以估量，因此在实际使用时 $\alpha(n)$ 可以看作一个小于 5 的常数。

[1] 可以在参考文献 [1] 中了解更多

[2] 可以在参考文献 [1]，[3] 中了解更多

2 树上并查集的线性做法

2.1 定义

给定一棵有根树，支持两种操作：

1. Union(x, fa_x)：将 x 与 x 的父亲之间的边设为已合并。
2. Find(x)：查询 x 沿已合并的边向父亲跳，能跳到的深度最小的点。

直接使用并查集维护，复杂度为 $O(q\alpha(n))$，下面将会给出一种时间复杂度为线性的做法。

2.2 解法

我们以阈值 $B = \lfloor \log n \rfloor$ 对原树分块[3]，使得每个块连通且每条边都恰好在一个块内。

我们对每个块内的点按深度从小到大标号，用一个 B 位整数维护块内没有跟父亲节点合并的点集，Unite 时只需要修改一位。再对每个点用一个 B 位整数储存到块内根节点上的路径集合。

并查集在块内的 Find 操作，就是一直向父亲跳直到跳到一个未和父亲合并的节点。在这里可以通过查询该点到根路径上的集合，与未与父亲合并的集合的交中深度最大的节点，可以通过整数的按位与和 highbit 查询来解决。由于涉及到的整数都是 B 位整数，实际值不超过 $2^B = O(n)$，因此 highbit 可以提前预处理。

并查集在块外的 Find 操作可以直接启发式合并解决。由于块内的 Find 操作已经可以 $O(1)$ 处理，每个块可以只保留根节点，缩点后的树点数只有 $O(\frac{n}{B})$。在缩点后的树上调用 Unite 时，我们将小的集合内的所有点的父亲直接设为大的集合的根，Unite 所需的复杂度为 $O(\frac{n}{B}\log(\frac{n}{B})) = O(n)$，查询复杂度为 $O(1)$，这部分总复杂度为 $O(n+q)$。

实际查询的时候，我们先进行一次块内的 Find 操作，如果查询点已经被合并到根节点，还要再进行一次块外的 Find 操作和一次块内的 Find 操作。总复杂度就是两种 Find 操作复杂度之和，即 $O(n+q)$。

2.3 应用

2.3.1 例题 1

给定一张纸条，纸条上依次有 n 个格子，第 i 个格子的颜色是 a_i。

每秒钟可以进行以下两种操作之一：

[3]树分块可见参考文献 [2]

1. 将某个格子染成一种颜色。

2. 移动到相邻的格子，要求移动前和移动后的两个格子的颜色相同。

给出 q 次独立的询问，查询从 l 走到 r 的最短时间。

保证 $n, q \le 10^6, l \le r$。[4]

2.3.2 例题 1 解法

考虑最暴力的走法。每次将下一个格子染成跟当前格子相同的颜色后走过去，这样需要花 $2(r-l)$ 的时间才能到终点。

思考一下什么情况下我们可以解决时间。如果有 $i < j, a_i = a_j$，那我们从 i 开始走到 j，j 这个格子是不用染色的，可以省下 1 的时间。我们希望选出尽可能多的这样的 (i, j) 对并满足所有 $[i, j)$ 区间不交，那就可以得到一个贪心做法。对每个点 i 处理出最小的 fa_i 满足 $[i, fa_i]$ 中有相同的数，如果没有合法的 fa 则令 $fa_i = n+1$。题意就可以转换为从 l 开始，不断执行 $l = fa_l$ 操作直到 $fa_l > r$，执行的操作数即为省下的时间。

原题到这里已经存在 $O((n+q)\log n)$ 的倍增做法了，可以通过此题，但我们可以给出一个线性做法。

注意到我们将所有的 i 和 fa_i 连上边，会形成一棵以 $n+1$ 为根的树，那么我们可以考虑离线处理。我们从小到大枚举询问的 r，每次把 r 和所有 r 的孩子 Unite 起来，此时调用 Find(l) 操作就可以找到 l 最终跳到的点是什么，两个点的深度相减即为省下的时间。

注意到这里的 r 满足 $r \le n$，因此对询问的排序可以线性，并查集部分是树上并查集，也是线性。最终整个题我们做到了 $O(n+q)$ 的复杂度。

2.3.3 例题 2

我们定义一个序列的权值为 max(前缀最大值个数，后缀最大值个数)。

给定一个长度为 n 的首尾相接的排列 p，你需要将其分成 k 段非空序列，使得每个数都恰好在一段里，且 k 段序列的权值之和尽可能大。

保证 $k \le n \le 6 \times 10^5, k \le 30$ [5]

2.3.4 例题 2 解法

我们假设最大值旁边没有断点，包含最大值的那一段左右两侧一定有一侧是没有贡献的，假设这段选择了贡献前缀最大值个数，那么最大值之后的数就不可能成为前缀最大值了。因此这个问题可以看成断环为链后可以剩下一个后缀不被划分进 k 段内。

[4]来源：王浏清，https://ioihw22.duck-ac.cn/problem/123

[5]来源：笔者命制的集训队互测题，https://ioihw22.duck-ac.cn/problem/102

可以考虑设 $dp_{i,j}$ 表示前 i 个数被分成了 j 段的最大权值。考虑如何从 $dp_{*,j-1}$ 转移到 $dp_{*,j}$。定义两个辅助数组 A,B，$A_{t,s}$ 表示 $dp_{s,j-1}$ 转移到 $dp_{t,j}$，且最后一段是通过前缀最大值个数转移的权值，$B_{t,s}$ 表示最后一段是通过后缀最大值个数转移的权值，那么 $dp_{t,j}$ 的值就是 $\max_{s<t}\max(A_{t,s},B_{t,s})$。

考虑如何高效维护 A,B 数组的值。先观察 A_t 与 A_{t-1} 之间的区别：如果 $A_{t,s}\neq A_{t-1,s}$，一定是 p_t 被选入了前缀最大值中，这意味着 $[s,t]$ 之间没有比 p_t 大的数，这样的 s 一定是一段后缀，因此 A_t 就是 A_{t-1} 进行一次后缀加 1 后得到的数组。

再观察 B_t 与 B_{t-1} 之间的区别：首先 p_t 一定是后缀最大值，因此要先进行一次全局加 1。接下来考虑有一些数原先是后缀最大值，p_t 加入后不再是后缀最大值了，这样的数产生的贡献要删掉，即进行一次前缀减 1 操作。我们可以用单调栈来维护需要删掉哪些数。

现在只需要找到一个数据结构来高效维护 A 与 B 即可，观察一下需要支持的操作：

1. 后缀加 1。
2. 前缀减 1。
3. 后缀插入一个数。
4. 求全局最大值。

2 操作可以看成全局减 1 后进行后缀加 1，现在只有后缀加正数操作了。注意到如果有 $i<j$ 且 i 的权值小于等于 j 的权值，那么 i 就不可能成为全局最大值了。我们可以用链表维护一个单调栈和它的差分数组，每次后缀加的时候找到加的位置在单调栈中的位置，剩余的部分就可以线性了，关键在于如何找到一次后缀加在单调栈中对应的位置。

我们用并查集维护每个点之后第一个在单调栈中的位置，初始 $fa_i=i$，在单调栈中删除一个数 p_i 时只需要将 i 和 $i+1$ 合并即可。注意到这其实也是一种树上并查集，不过树是一条链，因此并查集部分也可以做到线性，从 $dp_{*,j-1}$ 转移到 $dp_{*,j}$ 的过程可以在 $O(n)$ 复杂度内解决，于是整个题做到了 $O(nk)$ 的复杂度。

2.3.5 例题 3（最近公共祖先）

给定一棵 n 个节点的有根树，m 次询问 u 和 v 的最近公共祖先是哪个点。

保证 $n,m\leq 5\times 10^5$。[6]

2.3.6 例题 3 解法

考虑将询问离线，将每个询问 u,v 挂在 dfs 序较小的那个点上。对整棵树进行一次 dfs，dfs 到点 u 时考虑挂在 u 上的所有询问 v，u 和 v 的最近公共祖先就是 v 一直向父亲跳，直到

[6]来源：经典问题

跳到一个已经被 dfs 过的点。问题可以看成支持每次删一条树上的边，以及查询一个点沿着没删掉的边向上跳会跳到哪个点。

删边不太好做，我们可以按 dfs 倒序处理询问，就变成加入一条边，查询一个点往上跳会跳到哪个点，可以用树上并查集解决，时间复杂度为 $O(n+q)$。

2.3.7 例题 4

给定一个长为 n 的序列和 q 次操作，操作有两种类型：

1. 将 a_x 设为 y。
2. 查询 $l \le i \le r$ 中 a_i 的最大值。

1 操作次数 $\le n\sqrt{n}$，2 操作次数 $\le n$，$a_i \le n$，要求时间复杂度 $O(n\sqrt{n})$。

2.3.8 例题 4 解法

我们希望得到一个 $O(1)$ 单点修改，$O(\sqrt{n})$ 求区间最大值的方法。考虑以阈值 $B=\sqrt{n}$ 进行序列分块，散块可以暴力处理，于是问题变成 $O(n\sqrt{n})$ 次单点修改和 $O(n\sqrt{n})$ 次整块查最大值。

现在每个块已经独立了，我们只需要解决 $O(1)$ 单点修改和 $O(1)$ 查询全局最大值这个问题。

考虑对每个数处理出一个时间上的区间，表示这个数会在这个时间区间内出现。按数从大到小处理区间，就变成了每次将区间内未被覆盖的时间覆盖为当前数。我们可以用并查集来维护每个点下一个未被覆盖的点是什么，注意到这是链上并查集，因此时间复杂度就是线性的。

现在瓶颈在于将数从大到小排序，由于值域是比较小的，我们可以提前离线将数排好序，就能去掉排序的 log 了。

3 带权并查集

3.1 定义

初始有 n 个元素（元素有结合律），每个元素自己是一个集合，带权并查集可以支持三种操作：

1. 将一个集合内的元素全体加上一个元素。
2. 合并两个集合。

3. 查询一个元素的值。

3.2 解法

我们可以对每个节点维护一个子树加 tag。合并时不能直接将某个节点的父亲设为另一个点，因为另一个点上可能有子树加 tag，可以新建一个点，将两个节点的父亲都设为新建的节点。

注意到这个结构是可以支持路径压缩的，我们只需要把路径压缩过程中经过的点的权值设为实际权值即可，时间复杂度就是路径压缩并查集的复杂度 $O(q\log_{\frac{q}{n}+1} n)$。

如果信息可减，我们可以还是可以启发式合并的，假设要把 x 合并到 y 上，只需要先把 x 上的 tag 减去 y 上的 tag，就可以消除 y 对 x 的影响，复杂度为 $O(q\alpha(n))$。

3.3 应用

3.3.1 例题 1

有三类动物 A, B, C，这三类动物的食物链构成了环形。A 吃 B，B 吃 C，C 吃 A。

现在有 n 只动物，每只动物都是 A, B, C 中的一种。接下来给出 m 条信息，每条信息形如 x 和 y 是同类或 x 吃 y。如果信息跟之前出现的信息矛盾则为假信息，否则为真信息，问一共有多少条假信息。

保证 $n \le 5 \times 10^4, m \le 10^5$。[7]

3.3.2 例题 1 解法

我们可以用 $0, 1, 2$ 来代表 A, B, C 三类动物，运算在模三意义下进行。这样我们可以把给出的信息写成方程的形式：

1. x 与 y 是同类：$v_x = v_y$。

2. x 吃 y：$v_x + 1 = v_y$。

把每条方程都看成一条边，一个连通块内我们只关心它们的相对大小关系，可以先选择任意一组解。如果新加入的边两端点属于同一个连通块，只要查询两端点的差是否满足信息。如果不在同一个连通块，则可以通过将连通块全体加上一个数使得这条信息成立，可以用带权并查集实现。由于信息可减，复杂度为 $O(q\alpha(n))$。

[7]来源：https://www.luogu.com.cn/problem/P2024

3.3.3 例题 2

给定一棵 n 个点的树，每个节点上都有一个权值 w_i。给定 q 次询问，每次询问给出两个点 u 和 v，你需要求出 u 到 v 的路径形成的序列的最大子段和。

保证 $n, q \le 10^6, -10^9 \le w_i \le 10^9$。

3.3.4 例题 2 解法

最大子段和是可以支持合并的。我们可以维护一个四元组 (lmx, rmx, sum, ans)，分别表示一个序列的最大前缀和，最大后缀和，序列和以及序列内部的最大子段和。这个四元组是可以合并的，并且有结合律。

现在问题就转换成了求树链和，元素有结合律但不可减。考虑求出每个询问 u 和 v 的 lca，将一条链拆成两条直上直下的链。然后考虑离线。我们按深度从大到小枚举 lca，枚举到 u 时相当于把整个 u 的子树内的点全部加上 u 自身的四元组。如果用线段树实现常数较大，但这里可以直接用带权并查集来做。枚举到 u 时将 u 和 u 的所有孩子合并，然后将 u 所在的集合加上 u 自身的四元组，查询的时候调用 Find 即可。

注意到这个题信息不可减且无法启发式合并，因此时间复杂度就是路径压缩并查集的复杂度 $O(q \log_{\frac{n}{q}+1} n)$，空间复杂度线性。

4 可撤销并查集

4.1 定义

可撤销并查集支持三种操作：

1. Union(x, y)：合并 x 与 y 的所属集合（合并 x 和 y 所属的树）。
2. Find(x)：查询 x 所属集合（查询 x 所属树的根节点）。
3. Back()：撤销最后一个未被撤销的 1, 2 操作。

4.2 解法

注意到启发式合并的复杂度是单次操作严格 $O(\log n)$，而路径压缩并查集的复杂度是均摊的。我们可以用启发式合并并查集来维护这个可撤销的结构。

我们维护一个栈。每次 Union 操作时如果将 x 的父亲设为了 y，就在栈中加入一个元素 x。Back 操作时取出栈顶元素，切断它与父亲的连边，并将父亲的子树大小减去自己的子树大小。复杂度同启发式合并，单次操作 $O(\log n)$。

4.3 应用

4.3.1 例题 1

给定 n 个集合，第 i 个集合内初始状态下只有一个数，为 i。

有 q 次操作。操作分为 3 种：

1. 合并 a, b 所在集合。
2. 回到第 k 次操作（执行三种操作中的任意一种都记为一次操作）之后的状态。
3. 询问 a, b 是否属于同一集合，如果是则输出 1，否则输出 0。

保证 $n \leq 10^5, q \leq 2 \times 10^5$。[8]

4.3.2 例题 1 解法

我们可以直接用可持久化线段树维护并查集的操作，复杂度为 $O(m \log^2 n)$，但是复杂度偏大。

注意到题目并没有要求在线，我们可以建立一棵操作树，i 号节点的父亲由第 i 次操作决定，如果第 i 次操作为 1 或 3 操作父亲为 $i-1$，否则父亲为输入的 k。之后在操作树上 dfs，进入一个节点的时候加入节点上的边或处理节点上的查询，离开时撤销这个操作，复杂度为 $O(m \log n)$。

4.3.3 例题 2

你要维护一张 n 个点无向简单图。你需要执行 q 次操作，操作有如下三种类型：

1. 加入一条边。保证它不存在。
2. 删除一条边。保证它存在。
3. 查询两个点是否连通。

保证 $n \leq 5000, q \leq 5 \times 10^5$。[9]

[8]来源：https://www.luogu.com.cn/problem/P3402

[9]来源：https://loj.ac/p/121

4.3.4 例题 2 解法

注意到删边操作非常难处理，我们可以处理出每一条边的出现时间区间，将其插入到线段树上对应的 log 个区间内。查询时我们希望能处理出线段树上一个叶子节点到根节点上所有边的并查集。

考虑在线段树上进行 dfs，进入一个节点的时候将该节点上的所有边插入并查集，离开这个节点的时候撤销掉插入的边，dfs 到叶子节点时直接查询两个点是否连通。由于一条边会被插到 $O(\log q)$ 个线段树节点上，在并查集上插入一条边复杂度是 $O(\log n)$，总时间复杂度为 $O(q \log q \log n)$。

总结

本文介绍了线性树上并查集、带权并查集和可撤销并查集三种并查集模型，并且通过大量例题展示了其在 OI 中的应用。但并查集的应用肯定不止于此。笔者希望本文能起到抛砖引玉的作用，也希望感兴趣的读者能继续研究。

感谢

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彭思进、杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，李建老师的教导和同学们的帮助。

感谢徐安矣同学、叶子川同学、周欣学长、李欣隆学长、胡杨学长与我交流讨论、给我启发。

感谢李羿辰同学、徐安矣同学、叶子川同学、李建老师为本文审稿。

感谢其它所有本文所参考过的资料的提供者。

感谢各位百忙之中抽出宝贵的时间来阅读本文。

参考文献

[1] Tarjan, R. E., & Van Leeuwen, J. (1984). Worst-case analysis of set union algorithms. Journal of the ACM (JACM), 31(2), 245-281.

[2] 周欣. 《浅谈一类树分块的构建算法及其应用》，IOI2021 中国国家队候选队论文集.

[3] OI-wiki 贡献者. 并查集—OI-wiki, 2022.

[4] Gabow, H. N., & Tarjan, R. E. (1985). A Linear-Time Algorithm for a Special Case of Disjoint Set Union. JOURNAL OF COMPUTER AND SYSTEM SCIENCES, 30, 209-221.

浅谈静态 Top Tree 在树和广义串并联图上的应用

南京外国语学校 程思元

摘 要

本文介绍了静态 Top Tree 的概念，介绍了 Top Tree 上 pushup 维护信息、Top Tree 点分治、Top Tree 构建树分块这三种在树上问题中的应用，介绍了前两种应用推广到广义串并联图上的方法。并提出了一种利用 Top Tree 构建广义串并联图分块的方法。

引言

在信息学竞赛中，有很多树上的静态问题都可以用静态 Top Tree 解决。将 Top Tree 推广到广义串并联图上也可以解决一些广义串并联图上的问题。之前利用 Top Tree 解决广义串并联图问题的算法中，直接 pushup 维护信息的方法很难维护一些复杂的不可合并信息，而 Top Tree 点分治的方法由于需要处理子树外的边，很难支持对信息的修改。笔者通过研究，试图通过一种广义串并联图分块的方法解决上述问题。

1 簇的概念和运算

1.1 簇的概念

对于无向树 T，定义 T 上一个簇为一个三元组 $C = (V, E, B)$，其中 (V, E) 连通且是 T 的子图，集合 $B \subseteq V$，$|B| \leq 2$，且对于每个 $x \in V$，若 $\exists y \notin V, (x, y) \in E(T)$，则 $x \in B$。B 中元素称为 C 中的界点。[1]我们用 $V(C), E(C)$ 分别表示 C 的点集和边集，用 $B(C)$ 表示 C 中的界点集合。

我们这样定义的好处是在进行簇的运算时，只需要考虑 B 中的元素。由于 $|B| = 2$，我们可以将一个簇直观地视为一条边。

[1]有时，与一个点相邻的边之间是有序的，它们排成一了个环。此时，与每个界点相邻的边只允许有一端连续的区间在簇的边集中。不过这不在本文的讨论范围之内。

1.2 簇的运算

簇有两种运算，compress 和 rake。

1.2.1 compress 运算

对于无向树 T 上两个簇 C, D，满足 $B(C) \cap B(D) = v$，它们 compress 后的结果为 $(V(C) \cup V(D), E(C) \cup E(D), (B(C) - \{v\}) \cup (B(D) - \{v\}))$。

只有当 $|B(C) \cap B(D)| = 1$ 且运算结果仍是 T 上的簇时才可运算。

1.2.2 rake 运算

对于无向树 T 上两个簇 C, D，满足 $B(C) \cap B(D) = x$，它们 rake 后的结果为 $(V(C) \cup V(D), E(C) \cup E(D), B(C))$。

只有当 $|B(C) \cap B(D)| = 1$ 且运算结果仍是 T 上的簇时才可运算。

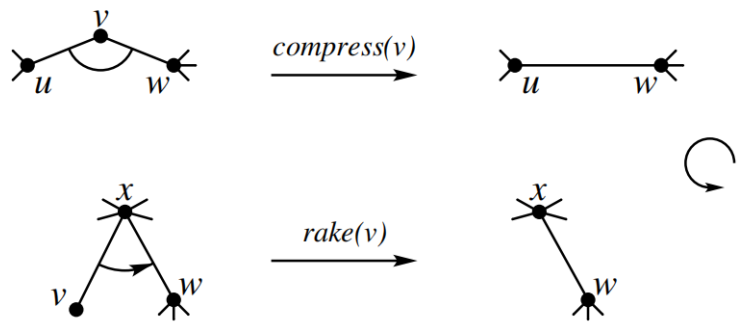


图 1: compress 和 rake[1]

2 Top Tree 的概念与静态构建

2.1 Top Tree 的概念

我们引入簇的概念，是为了把整棵树 T 写成簇运算的形式。即我们会进行一系列的簇运算，最终我们要得到一个簇 C 使得 $E(C) = E(T)$。

运算过程中的每个时刻，我们将当前存在的每个簇的两个界点之间连一条边，可以得到一棵树，称为收缩树。

我们选择所有 $(\{u,v\},\{(u,v)\},\{u,v\})|(u,v)\in E(T)$ 构成的树作为初始收缩树。由于初始每个簇中只含一条边，比较容易维护信息。

引理 2.1.1. 存在一个运算的序列从初始收缩树到达最终收缩树。

证明. 任选一个点作根，每次选择当前的收缩树上的一个叶子 x，对连接 x 和 x 的父亲 y 的簇 C 进行操作，若 x 有兄弟，就把 C rake 到它的兄弟上，否则把连接 y 和 y 的父亲 z 的簇和 C compress 起来，重复这样的操作一定可以得到整棵树构成的簇。 □

于是我们确定了初始收缩树和最终收缩树。下图是一个从初始收缩树通过运算得到最终收缩树的过程：

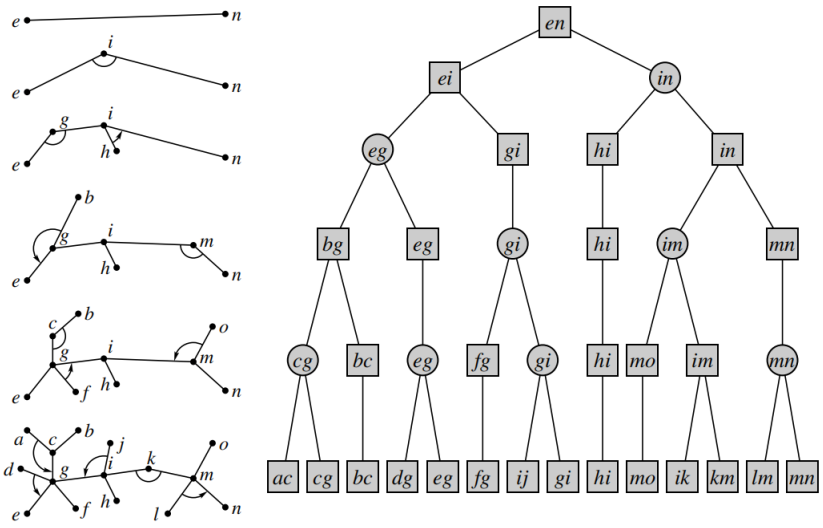


图 2: 每个时刻的收缩树和运算过程 [1]

对于一个从初始收缩树到最终收缩树的由一系列运算构成的过程，我们可以用一棵表达式树来描述它，称为 Top Tree，如下图所示（对于非叶子结点，方点表示 compress 操作，圆点表示 rake 操作）。

利用上述证明中的方法，我们已经可以对任意的树构建 Top Tree，事实上，还存在很多同样易于实现的算法，但是遗憾的是，它们构建出的 Top Tree 的深度是没有保障的。而一些将在下一节中提到的应用是依赖 Top Tree 深度的。

于是，我们想要构建一棵深度为 $\Theta(\log|V|)$ 的 Top Tree。

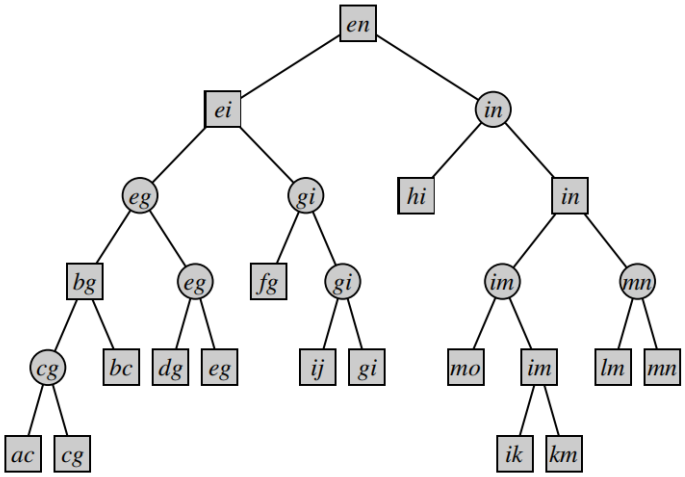


图 3: 图 2 中例子的 Top Tree[1]

2.2 Top Tree 的静态构建

2.2.1 重链剖分

我们首先解决一个问题。由于 Top Tree 是一棵表达式树，是一棵二叉树，所以若我们想把 k 个簇依次 compress 起来或依次 rake 起来，我们需要对其建立一棵线段树或广义线段树[2]。若我们建立不广义的线段树，则深度为 $\Theta(\log k)$。下面将为了实现这两种的操作建出的树分别称为 compresss 树和 rake 树。

我们考虑任选一个点做根，对原树进行重链剖分。对于一条重链，我们先递归它的所有轻儿子，然后我们进行如下的操作：首先，对于重链上的每个点，若它有轻儿子，则把它的轻儿子全部 rake 到重儿子上。然后，把整条重链上所有点以及重链顶的父亲（如果有的话）compress 到一起。这样，我们就成功把一条重链通过运算缩为了一个簇，使用的深度代价为 $\Theta(1)$ 棵 compress 树或 rake 树的深度，若使用上面提到的建不广义的线段树的做法，则一条重链的深度代价为 $\Theta(\log |V|)$。由于每个点到根的路径中至多包含 $\Theta(\log |V|)$ 条重链，这样构建的 Top Tree 的深度是 $\Theta(\log^2 |V|)$ 的。

2.2.2 全局平衡二叉树

我们发现，上面我们使用每次取中点分为两半的方法建线段树是浪费的。因为我们这样的建法保证了每个参与运算的簇在 Top Tree 上的深度几乎相同，但它们在 Top Tree 上的子树深度是不同的，而实际上，Top Tree 上子树深度较小的簇在这棵线段树上的深度可以适

[2]即每次递归时可以取任意点将当前区间划分为两半

量增大。

考虑一种新的建树算法：我们计算出待建广义线段树的每个簇在 Top Tree 上的子树大小，然后选择一个使左边的子树大小和尽可能大但不超过一半的位置，取这个位置将这些簇分为两部分，对两部分分别递归建广义线段树。特别地，如果这样得到的左子树为空，则令左子树只包含一个最左侧的子树。

引理 2.2.1. 对于一棵大小为 n 的树，使用这种算法建出的 *Top Tree* 深度为 $\Theta(\log n)$。

证明. 考虑 Top Tree 上的一个点 x，其大小为 sz_x。

若 x 的儿子的儿子中存在一个点，使得它和 x 不在同一棵 compress 树或 rake 树上，那么由于每个点到根的路径中至多包含 $\Theta(\log n)$ 条重链，在 Top Tree 上每个点到根的路径上也只有 $\Theta(\log n)$ 个这样的 x。这一部分点对深度的贡献为 $\Theta(\log n)$。

否则，x 的儿子的儿子都和 x 在同一棵 compress 树或 rake 树上。我们用下图表示对 x 及其子树建 compress 树或 rake 树时的划分方式，其中，红色的部分表示跨过 $\frac{sz_x}{2}$ 的那个叶子。显然 $a+b+c+d=\frac{sz_x}{2}$。

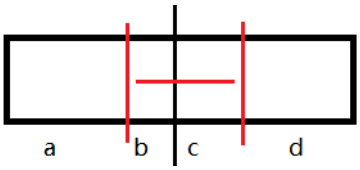


图 4: x 的划分

因为 x 的右儿子的左儿子和 x 在同一棵 compress 树或 rake 树上，故 x 的右儿子的左儿子的子树大小不超过 x 的右儿子的一半，也就是 $\frac{b+c+d}{2}$。而 x 的左儿子子树大小为 a，x 的右儿子的右儿子子树大小不超过 d，这三者均不超过 $\frac{sz_x}{2}$。

于是对于这样的 x，其儿子的儿子的子树大小一定不超过 $\frac{sz_x}{2}$，所以这部分点对深度的贡献也为 $\Theta(\log n)$。

于是我们证明了这样构建出的 Top Tree 的深度为 $\Theta(\log n)$。 □

以这种算法建出的广义线段树被称为全局平衡二叉树。

在某些问题上，rake 运算有比较好的性质，可以不建 rake 树。

3 Top Tree 在树上问题中的应用

3.1 直接 pushup 维护信息

我们利用簇的结构，可以维护一些和界点状态有关的半群信息。我们可以直接在 Top Tree 自底向上进行 pushup 来维护信息。

例如有一类树形 dp 问题，可以写成为每个点确定一个状态，每条边根据两个顶点的状态确定一个贡献。典型的此类问题有树上最大权独立集问题，在树上选常数个点计算某种权值等。对于这类问题，我们可以通过在簇中记录界点的每一对状态的 dp 值，把它写成簇合并的形式。由于 Top Tree 的深度是 $\Theta(\log|V|)$ 的，若我们想修改某条边的权值，只需对 $\Theta(\log|V|)$ 个点进行 pushup 操作。这样我们就在单次修改 $\Theta(\log|V|)$ 次信息运算的复杂度内解决了树上动态 dp 问题。

类似的，可以存储在簇上的类似信息均可以用此类方法维护。

例题 3.1.1. 【*NOIP2018*】保卫王国[3]

给定一棵 n 个点，点带权的无根树 T。m 次查询，每次固定 a,b 两个点的状态分别为 x,y，求 T 的最小权点覆盖。查询之间相互独立。

$n,q \le 10^5$。

我们来解决支持修改点权的问题。这样只需把点权改成正无穷或负无穷就可以解决固定点的状态的问题。

我们建出 T 的 Top Tree，在 Top Tree 上 pushup 维护信息。对 Top Tree 上的某个簇 c，在簇中记录 $f_{0/1,0/1}$ 表示簇的两个界点取到所有四种状态时，簇中其它点的最小代价。进行 compress 操作时，枚举两个簇的交点的状态；进行 rake 操作时，直接把相同的状态的代价加起来。

每次修改时，找到修改点权影响的 Top Tree 上的点，然后重新 pushup 修改的点到根的链即可。

时间复杂度为 $\Theta(n+m\log n)$。

例题 3.1.2. 【*NOI2022*】树上邻域数点[4]

给定一棵 n 个点的树 T。你可以进行一些 *compress* 和 *rake* 运算预处理出一些簇。接下来有 q 次询问，每次询问给定 x,d，要求使用不超过 1 次簇运算得到所有与 x 的距离不超过 d 的点构成的簇。运算使用交互的方式进行，强制在线。

$n \le 10^5, q \le 2\times 10^6$。预处理的运算次数不得超过 3×10^7。

建出 T 的 Top Tree。对于一次询问 x,d，我们找到与 x 相邻的任意边在 Top Tree 上的祖先中，完全包含所有与 x 的距离不超过 d 的点的簇中最浅的一个。设这个簇为 C，显然 d 不超过 C 在 Top Tree 上的父亲对应簇的直径。我们的答案就是 C 的两个界点在 C 以外距离不超过某个数的所有点构成的簇和 C compress 的结果。

于是对 Top Tree 上每个簇 C，我们要维护 $g_{0/1,d}$ 表示两个界点在 C 之外的距离不超过 d 的所有点构成的簇，其中 d 的取值范围是 C 在 Top Tree 上的父亲对应簇的直径。

[3]https://uoj.ac/problem/441

[4]https://uoj.ac/problem/766

考虑在 Top Tree 上 pushup 维护这个信息。对每个簇 C，我们维护辅助信息 $f_{0/1,d}$ 表示两个界点在 C 之内的距离不超过 d 的所有点构成的簇。我们可以先 pushup 维护出 f，然后利用 f 再进行类似换根 dp 的操作得到 g。

最后，我们回答一个查询需要两次运算，需要通过对每个簇 C 预处理其中一个界点对应的 g 中所有元素和 C compress 的结果来减少一次运算。

预处理的总次数是 Top Tree 上每个点的子树大小之和级别的，即 $\Theta(n \log n)$。

3.2 Top Tree 点分治

我们想要通过对 Top Tree 进行点分治来解决一些路径统计之类的问题。为了对 Top Tree 进行点分治，首先要对 Top Tree 上的连通子图的收缩树进行定义。

对于 Top Tree T 上的一个连通子图 T'，我们首先删掉原树中所有不在 T' 深度最浅的点子树中的边，然后依次对 $T - T'$ 中所有不和根连通的点进行对应的操作。这样得到的收缩树就是连通子图 T' 对应的收缩树。

有了这个定义之后，我们就可以对 Top Tree 进行下面的分治算法：当前有一个 Top Tree 上连通子图 T，在连通子图中选择一条边，使得两边的顶点个数尽可能接近，设这条边为 (x, fa_x)，则我们将 Top Tree 分为 $subtree_x$ 和 $T - subtree_x + \{x\}$，统计 T 的收缩树中所有端点是界点或端点不在同一个连通子图的路径的贡献，然后分别向两侧递归。当当前子图对应的收缩树中的所有点都在之前被作为界点计算过贡献时，就不用继续递归了。

由于递归了 k 层的连通子图对应的收缩树上最多只会有 k 条不在原树上的边，这些边的总出现次数是 $\Theta(|V| \log |V|)$。而原树上的一条边不会同时往两侧递归，这些边的总出现次数也是 $\Theta(|V| \log |V|)$。故所有收缩树的边数和为 $\Theta(|V| \log |V|)$。

3.3 树分块的构建

我们利用 Top Tree 可以构建一种性质较强的树分块。

假设我们要进行分块的树大小是 n，那么我们会选取 Top Tree 上所有满足子树大小 $\leq \sqrt{n}$，且父亲的子树大小 $> \sqrt{n}$ 的点对应的簇。

这样，我们就给出了原树的一个簇划分，每个簇的大小为 $\Theta(\sqrt{n})$，且簇的个数为 $\Theta(\sqrt{n})$。

这样分块有很多性质。首先，每个簇的大小都是 $\Theta(\sqrt{n})$ 的，这意味着每个簇的深度、直径等规模都是 $\Theta(\sqrt{n})$。而且，对于每个非界点，它向子树中走到的后继簇都是唯一的，这样也比较容易刻画子树的信息。

实际实现时，存在一些更简洁的树分块构建算法，由于不是本文讨论的重点，在此略去。相关的算法可参见参考文献 [4]。

4 更一般的图

我们发现，簇的定义实际上没有用到很多树的性质。我们试图将簇的概念扩展到连通图上。由于不再保证无环，我们首先要添加一种称为 twist 的运算。

4.1 twist 运算

对于无向连通图 G 上两个（广义的）簇 C, D，满足 $B(C) = B(D)$，它们 twist 后的结果为 $(V(C) \cup V(D), E(C) \cup E(D), B(C))$。

twist 运算也满足结合律。

从收缩树上看，twist 运算就是将两条平行边叠合在了一起。

4.2 能够收缩的图——广义串并联图

我们发现，即使有 compress、rake 和 twist 三种运算，仍有一些图不能通过这些运算收缩成一个簇，例如 K_4。

进一步的，如果一个图存在与 K_4 同胚的子图，那么它也是不能通过这些运算收缩成一个簇的。

事实上，不存在与 K_4 同胚的子图，既是能够收缩的充分条件，也是必要条件，我们将不存在与 K_4 同胚的子图的连通图称为广义串并联图。这个结论的证明比较比较复杂，可参见参考文献 [5]。

由于每个时刻簇组成的图不再是树了，我们这里将收缩树改称为收缩图。

我们可以通过重复以下步骤建立广义串并联图的 Top Tree[5]：

- 若收缩图上存在度为 1 的点 v，设它与 u 相邻，则将 $\{u, v\}$ 对应的簇 rake 到与 u 相邻的任意其它簇上。
- 若收缩图上存在度为 2 的点 v，设它与 u, w 相邻，则将 $\{u, v\}, \{v, w\}$ 对应的簇 compress 起来。
- 若收缩图上存在平行边，则将它们对应的簇 twist 起来。

根据上面的结论，这样最终一定能够将整张图缩为一个簇，不会在中途出现三种运算的条件都不满足的情况。

很遗憾，这样建出的 Top Tree 的深度是 $\Theta(|V|)$ 的。而且我们可以证明，在广义串并联图上，无论用什么方法建 Top Tree，建出的 Top Tree 的深度都是 $\Theta(|V|)$ 的。具体地，构造下面的图：

[5] 也称为串并联结构树

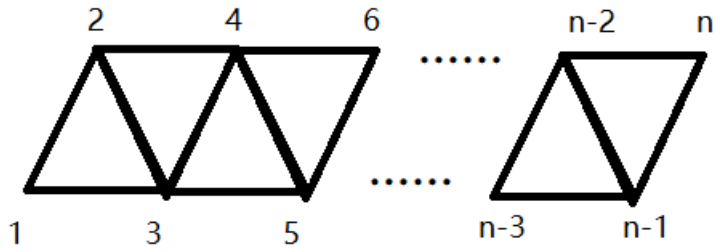


图 5: 将 Top Tree 的深度卡到 $\Theta(|V|)$ 的例子

无论如何，我们还是希望将上一节中提到的 Top Tree 的三种应用推广到广义串并联图的 Top Tree 上。

4.3 Top Tree 的 Top Tree

首先我们想要在 Top Tree 上直接 pushup 维护信息。但是直接 pushup 维护信息的时间复杂度是 Top Tree 深度也就是 $\Theta(|V|)$ 的。

我们发现，建 Top Tree 并在上面 pushup 维护信息，实际上是把一个在任意树上 pushup 维护信息的问题转化成了在 Top Tree 上 pushup 维护信息的问题，且单次修改复杂度为 Top Tree 的深度。我们现在的问题是在深度为 $\Theta(|V|)$ 的 Top Tree 上 pushup 维护信息，如果我们把这棵 Top Tree 看成普通的树，对这棵 Top Tree 再建一次 Top Tree，就可以把这个在深度为 $\Theta(|V|)$ 的 Top Tree 上 pushup 维护信息的问题，转化为在 $\Theta(\log|V|)$ 的 Top Tree 上 pushup 维护信息的问题。

我们考虑这样操作对簇中维护的信息量的影响。设原图中每个点有 k 种状态，由于簇中有两个界点，所以 Top Tree 上每个点有 k^2 种状态，所以 Top Tree 的 Top Tree 中的每个点有 k^4 种状态。实际问题中 k 一般是常数。

于是我们成功在单次修改 $\Theta(\log|V|)$ 次半群运算的复杂度内解决了问题。由于 Top Tree 是一棵二叉树，对 Top Tree 建 Top Tree 的时候，是不需要建 rake 树的。

例题 4.3.1. 【*2019* 集训队互测 *Day 3*】公园[6]

给定一张 n 个点的广义串并联图 G。每个可以染成黑色或白色。每个点有两个权值，分别表示染成黑色或白色的收益。每条边有两个权值，分别表示与这条边相邻的两个顶点的颜色相同或不同的收益。

有 q 次修改，每次修改一个点或一条边的权值，求任意染色的最大收益。修改之间不独立。

$n, q \le 10^5$。

建出 G 的 Top Tree T。对每个簇 C，记 $f_{0/1,0/1}$ 表示 C 的两个界点取到每种状态时，簇中的边和其它点贡献的最大收益。

对 T 再次建出 Top Tree T'。在 T' 的每个点对应的簇 C 上，维护 $g_{0/1,0/1,0/1,0/1}$ 表示 C 的每个界点对应的 G 上的簇的两个界点（共四个点）取到每种状态时，C 中其它点贡献的最大收益。直接 pushup 即可维护。

修改时，先找到修改的点或边在 T 上对应的点，然后找到这个点在 T' 上对应的点，重新 pushup 这个点到根的链即可。时间复杂度为 $\Theta(n + m \log n)$。

4.4 广义串并联图的 Top Tree 点分治

接下来考虑将树的 Top Tree 点分治推广到广义串并联图的 Top Tree 上。点分治本身不依赖树的深度，这里 Top Tree 的深度不会造成问题。

但是，由于广义串并联图不是树，我们如果沿用之前的 Top Tree 子图收缩图的定义，会产生问题。对于广义串并联图的 Top Tree T 上的一个连通子图 T'，记 T' 中深度最浅的点为 r。之前，我们直接删掉了所有不在 r 子树内的边，但是现在可能会存在一些路径，通过 r 子树外的边连接 r 对应簇的两个界点。这样，删掉 r 子树外的边就导致了我们少算了一类路径。

少算的路径的形式是，从 r 对应簇中一个点出发，走到 r 的一个界点，再通过 r 子树外的部分走到 r 的另一个界点，然后从这个界点走到 r 对应簇中的另一个点。我们找到 r 子树外的部分在原图中构成的子图，选取其中和 r 的两个界点均连通的子图，它一定也是一个广义串并联图，且一定可以缩成一个以 r 的两个界点为界点的簇。如果我们的维护的信息允许，我们可以用这样一个簇来等效替代 r 子树外的部分。

我们在分治时，设当前在处理 T 这一子图，选择点 r 进行分治，r 的子树记作 T'。对于 T' 这一子图，在向下递归时需要添加 r 子树外所有边等效替代成的簇。由于 T 中最浅点的子树以外的子图已经被等效替代成了一个簇，我们直接用这个簇等效替代 T 中最浅点的子树以外的子图，这样计算用于等效替代 r 子树外所有边的簇时，就只需计算规模为 $\Theta(|T - T'|)$ 的图等效替代成的结果。最终的复杂度仍为 $\Theta(|V| \log |V|)$。

[6]https://loj.ac/p/3076

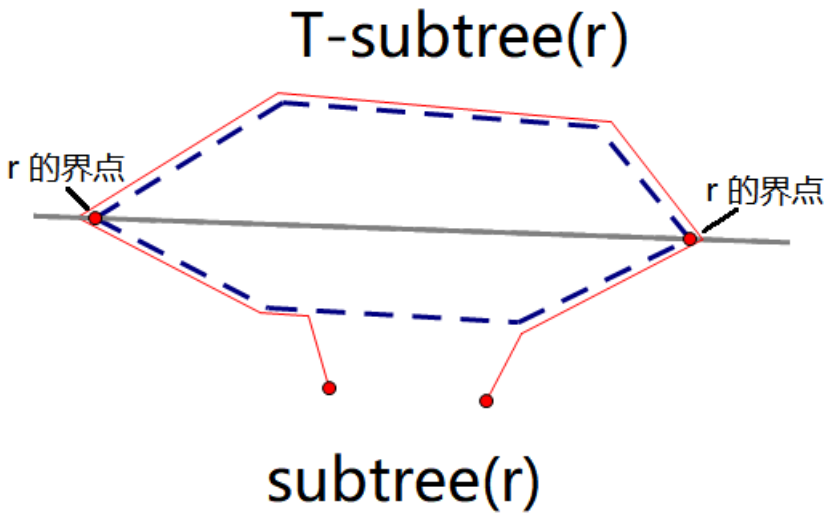


图 6: 通过 r 子树外的边的路径

例题 4.4.1. 【集训队互测 *2021*】逛公园 [7]

给定一张 n 个点的边带权的广义串并联图 G。有 q 次查询，每次给定一个点集，查询点集中的点的两两最短路之和对 2013265921 取模后的结果。

$n, q \le 10^5$，点集大小 $\sum k$ 之和不超过 10^5。

考虑离线处理所有询问。

建出 G 的 Top Tree T。对 T 进行点分治。

考虑每次选择了点 r，将当前子图 T 分成了以 r 为根的子树 T' 和 $T - T' + \{r\}$。设 r 的界点分别为 u, v。首先在 T 对应的簇内分别计算出以 u, v 为源点的单源最短路。

对每个询问依次处理。若 u, v 本身属于该询问的点集，则可以直接计算它们的贡献。现在考虑一个属于 T' 的点 x 和一个属于 $T - T' + \{r\}$ 的点 y，它们的贡献是 $x - u - y$ 和 $x - v - y$ 两条路径中较短的一条。

对于每个点，我们计算出它到 u 的距离和到 v 的距离的差值并排序。按照这个值从小到大枚举 x，则取 $x - u - y$ 这条路径的长度作为贡献的 y 是一个前缀，且这个前缀越来越短。使用 Two Pointers 维护即可。

最后考虑如何将 $T - T' + \{x\}$ 等效替代成一个簇。由于我们只关心最短路，我们可以直接将 $T - T' + \{x\}$ 等效替代成一个界点分别为 u, v 的，u, v 间距离为 $T - T' + \{x\}$ 中 u, v 最短路的簇。于是可以继续向下递归。

[7] https://uoj.ac/problem/598

最后的时间复杂度为 $\Theta(n\log^2 n + \sum k\log n)$。

4.5 广义串并联图分块的构建

最后，我们尝试利用 Top Tree 构建一种广义串并联图分块。

考虑对广义串并联图的 Top Tree 进行树分块。根据之前对 Top Tree 子图的收缩图的定义，求出对 Top Tree 分块得到的每个簇对应的收缩图，称为广义串并联图上的一个块，我们发现这些块有如下的性质：

- 块的数量为 $\Theta(\sqrt{|V|})$。
- 每块中的点数和边数均为 $\Theta(\sqrt{|V|})$。
- 每块可以收缩成这个块对应的 Top Tree 上的簇中最浅的点对应的簇，我们称这个簇的两个界点为这个块的外界点。
- 原图中每个点最多在一个块中作为非外界点存在，原图中每条边正好在一个块中存在。
- 每个块 C 至多只和两个块在不是 C 的外界点处有交，且这两个块在分块后的 Top Tree 上是 C 的儿子。因为这两个块及其子树能够立即收缩成一个簇，所以交点至多只有两个，我们把这至多两个点称为这个 C 的内界点。也就是说，每个块至多有四个界点。

于是我们完成了对广义串并联图的分块。

例题 4.5.1. [8]

给定一张 n 个点的广义串并联图 G，边带权。有 q 次操作，每次操作是以下两种之一：

- `1 i c`：将第 i 条边的边权修改为 c。
- `2 x d`：查询有多少个点到 x 号点的带权最短路长度不超过 d。

$q \le 3\times 10^4, n \le 10^5$。

首先对 G 建出 Top Tree，进行广义串并联图分块。

我们先考虑查询操作。我们讨论 x 和某个点 y 之间的最短路形式，这条路径要么没有经过任何界点，要么经过了某个界点。

[8]原创题

若 x,y 之间的最短路没有经过任何界点，那么 x,y 一定在同一个块中。我们可以直接在 x 所在块中跑单源最短路。注意即使 x,y 在同一个块中，最短路也有可能经过了某个界点，所以对于和 x 在同一个块中的 y，我们应该用下面的方法把经过了某个界点的最短路也求出来再计算答案，这样的 y 只有 $\Theta(\sqrt{n})$ 个。

若 x,y 之间的最短路经过了某个界点，我们就可以将路径分为三段（每段都可能退化为一个点）：从起点 x 到某个界点 u，途中不经过其它界点。从 u 到另一个界点 v。从 v 到终点 y，途中不经过其它界点。

对于一个 x，若它是一个界点，那么路径的第一段就退化成了一个点，有唯一的 $u=x$。否则，u 只能是 x 所在块的四个界点中的一个。

由于界点个数只有 $\Theta(\sqrt{n})$ 个，我们可以尝试计算出 x 到每个界点的最短路。具体地，我们以所有的 u 作为带距离的起点，在一张只含有界点的图 G' 上跑最短路，即可得到每个界点作为 v 时的最短路。这张图 G' 必须满足任意两个界点在 G' 上的最短路等于它们在 G 上的最短路，我们先解决当前的问题，再给出 G' 的具体建法。

得到每个界点作为 v 时的最短路后，我们就要统计有多少个合法的 y 了。对于一个 y，若它是一个界点，那么路径的第三段就退化成了一个点，有唯一的 $v=y$，于是我们可以直接统计答案。否则，v 只能是 x 所在块的四个界点中的一个，我们可以把 y 合法的条件写成存在一个 y 所在块中的界点 v，使得 v 到 y 的距离和 v 到 x 的距离之和不超过 d。考虑离线，对每个块记录下从每个查询的 x 走到这个块的四个界点后分别还能走多远，把这四个值作为这个查询 (x,d) 的四维坐标。然后求出这四个界点中每个到这个块中每个点 y 有多远，作为这个 y 的四维坐标。那么一个 y 对一个查询 (x,d) 没有贡献，当且仅当 y 对应四维坐标均大于 (x,d) 对应的四维坐标。于是四维数点即可。

接下来考虑如何建出 G'。对于两个界点 u,v，若它们之间的最短路经过其它界点 w，只要我们满足了 G' 上 u,w 间最短路等于 G 上 u,w 间最短路，G' 上 v,w 间最短路等于 G 上 v,w 间最短路，G' 上 u,v 间最短路就会等于 G 上 u,v 间最短路。所以我们只需使得所有最短路不经过其它界点的界点对 u,v 合法即可。

对于每个块，其最多有四个界点，我们要计算出只经过这个块内的边，它们两两间的最短路，并连上对应的边。这样每个块只会贡献 $\Theta(1)$ 条边，G' 的总边数为 $\Theta(\sqrt{n})$。

接下来考虑如何修改。考虑修改的边所在的块，由于这个块的边权发生了变化，我们需要立即处理之前在这个块中离线下来的询问。然后重新更新这个块对 G' 贡献的边的边权即可。

若使用高维分治的方法解决四维数点，时间复杂度为 $\Theta((q\sqrt{n}+n)\log^3 n)$；若使用 bitset 的方法解决四维数点，时间复杂度为 $\Theta(\frac{nq}{w}+q\sqrt{n}\log n)$，其中 w 为计算机字长。理论复杂度前者更优，但在当前计算机运行速度所能接受的数据范围下，后者的运行速度更快。

总结

本文介绍了静态 Top Tree 在树上问题和广义串并联图上问题中的一些应用。在本文的最后，笔者提出了一种基于 Top Tree 的构建广义串并联图分块的算法。希望本文能够激发读者的思考，起到抛砖引玉的作用。也希望大家能够在这一方面继续研究，取得新的成果。

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李曙老师、张超老师、史钋镭老师对我的教导。
感谢父母对我的关心和支持。

参考文献

[1] Werneck R F . Design and Analysis of Data Structures for Dynamic Trees[D]. Princeton University. 2006.

[2] Tarjan R E , Werneck R . Self-adjusting top trees[C]. Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005. ACM, 2005.

[3] 赵雨扬.《Top tree 相关东西的理论、用法和实现》, 2019.https://uoj.ac/blog/4912

[4] 周欣. 《浅谈一类树分块的构建算法及其应用》，IOI2021 中国国家队候选队员论文集。

[5] 吴作同.《〈公园〉命题报告》, IOI2019 中国国家候选队论文集。

[6] 林昊翰.《〈逛公园〉命题报告》, IOI2021 中国国家候选队论文集。

超立方体（Hypercube）及其相关算法初探

华东师范大学第二附属中学 管晏如

摘 要

超立方体，即 Hypercube，是一种特殊的无向图。其定义简洁，具有优美的结构。本文梳理了超立方体的多种定义方式和性质，探讨了实质与超立方体有关的算法，并讨论了与信息学密切相关的例题和模型。希望本文能起到抛砖引玉的作用，吸引读者进一步学习和探究超立方体上的问题。

1 引言

超立方体，是一种包含 2^n 个节点、$2^{n-1}n$ 条边的无向图，具高度的对称性。其在数学界的研究可以追溯到上世纪八十年代，也曾出现在 2013 年的江苏省选和 2021 年的欧洲初中生信息学竞赛（eJOI）中，但是至今并不为选手们熟知。

笔者借本篇论文的机会，梳理和总结了超立方体的性质，挖掘了其在信息学竞赛中的应用价值。本文的结构如下：

第二节介绍了超立方体的多种定义方式与结构特点。

第三节介绍了超立方体作为一般无向图所具有的性质。

第四节介绍了超立方体作为偏序集，在 Dilworth 定理与莫比乌斯反演中的应用。

第五节讨论了超立方体包含的一些特殊子图，例如环、网格等。

第六节讨论了超立方体上 Magic Labeling 的存在性和构造问题。

第七节为总结。

2 定义

本节我们介绍超立方体的基本定义。

2.1 记号约定

定义 K_n 表示一个包含 n 个点的完全图。

定义 C_n 表示一个包含 n 个点、n 条边的环。

定义 P_n 表示一条包含 n 个节点的链（或路径）。

定义 $N(v)$ 表示一个无向图中，v 的邻居集合。

定义 $N(V) = \cup_{v \in V} N(v)$。

定义两个集合 A, B 的笛卡尔积 $A \times B = \{(x, y) | x \in A, y \in B\}$。

定义两个简单无向图 $G_1 = (V_1, E_1)$ 和 $G_2 = (V_2, E_2)$ 的笛卡尔积 $G_1 \times G_2 = (V, E)$，其中 $V = V_1 \times V_2$，$E = \{((x, y), (z, w)) | x = z, (y, w) \in E_2 \text{或者} (x, z) \in E_1, y = w\}$。易见笛卡尔积具有结合律（假如不考虑多元组内部括号的话）。

2.2 超立方体的直接定义

对于 $n \in Z_+$，定义无向图 $Q_n = \{V, E\}$，其中 $V = \{0, 1, ..., 2^n - 1\}$,$E = \{(x, y) | popcount(x \oplus y) = 1, x, y \in V\}$。

换句话说，两个点有连边当且仅当它们的编号在二进制下恰有一位不同 $(*)$。我们称这样得到的 Q_n 为 **n 阶超立方体**（又称 Hypercube）。

容易发现，$\forall x \in V, deg_{Q_n}(x) = n$，因而 $|E| = \frac{1}{2} n|V| = 2^{n-1} n$。

下图 1 是 $n = 1, 2, 3$ 时的图例，可以看到 Q_1 同构于 K_2，Q_2 同构于 C_4，Q_3 则是一个三维立方体骨架。

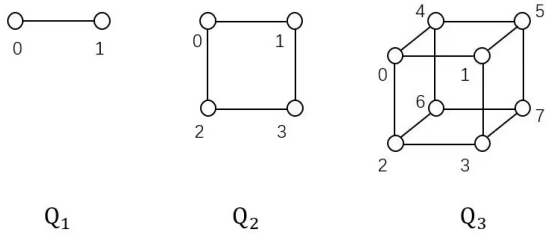


图 1

在后文，我们称一条连接 x 与 $x \oplus 2^k$ 的边具有“方向”k（共 n 种不同的方向）。

2.3 超立方体的归纳定义

我们使用图论里笛卡尔积的概念，来对超立方体进行归纳的定义：

Lemma 2.3.1：令 $Q_1 = K_2$，$Q_n = Q_{n-1} \times K_2$。则这种定义方式与 2.2 中的直接定义是等价的。

证明: 归纳。假设 Q_{n-1} 已经是一个 $n - 1$ 阶的超立方体，可以用 0 至 $2^{n-1} - 1$ 标号，使得连边符合 $(*)$。那么根据 $Q_n = Q_{n-1} \times K_2$，Q_n 可以分成两个完全同构于 Q_{n-1} 的部分 G_0, G_1。

我们令 $\forall v \in V_{G_0}$ 的标号为 v，$\forall v \in V_{G_1}$ 的标号为 $v + 2^{n-1}$，则容易验证 G_0、G_1 内部连边仍然符合 $(*)$，G_0 与 G_1 之间只有 v 与 $v + 2^{n-1}$ 有边，同样符合 $(*)$。 □

将上述结论展开，可得 $Q_n = K_2 \times K_2 \times ... \times K_2$（共 n 个 K_2）。

2.4 结构视图

通过 2.3 节的定义，我们已经获得了如下图 2 的一种视图方式。事实上，我们可以选择二进制下的任意一位进行分层（不必要是第 $n-1$ 位），也即令 $G_{i,k}$ 为 $V_{i,k} = \{v \in V | v$在二进制下的第i位 $= k\}$ 的导出子图（$0 \le i < n, k = 0, 1$）。

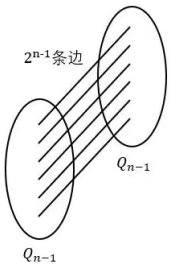


图 2

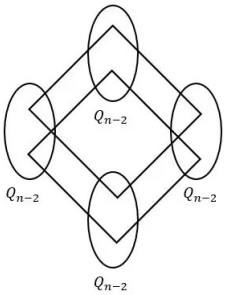


图 3

同时，我们还有推论 $Q_n = Q_{n-m} \times Q_m$。以 $m = 2$ 为例，Q_n 可以被看作由 4 个 Q_{n-2} 组成，不同 Q_{n-2} 之间的边即为 2^{n-2} 个 $Q_2 = C_4$，如上图 3 所示。易见图 2 对应于 $m = 1$ 的情形。

下面我们再给出一种视图方式。将 Q_n 中的点 $0 \le v < 2^n$ 按照 $popcount(v)$ 划分成 $n + 1$ 层，则：

1. 第 i 层有 $\binom{n}{i}$ 个点；
2. 只有相邻的层之间有边；
3. 第 i 层每个点向 $i - 1$ 层连 i 条边，向 $i + 1$ 层连 $n - i$ 条边。

在本文中我们称这种视图方式为“球形”视图，并约定 S_i 表示第 i 层的点（$0 \le i \le n$）所构成的点集。下图 4 是 $n = 4$ 时的情形。

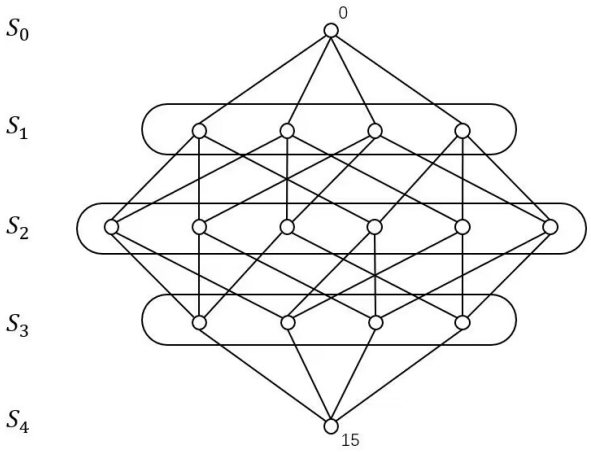


图 4

3 无向图上的性质

本节我们讨论超立方体作为无向图时所具备的性质。

3.1 基本性质

1. 二分图

将所有点按照 *popcount* 的奇偶性，染成黑色和白色，则任意一条边必然连接了一对黑点和白点。这说明 Q_n 是连通二分图。

2. 直径

无向图直径的定义是任意两点间最短距离的最大值。$\forall x, y \in V$，$dist(x, y) = popcount(x \oplus y)$，因而 Q_n 的直径 $= n$。

3. 最大独立集

一方面，取所有 *popcount* 为偶数（或奇数）的点，构成一个大小为 2^{n-1} 的独立集。

另一方面，若选出的点个数 $> 2^{n-1}$，则根据抽屉原理，一定存在两个被选出的点，除了最低位以外都相同。那么它们相邻，与独立集矛盾。

故 Q_n 的最大独立集大小为 2^{n-1}。

4. 最小点覆盖

这里的最小点覆盖，指选取最少的点，覆盖所有的边。

一方面，取所有 *popcount* 为偶数（或奇数）的点，构成一个大小为 2^{n-1} 的点覆盖。

另一方面，Q_n 上共有 $n2^{n-1}$ 条边，每个点的度数为 n，故至少需选择 2^{n-1} 个点进行覆盖。

故 Q_n 的最小点覆盖大小也为 2^{n-1}。

5. 完美匹配

易见，我们选择 n 种方向中的任意一种，将 Q_n 分成上下两层，则两层之间的边会构成一组完美匹配。亦即将所有的 v 和 $v \oplus 2^i$ 进行匹配。

此外，在第 5 节中，我们将看到 Q_n 具有哈密尔顿回路，因而在哈密尔顿回路上选择相邻的点进行匹配，也可以构成一组完美匹配。这也说明完美匹配的方案是非常多样的。

6. 自同构性

Hypercube 具有良好的对称性。将所有点的编号同时异或 Δ（$0 \le \Delta < 2^n$）所得到的图仍然是一个 Q_n。

类似地，若给定一个长度为 n 的排列 $p_0, ..., p_{n-1}$，令：

$$f(v) = \sum_{i=0}^{n-1} [\text{v 在二进制下第} p_i \text{位是1}] 2^i$$

则 f 为 V 到 V 的一一映射。将所有点编号作用函数 f 后，仍然是一个 Q_n。

3.2 例 1. [JSOI2013] 超立方体

【题目大意】给定一张 N 个点 m 条边的无向图，判断是否同构于一个 Hypercube，如果是，给出一种符合条件 $(*)$ 的点标号方式。

【数据范围】$2 \le N \le 32768$。

【解法 1】首先，检查点数、边数、度数及连通性等基本性质是否合法。

下面假设 $N = 2^n, m = 2^{n-1}n$ 且每个点的度数均为 n。

根据自同构的性质，我们可以假定任意一点的标号是 0，并以任意顺序假定 0 的邻居为 $2^0, 2^1 ..., 2^{n-1}$。

随后，从点 0 出发作 BFS，得到其余每个点到 0 的最短距离 $dist(0, v)$。依据 $dist(0, v)$ 进行分层，即得到“球形”视图结构。下面假定该图仍符合“球形”结构的若干基本特征。

注意到，当 $2 \le k \le \lfloor \frac{n}{2} \rfloor$ 时，$\forall v \in S_k$，v 的标号应当为满足 $u \in S_{k-1} \cap N(v)$ 的 u 标号的超集。进一步地，若图合法，则 v 的标号恰好 $= S_{k-1} \cap N(v)$ 内所有点标号的并集（按位或）。

因此，我们令 k 依次取 $2, 3, ..., \lfloor \frac{n}{2} \rfloor$，即完成了第 0 至 $\lfloor \frac{n}{2} \rfloor$ 层的标号。

而余下的部分是完全对称的，可以类似地从第 $n-2$ 层往第 $\lceil \frac{n}{2} \rceil$ 层标号。

需注意，因为我们已经按一定的顺序确定了 $2^0, ..., 2^{n-1}$，所以 $2^n - 1$ 的所有邻居，即 $2^n - 1 - 2^0, ..., 2^n - 1 - 2^{n-1}$ 的顺序不再是任意的了。可以从每个 2^i 节点出发作 BFS，找到最短距离 $= n$ 的点，标为 $2^n - 1 - 2^i$。如果这样的点不是恰好一个，那么显然也是不符合超立方体的“球形”结构的。

最后，需要扫描所有边进行检查，检查通过才能断定是 Hypercube。

时间复杂度 $O(m \cdot n)$。

事实上，当 $k \ge \lceil \frac{n}{2} \rceil$ 时，取 v 的标号为 $S_{k-1} \cap N(v)$ 内所有点标号的并集仍然是正确的。这样直接从上至下完成标号，时间复杂度 $O(m)$。

【解法 2】令 $ans[i]$ 表示填写了标号 i 的节点。同上述解法，不妨 $ans[0]$ 与 $ans[2^i](0 \le i < n)$ 均已确定。下面我们按 i 从小到大的顺序确定剩余的 $ans[i]$。

考虑找到两个在原图中应是 $ans[i]$ 邻居的点 $x = ans[j_1], y = ans[j_2]$，满足 $0 \le j_1 \ne j_2 < i, popcount(j_1 \oplus i) = popcount(j_2 \oplus i) = 1$。根据假设，$popcount(i) > 1$，因此 j_1, j_2 存在。

那么在原图中，$ans[i]$ 应当是 x, y 的公共邻居，并且尚未被标过号（即标号 $\ge i$）。下面我们说明，这样的点是唯一的。

注意到 j_1, j_2 的生成方式为，将 i 中某一位 1 置成 0。不妨假设是最低的两位，则 $i \equiv 3 \pmod 4$，$j_1 = i - 1, j_2 = i - 2$。

记 $pre = i - 3$，则可以发现 j_1, j_2 有两个公共邻居：$pre + 3$ 和 pre。显然 $pre < i$，已经被确定过，故 $i = pre + 3$ 是唯一未填的公共邻居。

而若前面的每一步都是唯一的选择，当前出现了不存在或不唯一的情况，就说明原图不是 Hypercube。

时间复杂度 $O(2^n \cdot n)$ 也即 $O(m)$。

事实上，上面的讨论能导出更一般的如下结论：

1. 对于 $x, y \in V, x \ne y$，若 $popcount(x \oplus y) = 2$，则 x, y 恰有两个公共邻居，否则没有公共邻居。

2. Q_n 上共有 $n!2^n$ 种不同的点标号方式满足条件 $(*)$。

3.3 例 2. [CF1543 E] The Final Pursuit

【题目大意】在例 1 的基础之上，给 Q_n 的每个点染 n 种颜色之一，使得每个点的邻居颜色各不相同。输出染色方案，或报告无解。

【数据范围】$1 \le n \le 16, 2 \le \sum 2^n \le 2^{16}$。

【解法】我们用数 0 到 $n - 1$ 表示 n 种颜色，并不妨假设 Q_n 的标号已经是符合 $(*)$ 的了。

由每个点的度数都是 n，可知每个点的邻居颜色恰好取遍 $0, 1, ..., n - 1$。利用算两次原理不难发现，每种颜色恰好出现 $2^n/n$ 次。这要求 $n|2^n$，即 $n = 2^k (k \in Z_+)$。若 n 不是 2 的幂，需报告无解。

现在，我们令染色方案如下

$$col(v) = \oplus_{v\text{在二进制下第}i\text{位是}1} i$$

则由 $\forall u \in N(v), u$ 取遍 $v \oplus 2^i (0 \le i < n)$，知 $\forall u \in N(v), col(u)$ 取遍 $col(v) \oplus i (0 \le i < n = 2^k)$。这样 v 的所有邻居颜色就取遍 $0, 1, ..., 2^k - 1$ 了。

4 偏序集上的性质

本节我们介绍将超立方体看作偏序集后的性质。

4.1 偏序集

首先，我们给出偏序集的概念。

对于集合 V 上的一个关系 $A \subseteq V \times V$，记 $x \preceq y \Leftrightarrow (x,y) \in A$，则我们称 A 为偏序关系，当且仅当 A 满足：

1. 自反性，即 $\forall x \in V, x \preceq x$；

2. 反对称性，即 $\forall x,y \in V, x \preceq y, y \preceq x \Rightarrow x = y$；

3. 可传递性，即 $\forall x,y,z \in V, x \preceq y, y \preceq z \Rightarrow x \preceq z$。

偏序关系可以简记为 (V,A)。除了一条链和空图以外，常见的偏序有 $(\mathrm{R}, \leq), (Z_+, |)$ 等。

在超立方体上，我们可以定义 $x \preceq y \Leftrightarrow$ 在二进制下 x 是 y 的子集。容易验证这是一组偏序关系。形式化地说，$(V_{Q_n}, \subseteq)$ 是一组偏序关系。

有时为了方便讨论，我们也会把超立方体的顶点直接看作是 $0, 1, .., n-1$ 的子集。即将顶点集看作集合族 $2^{\{0,1,...,n-1\}}$。

超立方体本身的边是上述偏序关系的 Hasse Diagram[1]，即本身构成骨架，只有添加了所有传递闭包内的边，才能称作是严格的偏序集。

在这一节，我们重点讨论偏序集 $(2^{\{0,1,...,n-1\}}, \subseteq)$ 上的问题。超立方体的“球形”结构仍然适用，并且将起到非常重要的作用。

4.2 Dilworth 定理

对于偏序关系 (V,A)，称子集 $V' \subseteq V$ 为**反链**，当且仅当不存在 $x,y \in V', x \neq y$ 满足 $x \preceq y$。

定义**最大反链**为 $|V'|$ 最大的反链 V'。

称 V 的一种划分 $V_1, V_2, ..., V_k$ 是**链覆盖**，当且仅当 $\forall 1 \leq i \leq k$，V_i 中的元素 $d_1, d_2, ..., d_t$ 可以以一定的顺序排列，满足 $d_1 \preceq d_2 \preceq ... \preceq d_t$。

定义**最小链覆盖**为 k 最小的链覆盖方案。

Dilworth 定理[2]告诉我们，最大反链与最小链覆盖的大小相等。

4.3 超立方体上的 Dilworth 定理

4.3.1 最小链覆盖

我们先构造超立方体上的最小链覆盖。注意到“球形”结构中的每一层都构成一个反链，反链中的元素必然不能位于同一条链上。因此最小链覆盖 $\geq |S_{\lfloor \frac{n}{2} \rfloor}|$。

[1]Hasse Diagram,https://en.wikipedia.org/wiki/Hasse_diagram

[2]详见参考文献 [8]

Lemma 4.3.1：对于 $0 \le i \le n-1$，S_i 与 S_{i+1} 之间所构成的二分图，有关于点数较少一边的完美匹配[3]。

证明：根据第 2.4 节中的性质，该二分图中，S_i 一侧每个点的度数均为 $n-i$，S_{i+1} 一侧每个点的度数均为 $i+1$。故这是一个正则二分图，满足 Hall 定理的条件，因而存在关于点数较少一边的完美匹配[4]。 □

现在，考虑从第 $\lfloor \frac{n}{2} \rfloor$ 层出发，初始时令 $S_{\lfloor \frac{n}{2} \rfloor}$ 内的每个元素单独构成一条链，然后往第 0 层的方向，进行链的拓展。根据 **Lemma 4.3.1**，可以覆盖到上半部分的所有点。

对称地，从第 $\lceil \frac{n}{2} \rceil$ 层出发，可以往下覆盖到剩余的所有点。

当 n 是奇数时，$S_{\lfloor \frac{n}{2} \rfloor}$ 与 $S_{\lceil \frac{n}{2} \rceil}$ 之间也存在完美匹配。故大小为 $|S_{\lfloor \frac{n}{2} \rfloor}|$ 的链覆盖构造完成。

另一方面，在上述覆盖方案中，一个反链与每条链至多一个公共元素。因此 $S_{\lfloor \frac{n}{2} \rfloor}$ 与 $S_{\lceil \frac{n}{2} \rceil}$ 都是取到大小最大值的反链。

4.3.2 最大反链的另证

事实上，如果只是证明最大反链的上界，最小链覆盖方案不是必要的。

Theorem 4.3.2（Sperner 定理）：$(Q_n, \subseteq)$ 的最大反链 $\le \binom{n}{\lfloor \frac{n}{2} \rfloor}$。

证明：假设 Q_n 上的一个反链 $S = \{v_1, v_2, ..., v_m\}$，$\forall 1 \le i \le m, v_i \subseteq \{0, 1, ..., n-1\}$。令 $F(v_i) = \{\pi | \{\pi_0, \pi_1, ..., \pi_{|v_i|-1}\} = v_i\}$，其中 π 是 0 到 $n-1$ 的排列。

则由 S 反链的条件，所有 $F(v_i)$ 应当无交 $\Rightarrow \sum_{i=1}^{m} |F(v_i)| \le n!$。

另一方面，$|F(v_i)| = (n-|v_i|)!(|v_i|)!$，在 $|v_i| = \lfloor \frac{n}{2} \rfloor$ 或 $\lceil \frac{n}{2} \rceil$ 时取到最小值。

从而，$m \le \frac{n!}{(n-\lfloor \frac{n}{2} \rfloor)!(\lfloor \frac{n}{2} \rfloor)!} = \binom{n}{\lfloor \frac{n}{2} \rfloor}$。 □

上述证明与参考文献 [3] 是基本等价的。Sperner 定理还有很多其他的证明方法和推广[5]，这里不展开叙述。

4.4 莫比乌斯反演

4.4.1 莫比乌斯反演的概念

对于偏序集 (V, A)，记 $\mathcal{A} = \{f : V \times V \to R | \forall x, y \in V, f(x, y) \ne 0 \Rightarrow x \le y\}$。函数 f 也可看作是大小为 $|V| \times |V|$ 的实数矩阵（这里仅讨论 $|V|$ 有限的情况）。

易见，全 0 矩阵 $\in \mathcal{A}$，$I \in \mathcal{A}$。

Lemma 4.4.1：若矩阵 $\alpha \in \mathcal{A}$ 可逆，则 $\alpha^{-1} \in \mathcal{A}$。

证明：将 V 中元素按拓扑序排列，可不妨假设 α 是上三角矩阵。

则 α 可逆说明 α 的主对角线上均非 0，且 α^{-1} 也为上三角矩阵。

[3] 具体构造详见参考文献 [9]

[4] https://www.sfu.ca/ mdevos/notes/graph/345_matchings.pdf

[5] Sperner's theorem,https://en.wikipedia.org/wiki/Sperner's_theorem

而对于 $\forall x,y\in V,x\not\leq y$，我们希望找到一个合法的拓扑序，满足 y 在 x 前面。这可以通过添加 y 到 x 的边来做到。

因而在这种拓扑序下，x,y 被变换到了 α^{-1} 的下三角区，即 $\alpha^{-1}(x,y)=0$。

对于不同的 $x\not\leq y$，拓扑序可能不同，但结论是相同的。从而，$\alpha^{-1}\in\mathcal{A}$。 □

记 (V,A) 的特征值矩阵 Z 为 $Z(x,y)=[x\preceq y]$，易见 $Z\in A$。根据 **Lemma 4.4.1**，知 $\mu=Z^{-1}\in A$。这里得到的 μ 就是我们所熟知的莫比乌斯反演。

4.4.2 莫比乌斯矩阵的性质

依据 $Z\times\mu=I$，对于 $x,y\in V$ 有：

$$\sum_{x\preceq z\preceq y}\mu(z,y)=[x\preceq y]$$

(**)

进一步地，$\mu(x,x)=1$，并由 **Lemma 4.4.1** 可知，当 $x\not\leq y$ 时 $\mu(x,y)=0$。另一种说明方法是：若不然，可固定 y，取 $x\not\leq y$ 为满足 $\mu(x,y)\neq 0$ 的极大元，那么由 (**) 知 $\exists x\prec z\not\leq y,\mu(z,y)\neq 0$，矛盾。

根据 (**) 式，我们可以按照 $dist(x,y)$ 从小到大的顺序，填写 $\mu(x,y)$。

μ 的意义在于，对于任意函数 $f:V\to R$，若 $g(x)=\sum_{y\preceq x}f(y)$，则 $f(x)=\sum_{y\preceq x}\mu(y,x)g(y)$。有时，直接求解 f 较为困难，可以先求出 g，再利用莫比乌斯反演来求 f。

一种常见的应用便是在求解数论函数方面，对应于偏序集 $(Z_+,|)$。事实上，如果把 V 限定为某个正整数 n 的所有约数（$n=\prod_{i=1}^{k}p_i^{\alpha_i}$），那么该偏序集结构同构于 $P_{\alpha_1+1}\times P_{\alpha_2+1}\times...\times P_{\alpha_k+1}$。这说明数论在某种意义上是偏序集的特例。

4.5 超立方体上的莫比乌斯反演

在超立方体 Q_n 上，注意到集合 A 的传递闭包同构于 $Q_{n-|A|}$，这表明 $\forall A\subseteq B$，有 $\mu(A,B)=\mu(\emptyset,B\setminus A)$。因而我们只要考虑所有的 $\mu(\emptyset,?)$。

按照 $i=1,2,...,n$ 的顺序，填写 $\mu(\emptyset,A)(A\in S_i)$。由对称性，知每一层上的 $\mu(\emptyset,A)$ 都是相等的。不妨简记为 $\mu(\emptyset,S_i)$。

根据 $\sum_{j=0}^{i}(-1)^j\binom{i}{j}=0$ 的恒等式（$i>0$），我们容易归纳证明 $\mu(\emptyset,S_i)=(-1)^i$。

一般地，当 $A\subseteq B$ 时，$\mu(A,B)=(-1)^{|B|-|A|}$。

这揭示了我们常用的容斥原理的实质。

5 超立方体包含的子图

本节我们讨论超立方体所包含的一些特殊子图。

5.1 环

Lemma 5.1.1: Q_n 有哈密尔顿回路。

证明: Q_n 上的哈密尔顿回路，其实是我们所熟悉的格雷码[6]（又称 Gray Code）。

这里简述一种较为常见的格雷码构造方法。设 G_n 表示长度为 2^n 的格雷码，$G_1 = \{0, 1\}$。

$\forall n \geq 2$，我们令 G_n 为如下序列：（$G_{n-1}(x)$ 表示 G_{n-1} 的第 x 项，从 0 开始）

$$G_{n-1}(0), G_{n-1}(1), ..., G_{n-1}(2^{n-1}-1), G_{n-1}(2^{n-1}-1)+2^{n-1}, ..., G_{n-1}(0)+2^{n-1}$$

例如，$G_2 = \{0, 1, 3, 2\}, G_3 = \{0, 1, 3, 2, 6, 7, 5, 4\}$。容易验证这样得到的序列为排列，且满足相邻两项（包括首尾）在二进制意义下恰有一位不同。 □

这样一来，在 Q_n 中就可以找到任意长度 $< 2^n$ 的简单路径了。

那么，是否任意长度 $< 2^n$ 的简单环都可以在 Q_n 中找到呢？显然，奇数是不可能的，因为 Q_n 是二分图。而对于偶数，答案是肯定的：

Lemma 5.1.2: $\forall 4 \leq len \leq 2^n$，$2|len$，在 Q_n 中可以找到一个简单环 C_{len}。

证明: 记 $l = len/2$。取 G_{n-1} 的前 l 项，构成序列 $d_0, d_1, ..., d_{l-1}$，则

$$d_0, d_1, ..., d_{l-1}, d_{l-1}+2^{n-1}, ..., d_1+2^{n-1}, d_0+2^{n-1}$$

是一个长度为 len 的简单环。 □

5.1.1 例 3.

关于哈密尔顿回路的构造，我们有更强的结论：

【题目大意】给定一组 Q_n 上的完美匹配（称为红边），要求选择另一组完美匹配（称为蓝边），使得红蓝边交替构成哈密尔顿回路。可以证明当 $n \geq 2$ 时一定有解。

【解法】首先，我们把命题加强为，红边连接的两点不要求在 Q_n 上有边相连。即看作给定 2^{n-1} 对点两两配对。记点 v 与 $mat[v]$ 配对。

对 n 归纳。容易验证 $n = 2$ 的情形。下面假设 $n-1$ 时结论成立。

把 Q_n 分成两层 Q_{n-1}，记作 G_0 和 G_1。令 $S_i = \{v | v \in G_i, mat[v] \in G_{i\oplus 1}\}, T_i = \{(v, mat[v]) | v, mat[v] \in G_i\}$。易知 $|S_0| = |S_1|$ 为偶数。

先考虑 $|S_0| > 0$ 的情况。我们把 S_0 中的点任意两两配对，然后结合 T_0 的配对，在 G_0 上应用归纳假设，得到一个由配对和蓝边交替构成的环 C_0。则可设 S_0 中的配对边，在 C_0

[6][CSP-S 2019] 格雷码

上的出现顺序为 $s_0 - s_1 - - - s_2 - s_3 - - - \ldots - - - s_{t-2} - s_{t-1} - - - s_0 - s_1$。其中 $- - -$ 表示一条可能经过多个点的路径，$-$ 表示配对。

接着，我们把 S_1 中的点两两配对：$mat[s_1]$ 与 $mat[s_2]$、$mat[s_3]$ 与 $mat[s_4]$ ... $mat[s_{t-1}]$ 与 $mat[s_0]$，结合 T_1 的配对，在 G_1 上应用归纳假设。注意此时 S_1 中的配对情况与前面获得的构造 C_0 有关。记此时 G_1 内获得的构造为 C_1，那么把 C_0 内 S_0 之间的配对，以及 C_1 内 S_1 之间的配对，全部断开，加入 S_0 与 S_1 之间原本的 $(v, mat[v])$，即得一组 Q_n 上的解。

而倘若 $S_0 = S_1 = \emptyset$，注意到最初划分成 G_0, G_1 两层的方向是任意的，故对于任意一对 $(v, mat[v])$，任选二进制下 $v \oplus mat[v]$ 为 1 的一位进行分层，即可使 $|S_0| > 0$。 □

5.2 网格

这里讨论的“网格”，是指 $P_{x_1} \times P_{x_2} \times \ldots \times P_{x_k}$。

令 $y_i = \lceil \log_2 x_i \rceil$，则当 $n \geq \sum_{i=1}^{k} y_i$ 时，可以直接沿用格雷码的结果：网格中的点 $(\alpha_1, \alpha_2, \ldots, \alpha_k)$（$\forall 1 \leq i \leq k, 0 \leq \alpha_i < x_i$）对应到 Q_n 上的点

$$(G_{y_1}(\alpha_1))_2 (G_{y_2}(\alpha_2))_2 \ldots (G_{y_k}(\alpha_k))_2$$

即二进制下顺次拼接。

下面这道例题则要求更为精细的结果。虽然它只考察了二维的情形，但从二维到多维的推广是相对容易的。

5.2.1 例 4. [eJOI2021] 抄作业

【题目大意】给定 n, m，要求给 $n \times m$ 的网格填上互不相同的非负整数，使得相邻（上下或左右）两数在二进制下恰有一位不同。

此外，整个网格上的最大数值需要取到最小。输出一种方案。

【数据范围】$1 \leq n, m \leq 2000$。

【解法】将 n, m 放大到最近的 2 的幂次是可行的填法，但是最大数值很可能不优。

事实上，网格的嵌入并不要求格雷码序列首尾合法，因此可以尝试构造任意长度、只有中间相邻合法的格雷码序列。在已知 Q_n 哈密尔顿回路的情况下，这是可以做到的：

Lemma 5.2.1: $\forall n \in Z_+$，存在 0 至 $n-1$ 的排列 $\{p_n\}$，满足 $\forall 1 \leq i < n, popcount(p_{i-1} \oplus p_i) = 1$。

证明: 设 $n = \sum_{i=1}^{k} 2^{d_i}, 0 \leq d_1 < d_2 < \ldots < d_k$。

构造排列 $\{p_n\}$ 为以下序列顺次拼接的结果，其中 $G_d + x$ 表示每一项都加 x。

$$
\begin{aligned}
&G_{d_1}\\
&G_{d_2}+2^{d_1}\\
&G_{d_3}+2^{d_1}+2^{d_2}\\
&\cdots\\
&G_{d_k}+\sum_{i=1}^{k-1}2^{d_i}
\end{aligned}
$$

为了确保序列拼接处合法，可令 i 是偶数时，G_{d_i} 以 0 开头、以 1 结尾，i 是奇数时，G_{d_i} 以 1 开头、以 0 结尾（$\forall 1 \le i \le k$）。根据 Hypercube 的对称性，这一定可以做到。当然，**Lemma 5.1.1** 的构造恰有 $0,1$ 相邻，可以直接循环移位。 □

接着，需要将 $\{p_n\}$ 与 $\{p_m\}$ 拼到一起，填进网格表。一种方法是如前所述的二进制直接拼接，但是仍然可能不优。

实际上，我们可以在二进制下选择 $\lceil \log_2 n \rceil$ 位填 $\{p_n\}$，余下的 $\lceil \log_2 m \rceil$ 位填 $\{p_m\}$，而不必要是前后缀的拼接。具体最优的选择方式可以通过二维 DP 实现，也可以从最高位开始贪心，每次选择剩余字典序最小的填写。

最后，我们说明这样做的正确性：

Lemma 5.2.2: 对于一个合法的方案以及二进制下的某一位 k，两个维度独立。

证明: 设一个合法的填法为 $A(i,j)$，其中 $0 \le i < n, 0 \le j < m$。

考察 $A(i,j), A(i+1,j), A(i,j+1), A(i+1,j+1)$，则它们互不相同，且 $A(i,j)$ 与 $A(i+1,j+1)$ 恰为 $A(i+1,j), A(i,j+1)$ 在 Hypercube 上的两个公共邻居。

这说明，$A(i,j) \oplus A(i+1,j) = A(i,j+1) \oplus A(i+1,j+1)$。

进一步地，$\forall 0 \le i < n-1, 0 \le j,k < m, A(i,j) \oplus A(i+1,j) = A(i,k) \oplus A(i+1,k)$。对于列来说也有类似的结论。

那么，对于行 i 可记 $H_i = A(i,j) \oplus A(i+1,j)$，对于列 j 可记 $L_j = A(i,j) \oplus A(i,j+1)$。易见 H_i 和 L_j 均为 2 的幂次。

假若结论不成立，即存在二进制下某一位 k 和行 i 列 j 使得 $H_i = L_j = 2^k$，则考察行 $i, i+1$ 与列 $j, j+1$ 的交界处：

$$A(i,j) = A(i+1,j) \oplus 2^k = A(i,j+1) \oplus 2^k = A(i+1,j+1)$$

矛盾。 □

6 Magic Labeling

本节我们讨论一种特殊的点（或边）标号方式，即 Magic Labeling[7]，在超立方体上的情况。这类问题完全可以被出在信息学竞赛中，一些解法本身也与算法有密不可分的关系。

6.1 基于点的 Magic Labeling

对于无向图 $G = (V, E)$，定义 $f : V \to \{0, 1, ..., |V| - 1\}$ 为一种基于点的 Magic Labeling，当且仅当 f 为一一映射，且存在 Δ 使

$$\forall v \in V, \sum_{(u,v)\in E} f(u) = \Delta$$

Lemma 6.1.1: 当 n 为奇数时，Q_n 上不存在基于点的 Magic Labeling。

证明: 假如存在，则

$$|V|\Delta = \sum_{v\in V} deg(v)f(v)$$
$$\Rightarrow \Delta = \frac{1}{2}(2^n - 1)n$$

n 是奇数时 $\Delta \notin Z$，矛盾。 □

同时，易见 $n = 2$ 时有解。下设 n 为 ≥ 4 的偶数。

Lemma 6.1.2: 当 $4|n$ 时，Q_n 上不存在基于点的 Magic Labeling。

证明: 假设存在，记 $F(i) = \sum_{v\in S_i} f(v)$。则在“球形”结构上考虑算两次，知：

$$\forall 1 \leq i < n, F(i-1) \cdot (n-i+1) + F(i+1) \cdot (i+1) = \Delta \cdot |S_i|$$

将 $i = 1$ 和 $i = n - 1$ 代入，可得 $F_0 = F_n \Leftrightarrow F_2 = F_{n-2}$。进一步地，

$$F_0 = F_n$$
$$\Leftrightarrow F_2 = F_{n-2}$$
$$...$$
$$\Leftrightarrow F_{2i} = F_{n-2i}$$

这表明，当 $4|n$ 时，$F_0 = F_n \Leftrightarrow F_{n/2} = F_{n/2}$。从而，$f(0) = f(2^n - 1)$，矛盾。 □

而余下的情形均有解：

Theorem 6.1.1: 当且仅当 $n \equiv 2 \pmod 4$ 时，Q_n 上存在基于点的 Magic Labeling。

[7]本文讨论的情形约束条件更强，因而在一些场合也被称作是 Super Magic Labeling。

证明: 以 $n = 6$ 为例，我们考虑构造一组 6 个长度为 6 的 01 向量，使得它们线性无关，且每一维度上恰好 3 个 0、3 个 1。记这 6 个向量看作二进制数后分别是 $v_0, v_1, ..., v_5$，则令

$$f(v) = \oplus_{v\text{在二进制下第}i\text{位}=1} v_i$$

由 $v_0, v_1, ..., v_5$ 线性无关，知所有 $f(v)$ 互不相同。同时，

$$\sum_{(u,v)\in E} f(u) = (2^0 + 2^1 + ... + 2^5) \cdot 3$$

故这样得到的 $f(v)$ 是一组合法的 Magic Labeling。

至于 $v_0, v_1, ..., v_{n-1}$ 的构造[8]，当 $n \geq 6, n \equiv 2 \pmod 4$ 时可以随机化 + 线性基验证。当 $4|n$ 时，由于要求每一维度上恰好 $n/2$ 个 1，可以发现 $\oplus_{i=0}^{n-1} v_i = 0$，必然线性相关，故构造不存在（与 **Lemma 6.1.2** 的结论吻合）。 □

6.2 基于边的 Magic Labeling

对于无向图 $G = (V, E)$，定义 $f : E \to \{0, 1, ..., |E| - 1\}$ 为一种基于边的 Magic Labeling，当且仅当 f 为一一映射，且存在 Δ 使

$$\forall v \in V, \sum_{(u,v)\in E} f(u, v) = \Delta$$

易见，Q_1 存在基于边的 Magic Labeling，Q_2 不存在。以下我们默认 $n > 2$。

Lemma 6.2.1: 当 n 为奇数时，Q_n 上不存在基于边的 Magic Labeling。

证明: 假如存在，则

$$\begin{aligned} 2^n\Delta &= m(m-1) \\ m &= n2^{n-1} \\ \Rightarrow \Delta &= \frac{n}{2}(n2^{n-1} - 1) \end{aligned}$$

当 n 是奇数时，$\Delta \notin Z$，矛盾。 □

Lemma 6.2.2: 若二分图 $G = (V, E)$ 由两个哈密尔顿回路叠加构成，则存在一种基于边的 Magic Labeling。

证明: 设 $n = |V|$，则由条件知 n 为偶数。称 G 拆分成的两个哈密尔顿回路为 C_0, C_1，并不妨 C_0 依次经过点 $0, 1, ..., n - 1$。

方便起见，记 $w(v) = \sum_{(u,v)\in E} f(u, v)$。令

[8]直接构造详见参考文献 [6]

$$\begin{aligned}&\forall 0 \le 2i < n, f(2i, 2i+1) = i\\&\forall 0 < 2i \le n, f(2i-1, 2i) = n-i\end{aligned} \tag{1}$$

其中编号 mod n。则当前的

$$\begin{aligned}&w(0) = n/2\\&\forall 1 \le i < n, i\text{为偶数}, w(i) = n\\&\forall 1 \le i < n, i\text{为奇数}, w(i) = n-1\end{aligned} \tag{2}$$

下图 5 展示了 $n = 8$ 时的结果。

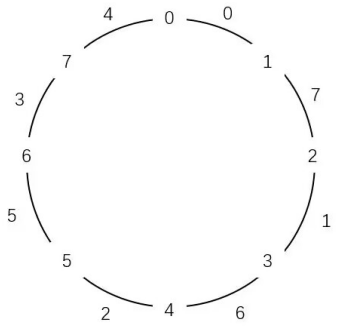


图 5

对于 C_1，我们仍然从 0 出发，作一样的构造。不过原来填 x 需要改为填写 $2n-1-x$。由于 G 是二分图，原来 $w(v) = n$ 的增量为 $2(2n-1)-n$，原来 $w(v) = n-1$ 的增量为 $2(2n-1)-(n-1)$（否则出现奇环），同时 $w(0) = n/2 + 2(2n-1) - n/2$。从而，$\forall v \in V, w(v) = 2(2n-1)$。 □

Lemma 6.2.3: Q_4 存在一种基于边的 Magic Labeling。

证明: 下图 6 展示了一种将 Q_4 拆分成两个哈密尔顿回路的方案。

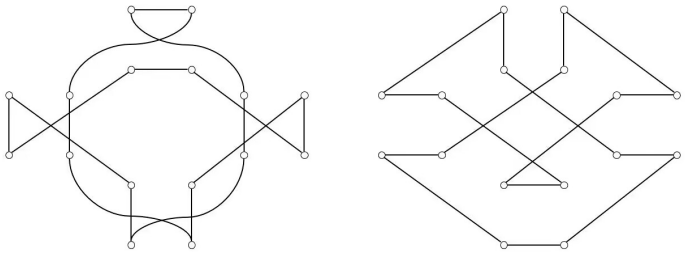


图 6

然后再用 **Lemma 6.2.2** 即得一种填法。 □

Lemma 6.2.4: 若 $Q_n(2|n)$ 存在一种基于边的 Magic Labeling，则 Q_{n+2} 也存在。

证明: 依据点的编号 $\bmod 4$，可把 Q_{n+2} 拆分成 4 个 Q_n，记作 G_0, G_1, G_2, G_3。

在每个 G_i 内部，考虑沿用 Q_n 的填法，并对每条方向为 j 的边加上 $d_{i,j} \cdot T (0 \le i < 4, 0 \le j < n)$。其中 $d_{i,j}$ 和 T 均为待定的常数。

而对于四个 G_i 之间的每个 C_4，考虑如下图 7 的填法，其中 x, y 对于不同的 C_4 来说是不同的，但我们希望 $x + y$ 相同。

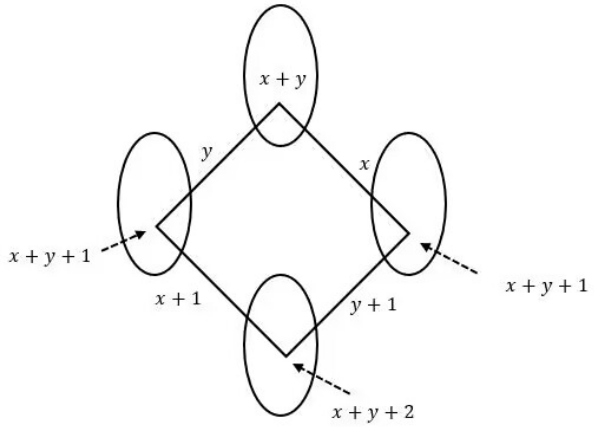


图 7

现在，为了使方案合法，我们构造的矩阵 d 需要满足如下条件：

1. $\forall 0 \le j < n$，$d_{i,j}$ 取遍 $\{0, 1, 2, 3\}$。
2. 记 $s_i = \sum_{j=0}^{n-1} d_{i,j}$，则 $s_0 - 1 = s_1 = s_2 = s_3 + 1$。

当 $n = 4$ 时，不难通过枚举或搜索等方法得到可行解：

	0	1	2	3
G_0	3	0	2	2
G_1	2	3	0	1
G_2	1	2	3	0
G_3	0	1	1	3

对于一般的偶数 n，只需在上面的矩阵右边交替添加 $[0, 1, 2, 3]^T$ 和 $[3, 2, 1, 0]^T$。

最后，我们需要确定 T 和所有的 x, y，使得所有边的标号互不相同。注意到四个 G_i 内部边的标号的并，恰好覆盖 $[i \cdot T, i \cdot T + n \cdot 2^{n-1}), \forall 0 \le i < 4$。

因此，考虑取 $4T = |E|_{Q_{n+2}}$ 即 $T = (n + 2)2^{n-1}$，可使未使用的标号构成区间 $[i \cdot T + n \cdot 2^{n-1}, (i + 1) \cdot T), \forall 0 \le i < 4$。容易首尾配对完成标号，即 x 与 $y + 1$、$x + 1$ 与 y 配对，所有 $x + y$ 均相等。 □

综上，我们有：

Theorem 6.2.1: 对于 $n > 2$，Q_n 存在一种基于边的 Magic Labeling 当且仅当 n 为偶数。

7 总结

本文从超立方体的基本定义出发，阐释了其作为无向图和偏序集时所具备的优美性质，并分析了若干道与之密切相关的信息学竞赛例题。

本文的最后一节详细讨论了超立方体上 Magic Labeling 的构造问题，将数学与算法相结合，展现了超立方体多种结构视图的巧妙应用。

笔者希望通过本文，让更多选手了解到超立方体这种独特的数学模型，并在未来进行更为深入的研究学习。

8 致谢

感谢中国计算机学会提供交流和学习的平台。

感谢父母和家人对我多年以来的关心和支持。

感谢华师大二附中金靖老师对我的指导和帮助。

感谢王又嘉同学和杨宁远同学为本文审稿。

9 参考文献

[1] Frank Harary, A Survey of the Theory of Hypercube Graphs (1988)

[2] Oliver Bernardi,On the Spanning Trees of the Hypercube and Other products of Graphs,https://arxiv.org/abs/1207.(2018)

[3] D. LUBELL,A Short Proof of Sperner’s Lemma

[4] Youcef Saad and Martin H. Schultz, Topological Properties of Hypercubes,Research Report YALEU/DCS/RR- 389 (1985)

[5] W.D.Wallis,Edy T.Baskoro,Mirka Miller and Slamin Edge-magic total labelings (2000)

[6] Petr Gregor and Petr Kovar,Distance magic labelings of hypercubes (2013)

[7] Curtis Greene, The Mobius Function of a Partially Ordered Set (1982)

[8] R.P.Dilworth, A Decomposition Theorem for Partially Ordered Sets,Annals of Mathematics, Second Series, Vol. 51, No. 1 (Jan., 1950), pp. 161-166

[9] 彭博，2021 年集训队互测，《子集匹配》解题报告

浅谈几种分解质因数方法

重庆市巴蜀中学 罗思远

摘 要

数论是信息学竞赛中一大板块，其中有不少题目都涉及分解质因数，分解质因数也是学术界具有重要地位的问题之一。本文介绍了三种在 OI 范围内认知度尚且较低，算法本身也比较简单的质因数分解算法：SQUFOF，CFRAC 和二次筛法。

1 约定

本文中，涉及到分解质因数时，设 N 为要分解的数。定义对 N 分解质因数的过程为将 N 写为素数之积的形式，也即求出 $p_1, p_2, \ldots, p_k$ 使得 $\forall 1 \le i \le k$，p_i 均为素数，且 $\prod_{1 \le i \le k} p_i = n$。

称正整数 d 是正整数 n 的非平凡因子当且仅当 $d|n$ 且 $1 < d < n$。

2 前置知识

2.1 Miller Rabin 素数判断

先简要介绍 Miller Rabin 素数判断。设待判断的数为 n（不妨假设 n 为奇数），设 $n-1 = 2^d \cdot r$，这里 r 为奇数，d 为正整数。选取 a，用快速幂计算 $a^r \bmod n$ 的值 x，接着对 x 执行至多 d 次自乘。最终，若 n 为素数，x 的值一定为 1，这是因为由费马小定理，当 n 为素数时 $a^{n-1} \equiv 1 \pmod{n}$ 总成立。当 x 变为 1 时算法终止。

如果在某次自乘后 x 变为 1，自乘前 x 的值在模 n 意义下非 1 也非 $n-1$，则 n 一定不是素数。这是因为，若 $2 \le x \le n-2$ 满足 $x^2 \equiv 1 \pmod{n}$，就有 $n|(x^2-1) = (x+1)(x-1)$，若 n 为素数，就有 $n|x-1$ 或 $n|x+1$ 而 $x+1, x-1$ 均在 $[1, n-1]$ 内，不是 n 的倍数，矛盾。

若对于某个底数 a 与合数 n，上面的过程不能确认 n 为合数，我们称 a 是 n 的一个 strong liar。[3] 指出，对于合数 n，strong liar 的数量不超过 $\frac{n}{4}$，故如果重复执行 k 轮上面的算法，每轮算法均随机选取 a，则错误率不超过 4^{-k}。[3] 还指出，对于 $3,825,123,056,546,413,051 > 10^{18}$ 以内的正整数 n，选取前 9 个素数作为 a 即可保证结果正确；对于 2^{64} 以内的正整数 n，选取前 12 个素数作为 a 即可保证结果正确。

Miller Rabin 素数判断共需执行 $O(k \log n)$ 次模 n 意义下的乘法，其中 k 是轮数。

2.2 试除法

设 n 为要分解的数。若 n 不是素数，则 n 的最小质因子一定小于等于 $\sqrt{n}$。先用 $O(\sqrt{n})$ 的线性筛求出所有 $\sqrt{n}$ 以内的所有素数，分解时只需枚举 $\sqrt{n}$ 以内的所有素数即可。根据素数定理 [1]，小于等于 x 的素数有 $O(\frac{x}{\log x})$ 个，因此时间复杂度为预处理 $O(\sqrt{n})$，单次分解 $O(\frac{\sqrt{n}}{\log n})$。

例 2.2.1. 给出 n 个正整数 $a_1, \ldots, a_n$，判断其中是否存在两个数 a_i, a_j $(i \neq j)$ 使得 a_i, a_j 不互素。$1 \leq n \leq 10^5$，$1 \leq a_i \leq 10^9$，时间限制 *3* 秒。[1]

解法　直接用试除法对所有 a_i 分解质因数，判断是否有质因子出现在两个以上 a_i 中即可。时间复杂度 $O(n\frac{\sqrt{m}}{\log m})$，其中 $m = \max a_i$。特别地，由于需要试除的质数个数很少，可以在适当预处理后使用 Barrete 模乘 [2] 优化除法运算的常数。 ■

需要注意的是，某些题目中，根据题目的特殊性质，可能可以调整试除上界来取得更优秀的复杂度。

例 2.2.2. T 次询问：给出 x 求 $\mu(x)$。$T \leq 5000, 1 \leq x \leq 10^{18}$。时间限制 *2* 秒。[2]

解法　直接用试除法对所有 a_i 分解质因数，但试除的质因子只枚举 $\sqrt[3]{x}$ 以内的。若除完后还有剩余，只可能是质数，质数的平方，或两个质数相乘。前两种情况分别用 Miller Rabin 素数判断与直接计算平方根判出，若都不是即可确定是两个质数相乘。容易发现，只凭以上信息，就能确定 $\mu(x)$ 的值。时间复杂度 $O(T\frac{\sqrt[3]{m}}{\log m})$，其中 $m = \max x$。 ■

2.3 连分数的简单介绍

在介绍下面的质因数分解算法之前，先要了解连分数的概念以及几个性质。

定义 对任意实数 x，都存在唯一的有限或无限的数列 $[a_0; a_1, a_2, \ldots]$ 使得

- $a_0 \in \mathbb{Z}$；$\forall i > 0$，$a_i \in \mathbb{N}^*$；
- $x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}$。

称 $[a_0; a_1, a_2, \ldots]$ 为 x 的一般连分数表示。以下，如果没有特殊说明，所说的“连分数”均为“一般连分数”之意。

求出连分数表示 假设要求出实数 x 的连分数表示 $[a_0; a_1, a_2, \ldots]$，可以这样计算：

[1] https://codeforces.com/contest/1771/problem/C

[2] 修改自 https://www.luogu.com.cn/problem/T129723

- 令 $x_0 = x$。
- 令 $a_0 = \lfloor x_0 \rfloor$，$x_1 = \dfrac{1}{x_0 - a_0}$。
- 令 $a_1 = \lfloor x_1 \rfloor$，$x_2 = \dfrac{1}{x_1 - a_1}$。
- 令 $a_2 = \lfloor x_2 \rfloor$，$x_3 = \dfrac{1}{x_2 - a_2}$。
- 以此类推。若中途某时刻 $a_i = x_i$（也就是 x_i 为整数），则算法终止。

性质 2.3.1. 算法会终止当且仅当 x 是有理数。

证明. 如果算法终止了，x 一定是有理数，因为此时连分数表示只有有限项，可以直接计算出 x 的值。

如果 x 是有理数，不妨假设 $x = \dfrac{P}{Q}$，其中 $Q > P$，$\gcd(P, Q) = 1$。则不难发现，从 x_0 到 x_1 的过程，就是令 $Q \to P'$，$P \bmod Q \to Q'$，$\dfrac{P'}{Q'} \to x_1$。$(P, Q) \to (P', Q')$ 的过程与计算最大公约数的欧几里得算法完全一致。由于欧几里得算法只会进行有限步，连分数表示自然也只有有限项，故算法会终止。 □

$\sqrt{n}$ 连分数表示的性质 考察 $\sqrt{n}$ 的连分数表示（不妨假设 n 不是完全平方数）$[a_0; a_1, a_2, \dots]$。

将 x_i 写成 $\dfrac{\sqrt{n} + P_i}{Q_i}$ 的形式（$P_0 = 0, Q_0 = 1, Q_i \neq 0$）。由此可得，$a_i = \lfloor \dfrac{\sqrt{n} + P_i}{Q_i} \rfloor$，$x_{i+1} = \dfrac{1}{x_i - a_i} = \dfrac{1}{\frac{\sqrt{n}+P_i}{Q_i} - a_i} = \dfrac{1}{\frac{\sqrt{n}+P_i - a_i Q_i}{Q_i}} = \dfrac{\sqrt{n} + (a_i Q_i - P_i)}{\frac{n-(P_i - a_i Q_i)^2}{Q_i}} = \dfrac{\sqrt{n} + (a_i Q_i - P_i)}{\frac{n - P_{i+1}^2}{Q_i}}$。

因此我们得到递推式：

$$P_{i+1} = a_i Q_i - P_i, Q_{i+1} = \frac{n - P_{i+1}^2}{Q_i} \ (i \geq 0) \tag{1}$$

性质 2.3.2. P_i, Q_i 总是整数。

证明. 用数学归纳法。当 $i = 0$ 时成立。当 $i = 1$ 时，$x_1 = \dfrac{1}{\sqrt{n} - a_0} = \dfrac{\sqrt{n} + a_0}{n - a_0^2}$，由此 $P_1 = a_0, Q_1 = n - a_0^2$，成立。

假设 $i = k-1, k$ 时 P_i, Q_i 均为整数，显然 P_{k+1} 也是整数。而 $n - P_{k+1}^2 = n - (a_k Q_k - P_k)^2 \equiv n - P_k^2 \pmod{Q_k}$。而由递推式，$n - P_k^2 = Q_k Q_{k-1}$，故 $n - P_k^2 \equiv 0 \pmod{Q_k}$，故 $n - P_{k+1}^2 \equiv 0 \pmod{Q_k}$，所以 Q_{k+1} 是整数。故 P_{k+1}, Q_{k+1} 都是整数。

综上，P_i, Q_i 总是整数。 □

性质 2.3.3. 对所有整数 $i \geq 0$ 都有 $0 < Q_i < 2\sqrt{n}, 0 \leq P_i < \sqrt{n}$。

证明. 由连分数的计算过程容易发现，当 $i \geq 1$ 时，总有 $x_i > 0$，$a_i \geq 1$。

下面用数学归纳法证明 $\forall i \geq 0$，$Q_i > 0, 0 \leq P_i < \sqrt{n}$。当 $i = 0$ 时成立。

假设 $i = k$ 时成立，分类讨论：

- $Q_k > \sqrt{n}$，注意 $a_k \geq 1$，因此 $P_{k+1} \geq Q_k - P_k > \sqrt{n} - P_k > 0$。
- $Q_k < \sqrt{n}$，则 $a_k = \lfloor \dfrac{\sqrt{n} + P_k}{Q_k} \rfloor \geq \lfloor \dfrac{Q_k + P_k}{Q_k} \rfloor = 1 + \lfloor \dfrac{P_k}{Q_k} \rfloor$。所以 $a_k Q_k > P_k$，因此 $P_{k+1} > 0$。

由于 $x_{k+1} > 0$，$P_{k+1} > 0$，所以 $Q_{k+1} > 0$。这意味着 $n - P_{k+1}^2 > 0$，也即 $P_{k+1} < \sqrt{n}$。

综上，总有 $Q_i > 0, 0 \leq P_i < \sqrt{n}$。

再结合 $P_{i+1} = a_i Q_i - P_i$，也即 $Q_i = \dfrac{P_i + P_{i+1}}{a_i} \leq P_i + P_{i+1} < 2\sqrt{n}$，就有 $0 < Q_i < 2\sqrt{n}$。 □

如果无理数 x 的连分数表示 $[a_0; a_1, a_2, \dots]$ 满足，存在正整数 len, s，使得对于任意正整数 $i \geq s$，都有 $a_i = a_{i-len}$，则称 x 的连分数表示有循环节。上述结论也说明，当 x 可以写成 $\sqrt{n}$ 的形式（其中 n 是非完全平方数的正整数）时，x 的连分数表示一定有循环节。实际上可以证明，无理数 x 的连分数表示存在循环节，当且仅当存在一元二次方程 $ax^2 + bx + c = 0$ 使得其存在一个实根为 x。证明与本文无关，故不展开。[4]

下面当提到 $\sqrt{n}$ 的连分数表示时，我们会沿用 P_i, Q_i 的记号。

连分数收敛子及其性质 设大于 1 的无理数 x 的的连分数表示为 $[a_0; a_1, a_2, \dots]$。设 $\{A_k\}, \{B_k\}$ 为两个正整数数列，满足 $[a_0; a_1, a_2, \dots, a_k] = \dfrac{A_k}{B_k}$。称 $\dfrac{A_k}{B_k}$ 为该连分数的第 k 个收敛子。

性质 2.3.4. 对于 $i \geq 2$，

$$A_i = a_i A_{i-1} + A_{i-2}, B_i = a_i B_{i-1} + B_{i-2} \tag{2}$$

证明. 用数学归纳法。对 $i = 2$ 容易直接验证成立。假设 $i = k$ 成立，考虑 $i = k + 1$ 时。

注意到，如果把在连分数表示的第 $0 \sim k$ 项里，令 $a_k + \dfrac{1}{a_{k+1}} \to a_k$，则最终算出来的值就会是 $\dfrac{A_{k+1}}{B_{k+1}}$。所以，

$$\frac{A_k}{B_k} = \frac{A_{k-1}a_k + A_{k-2}}{B_{k-1}a_k + B_{k-2}} \to \frac{A_{k+1}}{B_{k+1}} = \frac{A_{k-1}(a_k + \frac{1}{a_{k+1}}) + A_{k-2}}{B_{k-1}(a_k + \frac{1}{a_{k+1}}) + B_{k-2}} = \frac{A_k a_{k+1} + A_{k-1}}{B_k a_{k+1} + B_{k-1}}$$

因此上述递推式对所有 i 都成立。 □

上述过程求出的 A_k, B_k 是否可能不互质？事实上有下述性质：

性质 2.3.5. 对任意 $i \geq 1$，都有 $A_{i-1}B_i - A_i B_{i-1} = (-1)^i$。

该性质的证明只需使用数学归纳法后直接代入性质 4 即得，这里就不赘述了。结合 Bézout's identity [5] 即得 $\gcd(A_i, B_i) = 1$ 对 $i \geq 0$ 总成立。

性质 2.3.6. 对任意 $i \geq 2$，$x = \dfrac{A_{i-2} + A_{i-1}x_i}{B_{i-2} + B_{i-1}x_i}$ 总成立。

证明. 注意到，如果把在连分数表示的第 $0 \sim k$ 项里，令 $x_k \to a_k$，最终的值就会是 x。再结合性质 4，即证毕。 □

3 Square Form Factorization (SQUFOF)

Square Form Factorization，简称 SQUFOF，由 Shanks 提出。[13]

3.1 SQUFOF 算法梗概

有了上述准备，我们可以开始介绍 SQUFOF 的算法流程了。它基于下述性质：

性质 3.1.1. 对任意 $i \geq 1$，都有 $A_{i-1}^2 - nB_{i-1}^2 = (-1)^i Q_i$。

证明. 由性质 6：$\sqrt{n} = \dfrac{A_{i-2} + A_{i-1}x_i}{B_{i-2} + B_{i-1}x_i}$。代入 $x_i = \dfrac{\sqrt{n} + P_i}{Q_i}$ 得

$$\sqrt{n} = \frac{Q_iA_{i-2} + P_iA_{i-1} + \sqrt{n}A_{i-1}}{Q_iB_{i-2} + P_iB_{i-1} + \sqrt{n}B_{i-1}} \to \sqrt{n}(Q_iB_{i-2} + P_iB_{i-1} - A_{i-1}) = Q_iA_{i-2} + P_iA_{i-1} - nB_{i-1}$$

由于 $\sqrt{n}$ 是无理数，P, Q, A, B 均为整数数列，所以 $Q_iB_{i-2} + P_iB_{i-1} - A_{i-1} = 0, Q_iA_{i-2} + P_iA_{i-1} - nB_{i-1} = 0$。

前式乘以 A_{i-1}，后式乘以 B_{i-1}，相减得 $Q_i(B_{i-2}A_{i-1} - A_{i-2}B_{i-1}) = A_{i-1}^2 - nB_{i-1}^2$。代入性质 5 即得到待证式。 □

因此，$A_{i-1}^2 \equiv (-1)^i Q_i \pmod{n}$。而 SQUFOF 算法的思想即：找到某个偶数 i，使得 Q_i 为完全平方数 t^2，这样就有 $A_{i-1}^2 - t^2 \equiv 0 \pmod{n}$，则很大概率 $\gcd(n, A_{i-1} \pm t)$ 是 n 的一个非平凡因子。

据此可以得出下面的算法流程：

算法流程（一）

- 初始化 $P_0, Q_0, A_0, a_0, P_1, Q_1, A_1, a_1$ 的值。
- 根据递推式 (1) (2) 递推求出 P, Q, A, a 的每一项，直到在某个偶数位置 i，Q_i 是完全平方数。
- 返回 $\gcd(n, A_{i-1} \pm \sqrt{Q_i})$。

有概率上述过程返回的仍是平凡因子，这时可以选取一个较小的正整数 k 作为 multiplier，对 $n' = kn$ 重复上述算法。

3.2 优化

常数优化

- n 不太大时，可以用 Barrete 模乘或类似方式优化 A_i 的计算。注意：无论事先选取的 multiplier 是什么，A_i 都只对 n 取模即可（而不是 kn）。
- 判断 x 是否是平方数：选取一个值 M，先判断 x 模 M 的余数是不是二次剩余，若是再调用平方根函数。
- 过程中，P, Q, a 的计算共用到了两次除法（Q 一次，a 一次），但实际上可以省略 Q 这一次：$Q_iQ_{i-1} = n - P_i^2, Q_iQ_{i+1} = n - P_{i+1}^2$，两式相减得 $Q_i(Q_{i+1} - Q_{i-1}) = P_i^2 - P_{i+1}^2 = (P_i + P_{i+1})(P_i - P_{i+1}) = a_iQ_i(P_i - P_{i+1})$。也即，

$$Q_{i+1} = Q_{i-1} + a_i(P_i - P_{i+1}) \tag{3}$$

  这样，计算 Q_{i+1} 就不需要除法了。
- 所有变量都可以滚动计算，不需要开数组。

避免计算 A_k 的优化

受到计算机字长的限制，当 n^2 超过计算机能一次性处理的范围但 n 未超过时，P, Q, a 的计算仍然可以照常进行，但 A 就不便于计算了。事实上，存在能够避免直接计算 A_k 的方法。下面我们先给出算法流程再证明正确性。

算法流程（二）

- 初始化 $P_0, Q_0, a_0, P_1, Q_1, a_1$ 的值。
- 根据递推式 (1)(3) 递推求出 P, Q, a 的每一项，直到在某个偶数位置 k，Q_k 是完全平方数 w^2。
- 从 $\frac{\sqrt{n} - P_k}{w}$ 开始，按照与上面相同的方法计算 P', Q', a'。具体地，令 $P'_0 = -P_k$，$Q'_0 = w$，并用与递推式 (1)(3) 一样的方法递推 P', Q'，直到某处 $P'_i = P'_{i+1}$。
- 返回 $\gcd(Q'_i, n)$。

为说明算法的正确性，我们先证明以下性质：

性质 3.2.1. 若某个 $i \geq 0$ 满足 $P'_i = P'_{i+1}$，就有 $n = Q'_i(Q'_{i+1} + \frac{{a'_i}^2 Q'_i}{4})$。

证明. 由算法流程，a', P', Q' 也服从递推式 (1)(3)。由 $P'_i + P'_{i+1} = a'_iQ'_i$，得 $P'_{i+1} = \frac{a'_iQ'_i}{2}$。再代入 $n = {P'_{i+1}}^2 + Q'_iQ'_{i+1}$，化简即得上式。 □

性质 8 告诉我们，最后一步中返回的 $\gcd(Q'_i, n)$ 的确很有可能就是 n 的因数。之所以不直接返回 Q'_i，是因为 $\frac{a_i'^2 Q'_i}{4}$ 可能不是整数。

性质 3.2.2. 递推 P', Q' 的过程中，对任意 $i \geq 1$ 都有 $0 < Q'_i < 2\sqrt{n}, 0 \leq P'_i < \sqrt{n}$。

证明. 由算法流程，$a'_0 = \lfloor \frac{\sqrt{n} - P_k}{w} \rfloor$。$P'_1 = a'_0 \cdot w + P_k \leq \sqrt{n} - P_k + P_k = \sqrt{n}$，又因为 $\sqrt{n}$ 不是整数，所以 $P'_1 < \sqrt{n}$，而显然 $P'_1 \geq 0$。由 $Q'_1 = \frac{n - P_1'^2}{Q'_0}$ 也可得到 $Q'_1 > 0$。这样，套用性质 3 证明中的归纳，即可证明结论。 □

这样我们知道第二次递推过程中 P', Q' 的大小仍为 $\sqrt{n}$ 级别，因此两次递推在实现上完全没有区别。

通过性质 8, 9，我们已经能看出算法的正确性，接下来考虑第二次递推的时间复杂度。

性质 3.2.3. 第二次递推的下标数量约为第一次递推下标数量的一半。

该性质的证明（参见 [13]）涉及到二元二次型 (Binary Quadratic Forms) 的性质，由于作者学识浅薄，这里略去。由性质 10，分析时间复杂度时就可以只分析第一次递推的复杂度。

[13] 中还指出了可以维护一个列表以在不计算 A 的前提下判断某个完全平方数 Q 是否仅会带来平凡因子，这里不具体介绍。

3.3 复杂度分析与运行效果

注意到 $Q_i < 2\sqrt{n}$，而 $2\sqrt{n}$ 以内的完全平方数约占 $\frac{1}{O(\sqrt[4]{n})}$，因此生成 $O(\sqrt[4]{n})$ 项后很可能就能找到一个完全平方数 [6]（注意这依赖于 Q_i 几乎是一列随机整数的假设）。实践中，SQUFOF 约能在 1 秒内分解 10^{30} 以内的随机整数（随机方式为随机两个 10^{15} 左右的质数并相乘得到 n），可参考笔者在此处[3] 的提交。

4 Continued Fraction Method (CFRAC)

CFRAC 是另一种基于 $\sqrt{n}$ 的连分数表示的质因数分解算法，由 Morrison 和 Brillhart 提出。[14]

4.1 算法思想

与 SQUFOF 一样，它也基于性质七；然而，SQUFOF 主要思想是“等待”一个完全平方 Q_i 的出现，而 CFRAC 尝试从已有的 Q_i 构造出一个完全平方数。具体地，若下标集合 $U =$

[3] https://loj.ac/p/6466

$\{i_1, i_2, \ldots, i_k\}$ 满足 $S = \prod_{i\in U}(-1)^i Q_i$ 是完全平方数，则取 $x = \sqrt{S} \bmod n$，$y = \prod_{i\in U} A_{i-1} \bmod n$，就有 $x^2 - y^2 \equiv 0 \pmod n$。这样，$\gcd(n, x \pm y)$ 很可能就是一个非平凡因子。

如何找到一个 Q_i 的子集乘积是完全平方数？如果可以分解 Q_i 的质因数，问题就比较清楚了：将分解质因数的结果看成一个无限维的向量，向量的第 j 位是在分解结果中，第 j 个质数的幂次模二的结果，只需要找到一个在模二意义下和为零向量的向量子集，就找到了一个合法子集。

称一个数是 B-smooth 的，当且仅当其所有质因数都不超过 B。CFRAC 的思想即，设置阈值 B，只用 $\le B$ 的质数去试除 Q_i，只考虑能够在这个过程中就完全分解的 Q_i（也即只考虑 B-smooth 的 Q_i），并用高斯消元求解和为零向量的向量子集。注意 Q_i 有时会带有额外的 -1 系数，可以将 -1 也视为一个质数作为高斯消元的一部分。一个重要的优化是，由性质 7，$A_{i-1}^2 - nB_{i-1}^2 = (-1)^i Q_i$，若 Q_i 有质因子 p，就有 $A_{i-1}^2 - nB_{i-1}^2 \equiv 0 \pmod p$。因为 $\gcd(A_{i-1}, B_{i-1}) = 1$，所以不可能 $B_{i-1} \equiv 0 \pmod p$，因此可以在等式两边同除 B_{i-1}^2 即得 $(\frac{A_{i-1}}{B_{i-1}})^2 \equiv n \pmod p$，这说明 n 是模 p 意义下的二次剩余。因此可以先用快速幂计算勒让德符号 (n/p) 判断 n 是否是模 p 意义下的二次剩余，提前排除约一半无用质数。

CFRAC 还存在一个效果较好优化：设执行完毕试除后，剩余的数为 p。若存在两个数剩余的数相同，均为 p，不妨设已知的两个同余式为 $a_1^2 \equiv p \cdot q_1 \pmod n$，$a_2^2 \equiv p \cdot q_2 \pmod n$（$q_1, q_2$ 都是已经确定质因数分解的 B-smooth 数），结合两式可得 $(\frac{a_1 a_2}{p})^2 \equiv q_1 q_2 \pmod n$，这样我们又获得了一条可以直接用于高斯消元的等式。这样可以不小地提高 Q_i 的利用率，进而优化效率。由于存储剩余数 p 对应的 Q 及质因数分解也需要不小的时间，所以可以预先设置阈值 $B' \approx B^2$，仅当 $p < B'$ 时才执行上述优化。

4.2 复杂度分析与运行效果

实践上，由于同一个数质因子个数不多，所以高斯消元的矩阵非常稀疏，而且可以压位，几乎不会成为耗时瓶颈，程序大部分时间都消耗在试除上 [12]。这里我们分析未加入最后一个优化的 CFRAC 的试除部分复杂度（注意加入最后一个优化并不等价于直接将 B 设为 B'，所以不能直接套用下述分析！）。

这里仍然需要假设 $\sqrt{n}$ 的连分数表示中求出的 Q_i 是在 $(0, 2\sqrt{n})$ 之间的一列随机整数，设 B-smooth 的整数在 $(0, 2\sqrt{n})$ 中的占比为 $\frac{1}{p(B)}$，则粗略估计算法越需要执行 $p(B)\cdot\pi(B)^2$ 次试除以获得一个线性相关向量子集（期望 $p(B)$ 次找到一个 B-smooth 的整数；需要大约 $\pi(B)$ 个这样的整数才能找到一个线性相关子集；每个整数执行 $\pi(B)$ 次试除）。我们即需要最小化 $p(B)\cdot\pi(B)^2$。

在上述假设下，我们粗略估计 CFRAC 的复杂度：首先不妨忽略乘积中的 log 因子，这样，复杂度约为 $p(B)\cdot B^2$。根据 [7]，x 以内 $x^{1/\alpha}$-smooth 数的占比的一个粗糙估计为 $\alpha^{-\alpha}$。因此可以估计复杂度约为 $O((\frac{\log n}{2\log B})^{\frac{\log n}{2\log B}}\cdot B^2)$。取对数得到 $2\log B + \frac{\log n}{2\log B}(\log\log n - \log(2\log B))$。

可以发现整个式子形如 $A(n)\frac{\log n}{2\log B}+B(n)\cdot 2\log B$，$A(n),B(n)$ 的级别都远小于 $\log n$，因此粗略估计得 $2\log B$ 大体约为 $\sqrt{\log n}$，所以 $\log\log n-\log(2\log B)\approx\frac{1}{2}\log\log n$，所以原式约等于 $2\log B+\frac{\log n}{4\log B}\log\log n$，最小值为 $\sqrt{2\log n\log\log n}$。因此时间复杂度可以估计为 $O(\exp((1+o(1))\sqrt{2\log n\log\log n}))$，在 B 为 $\exp(\sqrt{\log n\log\log n/(2\sqrt{2})})$ 级别时取到。因此，CFRAC 是一种亚指数的质因数分解算法。

该算法实现非常简洁，运行速度也较快，实现可参考笔者在此处[4] 的提交。CFRAC 可以在几秒内分解 `__int128` 范围内的整数。

5 The Quadratic Sieve (QS)

二次筛法（The Quadratic Sieve，简称 QS）是一种质因数分解算法，由 Carl Pomerance 发明。[12]

5.1 算法思想

事实上，如果仅仅只需要快速生成很多 $x^2\equiv y\pmod n$ 形式的同余式，最简便的方法就是从 $\lfloor\sqrt{n}\rfloor+1$ 开始从小到大枚举 x，令 $y=x^2\bmod n$。而且，如果用该过程替换 CFRAC 中连分数计算过程，可以发现指数的级别是相同的！这被称为 Kraitchik's method[12]。然而，这种算法在效率上比 CFRAC 差了不少，因为需要判断是否 B-smooth 的数的大小大约是 $C\times\sqrt{n}$ 级别（其中 C 是需要判断的数的个数，约为 $B\cdot p(B)$），比 CFRAC 的 $2\sqrt{n}$ 大了不少倍，这会导致 smooth 的数相对稀疏很多。

然而，上述方法还有优化的余地。注意有不少时间都浪费在了对无用的数执行试除法上，如果能直接算出哪些数是 B-smooth 的，时间复杂度中，立马可以少乘以一个 $\pi(B)$。而 Kraitchik's method 生成的 x^2-n 形式的数具有特别的形式，这为我们快速判断哪些数是 B-smooth 的提供了便利。

注意到，$p|x^2-n$ 等价于 $x^2\equiv n\pmod p$，用计算二次剩余的算法（如 Cipolla 算法）可以快速求出 x 在模 p 意义下的 $O(1)$ 个可能值，我们如果只用 p 去分解模 p 意义下合法的 y，根据埃氏筛 [8] 的复杂度分析，该过程的时间复杂度为 $O(C\log\log C)$。

直接用上述筛法分解 x^2-n，我们就得到了一种分解质因数部分时间复杂度为 $O(C\log\log C)$ 的算法。不过，直接这样实现的实际运行效果并不好，因为待分解的数的大小随着它的下标是线性增长的，当待分解的数越来越多、越来越大时，B-smooth 的比例会越来越少。所以，实践中通常会同时取多个二次函数 $f(x)$，每个二次函数仅计算相对少量的 $f(x)$ 值，以

[4] http://119.27.163.117/problem/226

此控制 $f(x)$ 的范围 [10]。具体地，取 $f(x) = (Ax+B)^2 - n$，且整数 C 满足 $B^2 - n = AC$，就有 $f(x) = A(Ax^2 + 2Bx + C)$。若 A 是完全平方数 p^2，就有 $(Ax^2 + 2Bx + C) \equiv (\frac{Ax+B}{p})^2 \pmod{n}$。

这样，我们得到下列新的算法流程：多次选取 $A = p^2$。取 B 使得 $b^2 \equiv n \pmod{A}$。取 $C = \frac{B^2 - n}{A}$。这样，对于 x 而言，需要被筛的数值即 $Ax^2 + 2Bx + C$。通过适当选取 A, B, C，可以使得 $Ax^2 + 2Bx + C$ 的绝对值（若为负数，将 -1 视为一个质数即可）较小，从而增加 smooth 数的占比。

与 CFRAC 一样地，若存在两个数在分解后剩余的数相同，同样可以将其结合得到新的可以用于高斯消元的关系。

5.2 复杂度分析与运行效果

使用与 CFRAC 相同的复杂度分析方法，容易得到分解质因数部分最终的时间复杂度为 $O(\exp((1+o(1))\sqrt{\log n \log\log n}))$，在阈值 B 取 $\exp(\sqrt{\log n \log\log n}/2)$ 级别时取到。在 B 较大时，可以将高斯消元换为针对稀疏矩阵的其它算法。

已有的一些 QS 实现，如 [9] 是一份 C 语言的实现，可以在几秒内分解 2^{200} 左右的整数。

6 总结

本文简单介绍了几种在 OI 界认知度较低的质因数分解算法，也展现了目前对分解质因数问题的一些思路：通过分析连分数表示的性质、对 smooth numbers 分解质因数后用消元寻找同余式、将数论中的“筛法”一般化等方式，得到了几种分解算法。希望本文中的介绍能起到抛砖引玉的作用，为 OI 中的数论问题带来一些启发。

致谢

感谢中国计算机学会提供学习交流的平台。
感谢父母的养育之恩。
感谢黄新军老师的关心和指导。
感谢郭雨豪同学为本文验稿。

参考文献

[1] Wikipedia, Prime Number Theorem, https://en.wikipedia.org/wiki/Prime_number_theorem

[2] Wikipedia, Barrett Reduction, https://en.wikipedia.org/wiki/Barrett_reduction

[3] Wikipedia, Miller-Rabin primality test, https://en.wikipedia.org/wiki/Miller-Rabin_primality_test

[4] OI Wiki, 连分数, https://oi-wiki.org/math/number-theory/continued-fraction/

[5] Wikipedia, Bézout's identity, https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity

[6] Wikipedia, Shanks's square forms factorization, https://en.wikipedia.org/wiki/Shanks%27s_square_forms_factorization

[7] Wikipedia, Dickman function, https://en.wikipedia.org/wiki/Dickman_function

[8] OI Wiki, 筛法, https://oi-wiki.org/math/number-theory/sieve/

[9] michel-leonard, C Factorization using Quadratic Sieve, https://github.com/michel-leonard/C-Quadratic-Sieve

[10] 钟子谦, 二次筛法 (Quadratic Sieve), https://zhuanlan.zhihu.com/p/106650020

[11] Hans Riesel. Prime Numbers and Computer Methods for Factorization. Springer Science + Business Media, LLC, 1994.

[12] Carl Pomerance. A Tale of Two Sieves. Notices of the AMS, Volume 43, Number 12, 1473-1485, 1996.

[13] Jason E. Gower; Samuel S. Wagstaff, Jr. Square Form Factorization. Mathematics of Computation, Volume 77, Number 261, 551-588, January 2008.

[14] Michael A. Morrison; John Brillhart. A Method of Factoring and the Factorization of F_7. Mathematics of Computation, Volume 29, Number 129, 183-205, January 1975.

一类基础子串数据结构

中国人民大学附属中学 许庭强

摘 要

本文介绍了一种新的字符串数据结构：基本子串结构。该结构功能强大，可同时处理前后缀字符串信息。此外，本文介绍了其在信息学竞赛中的应用。

1 前言

字符串理论已在 OI 发展很久，其中逐渐产生了 SA、SAM 等高级后缀数据结构。这些结构深入剖析了字符串的性质，可以用来高效地解决大量的字符串问题。而这些后缀数据结构大多为单向联系，仅能处理后缀关系，而无法同时处理与前缀关系相关的内容。由此，笔者借助 SDOI2022 子串统计这道题目，对一类能够同时处理前后缀的字符串结构展开了研究，并写作了本文。该结构在此前从未出现过，所以笔者将其命名为基本子串结构。另外，笔者在其基础上研究了一类特殊的树链剖分结构。

本文第二节中会规定一些约定和记号。第三节中将介绍基本子串结构的内容，构造及其应用。第四节则会对基本子串结构上一类特殊的树链剖分进行分析，并给出了这一类树链剖分的应用场景。

2 约定与记号

对长度为 n 的字符串 $s = s_1 s_2 \dots s_n$，我们用 $s[l, r]$ 表示其在第 l 到第 r 个字符构成的子串 $s_l s_{l+1} \dots s_r$。字符集大小默认为 $O(1)$。

与后缀自动机和后缀树相关的结论，可以在 [1] 找到，我们默认读者都已掌握，这里不再赘述。

定义 2.1 (出现次数)**.** 对固定串 s，和其子串 t，定义 $\mathrm{occ}(t)$ 为 t 在 s 中的出现次数，即满足 $s[l, r] = t$ 的 (l, r) 对个数。每个 $s[l, r]$ 称为一次 t 的出现位置。

定义 2.2 (后缀树)**.** 定义 T_0 为正串 *SAM* 的 *parent* 树，即反串后缀树。定义 T_1 为反串 *SAM* 的 *parent* 树，即正串后缀树。

有时称 SAM 节点时也指对应 parent 树上的节点。

定义 2.3. 对于正串 *SAM* 节点 u，定义 $s_0(u)$ 为节点 u 表示的串的集合。对于反串 *SAM* 节点 v，定义 $s_1(v)$ 为节点 v 表示的串的集合。

定义 2.4 (border)**.** 称一个串 t 为 s 的 *border* 当且仅当 t 既为 s 的前缀也为 s 的后缀。

3 基本子串结构

3.1 引入

首先引入一个问题：

例题 3.1.1. 给定串 s，q 次查询正串 *parent* 树 T_0 上的一个节点 u 和反串 *parent* 树 T_1 上的一个节点 v，求它们代表的串的交集，即 $s_0(u) \cap s_1(v)$。保证 $n, q \le 5 \times 10^5$。

对于这个问题，我们需要将正反两个串的 parent 树一起考虑，这对于普通的 SA 和 SAM 是较为困难的。所以要想办法将两个 SAM 合在一起。这就产生了如下结构，笔者将其起名为**基本子串结构**。以下内容均在固定字符串 s 下考虑。

3.2 内容

本节中将会介绍基本子串结构的内容及性质。

定义 3.1 (扩展串)**.** 对于 s 的一个子串 t，定义其扩展串 $\mathrm{ext}(t)$ 为最长的 s 的子串 t' 使得其包含 t 且满足 $\mathrm{occ}(t) = \mathrm{occ}(t')$。

需要证明这是良定的。

引理 3.2.1. $\mathrm{ext}(t)$ 存在且唯一。

证明. 存在性显然，因为 t 本身满足条件。现考虑唯一性。如果有两个串 t', t'' 均满足条件，我们有 $\mathrm{occ}(t) = \mathrm{occ}(t') = \mathrm{occ}(t'')$。而 t', t'' 又均包含 t，所以 t' 的所有出现位置与 t 的所有出现位置的一一对应。对 t'' 同理。考虑一次 t 在 s 中的出现，设为 $s[l, r]$。可以找到其对应的 t' 与 t'' 的出现位置，分别设为 $s[l', r']$ 与 $s[l'', r'']$。我们有 $l', l'' \le l \le r \le r', r''$ 且 $s[l, r] = t, s[l', r'] = t', s[l'', r''] = t''$。

设 $L = \min(l', l''), R = \max(r', r'')$，考虑 $s[L, R]$ 这个子串。由于 l, l', l'', r, r', r'' 的相对位置不取决于 l, r，所以对所有 t 的出现位置，对应的 $s[L, R]$ 均相同。即 $s[L, R]$ 被 t, t', t'' 唯一确定，且在每一个 $s[l, r] = t$ 中均会出现一次。所以 $\mathrm{occ}(s[L, R]) \ge \mathrm{occ}(t)$。又由于 $s[L, R]$ 包含 $s[l, r] = t$，$\mathrm{occ}(s[L, R]) = \mathrm{occ}(t)$。由 $\mathrm{ext}(t)$ 的最长性，一定有 $R - L = r' - l' = r'' - l''$，该式成立当且仅当 $l' = l'', r' = r''$，即 $t' = t''$。 □

从证明过程中可得到如下两个推论：

推论 3.2.1. 对于 t 与 $t' = \text{ext}(t)$，设 t 的一个出现位置为 $s[l, r]$，其对应的 t' 的出现位置为 $s[l', r']$，有 $l' \le l \le r \le r'$，且对任意 $l' \le l'' \le l, r \le r'' \le r'$，均有 $\text{ext}(s[l'', r'']) = t'$。

推论 3.2.2. 对任意子串 t，有 $\text{ext}(\text{ext}(t)) = \text{ext}(t)$。

下面根据 ext 将 s 的所有本质不同子串分类。

定义 3.2 (等价类)**.** 两子串 x, y 等价当且仅当 $\text{ext}(x) = \text{ext}(y)$。

定义 3.3 (代表元)**.** 若 $\text{ext}(t) = t$，则称 t 为其所在等价类的代表元，记为 $\text{rep}(a)$，其中 a 代表这个等价类。

引理 3.2.2. 每个等价类的代表元唯一。

证明. 由推论 3.2.2 即得。 □

引理 3.2.3. 如果两串 occ 相同，且有包含关系，则它们在同一个等价类中。

证明. 设两串为 t_1, t_2，其中 t_2 包含 t_1。那么 $\text{ext}(t_2)$ 包含 t_1, t_2，且 occ 相同。所以 $\text{ext}(t_1) = \text{ext}(t_2)$。 □

定义 3.4. 定义一个子串 t 在 s 中的第一次出现位置为 $s[\text{posl}(t), \text{posr}(t)]$。

下面的定理揭示了基本子串结构的一个重要性质。

定理 3.1. 建立以 l, r 为轴的平面直角坐标系。对于同一个等价类，所有串的 $(\text{posl}, \text{posr})$ 构成的点集在平面上形成一个上端与左端对齐的阶梯状网格图。

证明. 显然 t 的第一次出现位置对应 $\text{ext}(t)$ 的第一次出现位置。对应对于一个等价类中，长度最大的即为代表元，也即点集中左上角的点。而由推论 3.2.1，所有该点集中的点都在代表元的点的右下方（即 l 更大，r 更小），且以两点为端点的矩形内部的所有整点也在点集内。该定理即证。 □

推论 3.2.3. 平面中 $1 \le l \le r \le |s|$ 的所有整点被划分成了若干个互不相交的阶梯形，每个等价类 a 会对应其中 $\text{occ}(\text{rep}(a))$ 个相同的阶梯形。

定义 3.5. *(*周长*)* 定义一个等价类 a 的周长 $\text{per}(a)$ 为其阶梯状网格图中行数与列数之和。

由于该定理的存在，基本子串结构具有很好的几何直观。我们也称一个等价类为一块。接下来的定理揭示了基本子串结构与 SAM 的关系。

定理 3.2. 对于每一个等价类，在其对应的阶梯状网格图中，每一行（r 相同）对应一个正串 *parent* 树 T_0 的一个节点，每一列（l 相同）对应一个反串 *parent* 树 T_1 的一个节点。且所有等价类的所有行（所有列）与正串（反串）*parent* 树节点一一对应。对应指字符串的集合相同。

证明. 我们只证明行的部分，对于列的部分是对称的。对于一块中的一行，其 r 不变，l 连续变化，occ 不变，所以它们都在同一个正串 SAM 节点上。与此同时，对于某个 SAM 节点，其上所有串两两有包含关系，且 r,occ 相同，所以它们一定在同一个块中的同一列上。因此，所有块的所有列与所有 SAM 节点一一对应。 □

这也说明了如下命题：

定理 3.3. 所有块的周长之和为 $O(n)$。

最后我们需要将两棵 parent 树的树边补上去。方法也很简单，T_0 中的边可以看作是某列的上边界连向某个其他块的列的下边界，T_1 中的边可以看作是某行的左边界连向某个其他块的行的右边界。至此，我们完成了基本子串结构的构建。

从另外一个角度，基本子串结构也可以看作为一个包含所有 s 的本质不同子串的有向图，其中 t 连向所有 $t+c$ 与 $c+t$，其中 t 为一个子串，c 为一个字符。

以下考虑树边所拥有的性质。先证明一个引理。

引理 3.2.4. 如果 $s[l,r_1]$ 与 $s[l,r_2]$ 在同一个等价类中，那么对 $\forall 1 \le i < l$，$s[i,r_1]$ 与 $s[i,r_2]$ 在同一个等价类中。

证明. 不妨设 $r_1 \le r_2$。首先由包含关系有 $\text{occ}(s[i,r_2]) \le \text{occ}(s[i,r_1])$。但每一次 $s[i,r_1]$ 的出现都伴随着一次 $s[l,r_1]$ 的出现，而 $s[l,r_1]$ 的出现与 $s[l,r_2]$ 的出现一一对应。所以若 $s[p,p+r_1-i] = s[i,r_1]$，有 $s[p+l-i,p+r_1-i] = s[l,r_1]$，从而 $s[p+l-i,p+r_2-i] = s[l,r_2]$，最后得到 $s[p,p+r_2-i] = s[l,r_2]$。所以 $\text{occ}(s[i,r_1]) \le \text{occ}(s[i,r_2])$。由引理 3.2.3 即证。 □

下面的定理揭示了等价类在 SAM 上的一个重要性质。

定理 3.4. 如果两个 *SAM* 节点 u,v 在同一个等价类中，那么这两个点在 *parent* 树上的子树除根节点外结构完全相同。结构完全相同指可将子树内的点一一对应，使其作为无根树同构，且对应的点满足：包含的串的个数，occ 大小均相同。进一步的，对应的 *SAM* 节点在阶梯状网格图的相对位置（两列/行之间的距离）均与 u,v 在阶梯状网格图的相对位置（两列/行之间的距离）相同。

证明. 不妨设 u,v 在正串 SAM 上。设 u,v 包含的串中最长的分别为 b_u,b_v。不妨设 $\text{len}(b_u) \le \text{len}(b_v)$。由于 u,v 在同一块中，且等价类对应的阶梯状网格图左边界对齐，有 $\text{posl}(b_u) = \text{posl}(b_v)$。

设 $k = \mathrm{occ}(b_u), \delta = \mathrm{len}(b_v) - \mathrm{len}(b_u)$，令 b_u, b_v 的所有出现位置分别为 $s[l_1, r_1], \ldots, s[l_k, r_k]$ 与 $s[l_1, r_1 + \delta], \ldots, s[l_k, r_k + \delta]$。由引理 3.2.4，$\forall 1 \le i \le k, 1 \le j < l_i$，$s[j, r_i]$ 与 $s[j, r_i + \delta]$ 在同一个等价类中，且在网格图中的相对位置与 b_u, b_v 间相同。而 u, v 子树中包含的串恰恰就是这些。因此，我们构造了一个串与串之间的一一映射，使得 trie 树作为无根树同构，且对应点 occ 相同。而 parent 树即为将 trie 的 2 度点缩起来后得到的树，因此定理正确。 □

推论 3.2.4. 可以把一个块的上边界连出的边分成若干组，每组会连向某个其他块一段连续且对齐的下边界。组数即 *parent* 树上的儿子个数。对于左边界同理。

证明. 分组即为上述定理中作为同构无根树对应的边。由于相对位置相同，组内的 SAM 节点为连续的若干列，每组中连向的 SAM 节点也为连续的若干列。而它们包含的串个数相同，所以连向的 SAM 节点下边界对齐。 □

该引理同样拥有几何直观。有了该引理，自动机上的链就很容易识别了。

引理 3.2.5. 对于一个 T_0 上的点，如果其不对应其所在块的左边界，那么它在 *SAM* 上有且仅有一条出边，连向它左侧的列所对应的节点；如果其对应的是左边界，则它的所有出边即连向从该左边界用另一棵 *SAM* 的 *parent* 树边连出能到的所有左边界表示的 *SAM* 节点。

证明. 证明可通过上述定理与 SAM 的基本性质得到。 □

至此，我们完整包含了两个 SAM 的所有信息，也完成了基本子串结构的构建。整体上讲，基本子串结构由所有等价类与 parent 树边构成，而等价类对应的阶梯状网格图为该结构的核心性质。SAM 的自动机结构已经被完全抛弃，因为其作为 DAG 性质不够优秀，转而被另一个 SAM 上的 parent 树所替代。

3.3 构造

本节将会介绍一个 $O(n)$ 时间内构建基本子串结构的算法。

定理 3.5. *SAM* 可以在 $O(n)$ 时间复杂度内构建。

由于该定理不是本文的主题，我们略去其证明。

定理 3.6. 基本子串结构可以在 $O(n)$ 时间复杂度内构建。

证明. 首先建出正反串的 SAM。对于任意等价类 a，$\mathrm{rep}(a)$ 一定为一个节点中的最长串。而因为有引理 3.2.5 的存在，识别一个节点中的最长串是否是代表元是容易的，只需检查其在自动机上的出边即可。在找到所有代表元后，同样可依靠引理 3.2.5 找到每个 SAM 节点属于哪个代表元表示的等价类。

将两个 SAM 的等价类对应起来只需分别找到代表元的 posl, posr 即可。每块中网格图的构建只需将所有 SAM 节点按照 r 或 l 进行排序，但其天然与 SAM 构建时加入的顺序相同。至此所有信息都已构建完毕，总时间复杂度为 $O(n)$。 □

该实现容易简洁，常数较大但复杂度优秀。

3.4 应用

先处理最开始引入的问题。介绍完基本子串结构，例题 3.1.1 的解法呼之欲出。

解法 建出基本子串结构。如果两点不在同一个块内，则答案为空集；否则考虑该块的阶梯状网格图，$s_0(u) \cap s_1(v)$ 即为求一条横线段与一条竖线段的交点。容易在 $O(1)$ 时间复杂度内解决。总时间复杂度 $O(n+q)$。 ■

接下来回顾若干旧的字符串问题，我们可以用基本子串结构将其统一起来。

例题 3.4.1 (打击复读[1])**.** 给定长度为 n 的字符串 s，第 i 个字符有两个权值：**左权值** wl_i 和**右权值** wr_i 。定义一个子串 $s[l,r]$ 的**左权值** $vl(s[l,r])$ 为：其在原串中各个出现位置的左端点的**左权值** wl 和；**右权值** $vr(s[l,r])$ 为：其在原串中各个出现位置的右端点的**右权值** wr 和。定义一个子串 $s[l,r]$ 的复读程度是它的**左权值**与**右权值**的乘积，即 $w(s[l,r]) = vl(s[l,r]) \cdot vr(s[l,r])$ 。

s 的复读程度定义为所有子串复读程度的和，即：

$$\sum_{i=1}^{|S|}\sum_{j=i}^{|S|} w(s[i,j])$$

q 次修改某个 wl_i，查询整串的复读程度，对 2^{64} 取模。

解法 显然答案是一个关于 wl 的固定线性函数。由于存在对于 wl 的修改，我们需要将函数的系数求出。建出 SAM 后容易发现，$vr(s[l,r])$ 即为子树中所有 $s[1,i]$ 的 wr_i 之和。同理 $vl(s[l,r])$ 即为子树中所有 $s[i,n]$ 的 wl_i 之和。wr 始终固定，可预处理求出。

建出基本子串结构，考虑所有块分别的贡献。观察阶梯状网格图，相当于每行有一个权值 vl，每列有一个权值 vr，求网格图中所有格子对应的行和列的权值乘积之和。可以对每个 vl，用前缀和找到其对应的 vr 之和。这样就把答案写为了所有 vl 的线性组合。再通过在反串 SAM 上做一次递推，将答案变为所有 wl 的线性组合。有了线性组合的系数，自然很容易支持修改。时间复杂度 $O(n+q)$。 ■

值得一提的是，本题原先的做法中包含对称压缩 SAM，而对称压缩 SAM 本质上即为把基本子串结构的一个等价类看成一个点，形成两个方向的自动机结构。其忽略了重要的阶梯状网格图结构，但也激发了将两个 SAM 合并考虑的想法。

例题 3.4.2 (广为人知题[2])**.** 给定长度为 n 的字符串 s。给定 m 个 s 的子串作为模式串。q 次查询，每次给出 $\mathrm{ql}_i, \mathrm{qr}_i$，求所有模式串在 $s[\mathrm{ql}_i, \mathrm{qr}_i]$ 中的出现次数之和。

[1]来源：https://uoj.ac/problem/577

[2]来源：https://uoj.ac/problem/697

解法 t_2 在 t_1 中的出现次数即为有多少个 t_2 的后缀的前缀是 t_1。建出基本子串结构，首先求出每个串有多少个前缀是模式串。即为设定所有模式串为 1 其余为 0，然后沿反串 parent 树，即阶梯状网格图中的横向求树上前缀和。然后求出每个串有多少个后缀的前缀是模式串，即为延续前面的权值，再沿正串 parent 树，即阶梯状网格图的纵向求树上前缀和。最后查询 q 个单点的答案。

但点数最多为 $O(n^2)$，需要进一步优化。考虑阶梯状网格图。首先将所有模式串在其所在行（即其所在 SAM 节点）造成的贡献删去，只考虑其对 parent 树子树中除所在点以外的点造成的贡献。最后再把对其所在行的贡献补上。

删去后，在第一次前缀和完成后，每行中所有点的权值均相同。第二次前缀和后，单点查即为在该点前面的所有点所在行的权值之和。块外的部分可简单前缀和转移预处理得到。块内的部分也是一个区间求和，仍然可以用前缀和解决。

现考虑之前删去的部分。删去的部分是若干个形如一行中一个后缀加 1 的贡献。其对于块外答案的贡献是容易的，可与前一部分合并转移。对于块内答案的贡献为一个矩形加 1，所以离线查询后对每个块做一次二维数点即可。

在 SAM 上定位串需要 $O((m+q)\log n)$ 时间，二维数点需要 $O((m+q)\log n)$ 时间，其余部分为线性。总复杂度 $O(n+(m+q)\log n)$。 ■

本题原做法中用到 DAG 剖分来处理第二次前缀和。做法中要在每条 DAG 剖分的重链上二维数点，需要 $O(q\log^2 n)$ 的复杂度。通过运用基本子串结构，我们能降低时间复杂度，得到一个容易实现的单 log 做法。

从另一个角度来看，本题统计的是“后缀的前缀个数”，如果将题目加强为统计“后缀的前缀的后缀的前缀个数”，基本子串结构仍能在同样时间复杂度内解决，而 DAG 剖分则会对此束手无策。这种加强方式实际上是如下题目的原型。

例题 3.4.3 (子串统计[3])**.** 给定长度为 n 的字符串 s。令 $T_0 = s$。每次删除 T_i 的开头或结尾的字符，得到新的字符串 T_{i+1}，经过 $n-1$ 次操作之后，会得到只有一个字符的串 T_{n-1}。根据每次删除的选择，一共有 2^{n-1} 种可能的操作序列。

对于一个操作序列，记其权值为

$$\prod_{i=1}^{n-1} \mathrm{occ}\,(T_i)$$

求出所有操作序列的权值和，对 998244353 取模。

解法 简单在以 l, r 为轴的二维平面 dp 即可得到一个 $O(n^2)$ 时间的做法。考虑优化该 dp。首先改变 dp 顺序，改为从空串开始，不断在两端加字符，最后加到 s。那么对于出现在不同位置的相同子串，其 dp 值均相同。注意到添加字符过程中 occ 单调不增，考虑按照 occ 递减的顺序枚举等价类进行 dp 转移。

[3]来源：https://loj.ac/p/3723

注意到在同一个等价类中 occ 不变，且 dp 转移可看作二维平面上阶梯状网格图上的转移。为继续向接下来的等价类转移，我们需要求出阶梯状网格图的上边界与左边界的 dp 值。

那么问题被转化成：在阶梯状网格图上，右下的阶梯状轮廓线上有若干起点，需要对每个上边界与左边界的点求出与所有起点的路径个数之和。

其生成函数大致为 $\sum \frac{(\mathrm{occ}\cdot x)^j}{(1-\mathrm{occ}\cdot x)^i}$，其中 i, j 均单调，且最大值总和不超过 per。可分治 FFT 计算。

在一个等价类 a 内的复杂度为 $O(\mathrm{per}(a)\log^2 n)$，因此总复杂度为 $O(n\log^2 n)$。 ■

阶梯状网格图结构为本题解法中的核心。需要高度利用阶梯状网格图的性质，才能使用分治 FFT 优化复杂度，这对于一般后缀数据结构是不可能做到的。

例题 3.4.4 (复读程度[4])**.** 给定长度为 n 的字符串 s，第 i 个字符被赋予两个权值：**左权值** wl_i 和**右权值** wr_i。定义一个子串 $s[l,r]$ 的**左权值** $vl(s[l,r])$ 为：其在原串中各个出现位置的左端点的**左权值** wl 和；**右权值** $vr(s[l,r])$ 为：其在原串中各个出现位置的右端点的**右权值** wr 和。定义一个子串 $s[l,r]$ 的复读程度是它的**左权值**与**右权值**的乘积，即 $w(s[l,r]) = vl(s[l,r])\cdot vr(s[l,r])$。

q 次查询 l_1, r_1, l_2, r_2，令 $S = \{(l,r) \mid r-l \geq \max\{r_1-l_1, r_2-l_2\}, s[l, l+r_1-l_1] = s[l_1, r_1], s[r-r_2+l_2, r] = s[l_2, r_2]\}$，即以 S 为所有满足 $s[l_1,r_1]$ 为 $s[l,r]$ 的前缀，$s[l_2,r_2]$ 为 $s[l,r]$ 的后缀的 (l,r) 的集合。求 $\sum\limits_{(l,r)\in S} w(s[l,r])$，对 2^{64} 取模。

解法　与 3.4.1 相同，vl, vr 可在正反串 SAM 上分别 dfs 一遍求出。直观上看，以 $s[l_1,r_1]$ 为前缀的串均在反串 parent 树上 $s[l_1,r_1]$ 的子树中，以 $s[l_2,r_2]$ 为后缀的串均在正串 parent 树上 $s[l_2,r_2]$ 的子树中，一次查询即为找出这两个子树的交集信息。但在 $s[l_1,r_1], s[l_2,r_2]$ 所在的 SAM 节点上会有一部分串不应被包含。因此，我们可以把一次查询中的 S 可分为两部分的差：一部分为两子树的交集；第二部分为满足在两子树交集中但却不在 S 中（即在 $s[l_1,r_1]$ 或 $s[l_2,r_2]$ 所在的 SAM 节点上但长度小于 r_1-l_1+1 或 r_2-l_2+1）。

首先考虑第二部分。分别算出在 $s[l_1,r_1]$ 与 $s[l_2,r_2]$ 的答案，再去掉交集。交集很容易，放在网格图上与引入时的例题 3.1.1 相同。再考虑分别的答案。两侧对称，现只考虑其中一侧。问题被转化为：求出有多少在正串 parent 树上与 $s[l_2,r_2]$ 在同一节点，但长度 $< r_2-l_2+1$，且在反串 parent 树上 $s[l_1,r_1]$ 的子树中。在网格图上即为一个二维数点问题。该部分时间复杂度 $O(q\log n)$。

接下来讨论第一部分。使用 dfs 序，并差分，变成查询两个 dfs 序前缀的交集。由于网格图的优秀性质，可直接莫队二次离线求解。需要实现一个 $O(\sqrt{n})$ 单点修改 $O(1)$ 区间求和的数据结构。时间复杂度 $O(n\sqrt{q}+q)$，空间复杂度 $O(n+q)$。

总时间复杂度 $O(n\sqrt{q}+q\log n)$，空间复杂度 $O(n+q)$。 ■

[4]来源：https://qoj.ac/problem/5014

4 基本子串结构上的树链剖分

4.1 内容

本节中尝试解决对于 SAM 上一条链的查询。以下描述均在正串 parent 树 T_0 上。对于另外一侧可对称处理。

定义 4.1 (重儿子)**.** 非叶节点 u 的重儿子 son(u) 为子节点中 siz 最大的一个，若多个相同，取其中 posr 最大的一个。

定义 4.2 (轻重边)**.** 所有非叶节点 u 和点 son(u) 间连接的树边称为重边。其余边称为轻边。

由这样选取重儿子的方式构造树链剖分，我们有如下性质：

定理 4.1. 在推论 *3.2.4* 中提到的任意一个组中，要么同时为重边，要么同时为轻边。

证明. 由定理 3.4，这些子树完全一致，且 posr 相对位置相同，所以子树内所有非叶点的重儿子也对应相同。 □

推论 4.1.1. 在推论 *3.2.3* 中提到，每个等价类会占据 occ 个不同位置。可以对所有等价类选择一个出现位置，使得每个非叶节点 u 对应的阶梯状网格图上的横线的左边界与 son(u) 对应的阶梯状网格图上的横线的右边界重合。

证明. 从叶子开始递归选取。首先叶子选取方式唯一，因为其 occ = 1。对于非叶节点，找到其重儿子选取位置，选择在其右端对应的位置即可。 □

我们相当于重新赋予了每个等价类中点的坐标，使得查询一条 T_0 上的链被转化为了查询 $O(\log n)$ 个 r 坐标相同的直线。同样的，对 T_1 上对链查询可以被转化为查询 $O(\log n)$ 个 l 坐标相同的直线。但需要注意，在两棵树上树剖产生的坐标并不相同。

定理 4.2. 将重链按照其拥有的叶节点的 posr 标号，则每个等价类中的 *SAM* 节点所在重链标号为一段连续区间。

证明. 叶节点的 posr 即为对应横线的 r 坐标。定理显然成立。 □

4.2 应用

本节中将会尝试用该结构解决部分问题。首先是经典的基本子串字典。

例题 4.2.1 (基本子串字典[5])**.** 给定长度为 n 的字符串 s。q 次询问，每次给定一个 s 的子串，查询其最短 *border*、最长 *border* 和 *border* 个数。n, q 同阶。

[5]https://acm.nflsoj.com/problem/338

解法　设查询串为 $s[l,r]$。把 border 看为 $s[l,r]$ 的后缀与 $s[l,r]$ 的前缀的交集。而 $s[l,r]$ 的后缀即为一条 T_0 上到根的链，$s[l,r]$ 的前缀即为一条 T_1 上到根的链。所求即为链交的信息。

将所有重链信息找出，由于有交集需要长度区间有交集，所以只有 $O(\log n)$ 对重链对需要查询交集。每个重链对的查询内容为：有多少长度 $\le l$ 的串既在 T_0 标号为 x 的重链上，也在 T_1 标号为 y 的重链上。

考虑如何提供所有串的信息。我们只需分别考虑 SAM 节点即可。对于一个 T_0 的结点，由定理 4.2，其上所有串所在的 T_1 重链标号连续，所以这些串对应一个在以 T_1 重链标号，长度为轴的二维平面上的一条斜率为 1 的线段。

问题转化为一个 $O(q\log n)$ 次查询，$O(n)$ 次修改的的简单二维数点。总时间复杂度 $O\left(\frac{q\log^2 n}{\log\log n}\right)$。

■

我们用基本子串结构给出了第二种求出 border 序列的方式，但付出了额外的一个 $\log n$ 的代价。随之换来的好处是：我们能够定位这些 border 在 SAM 上的位置。一个很简单的修改，比如改为查询区间所有 border 的 occ 之和，原先的 SA 做法就会宣告破产。但实际上，这个修改后的问题仍能用 DAG 剖分在 $O(q\log^2 n)$ 时间内解决。由于这两个做法都不是本文的主题，在这里不做讨论。

那么 DAG 剖分是否能够覆盖基本子串结构所能做的事情？答案是否。下面这个例题是 border 问题上基本子串结构强于 DAG 剖分的有力证据。

例题 4.2.2. 给定长度为 n 的字符串 s。给定 $f_1,\ldots,f_n$ 与 $g_1,\ldots,g_n$。q 次询问，每次给定一个 s 的子串，查询其所有 *border* 的 $f_{\text{occ}}\cdot g_{\text{len}}$ 之和。n,q 同阶。

解法　与上题相同，首先转换为 $O(q\log n)$ 次查询：有多少长度 $\le l$ 的串既在 T_0 标号为 x 的重链上，也在 T_1 标号为 y 的重链上。

这时由于需要让 occ 与 len 同时保持一致，我们可以把统计每个 SAM 节点改为统计每个等价类中的斜线。具体来说，对于每个等价类 a 对应的阶梯状网格图，其每一条 $r-l$ 相同的斜线对应一系列 occ 与 len 均相等的子串。而这样斜线的个数容易证明不超过 $O(\text{per}(a))$。因此斜线总数不超过 $O(n)$。

问题同样被转化为一个 $O(q\log n)$ 次查询，$O(n)$ 次修改的的简单二维数点。总时间复杂度 $O\left(\frac{n\log^2 n}{\log\log n}\right)$。 ■

该解法利用了阶梯状网格图的性质，使得 occ 与 len 同时保持相同。趣味的是，虽然 DAG 剖分与 SA 无法解决该问题，但笔者还是发现了本题的第二个做法。利用 SA 找出 border 的 $O(\log n)$ 段等差数列，再结合 runs 的优美性质，就能够得到一个 $O(q\log^2 n)$ 的做法。但由于这与本文主题无关，在此略去细节。

5 总结

本文介绍了一种新的数据结构：基本子串结构。相比普通的 SA、SAM 而言，该结构功能强大，能同时处理前后缀字符串问题。从某种程度上讲，基本子串结构为对称压缩 SAM

的拓展，将对称压缩 SAM 中一个点详细扩充成了一个详细的阶梯状网格图，使其拥有更多的信息和可操作性。与此同时，该结构易于实现，复杂度优秀，在信息学竞赛中具有广泛的应用前景。

但字符串理论中还有很多的问题悬而未决。希望本文能够启发读者继续深入研究相关理论，使其更加成熟。

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的关心和指导。
感谢家人、朋友对我的支持与鼓励。
感谢戴江齐同学与我讨论以及给予我的启发。
感谢杜瑜皓学长，常瑞年同学、叶子川同学、吴航同学为本文审稿。

参考文献

[1] OI-wiki 贡献者. 后缀自动机 (SAM) — OI-wiki, 2022.

[2] 与戴江齐同学的私人交流, 2022.

[3] 徐翊轩. 浅谈压缩后缀自动机. IOI2020 中国国家候选队论文集, 2020.

几个经典数论问题的再探讨

长沙市雅礼中学 郑玄晔

摘 要

本文主要探讨了三个古老的数论问题：素性判定、整数分解和离散对数。回顾了 OI 中已被推广的几种算法并介绍了它们的一些特殊性质，并引入了几种目前尚未在 OI 中广泛使用的优秀算法。

1 概述

1.1 前言

素性判定、整数分解和离散对数三个问题已经是非常经典的数论问题了。然而，作者观察到，在目前的 OI 环境中，大多数选手对相关的算法还知之甚少，甚至有部分高水平选手也是如此。在查阅相关资料的过程中，也常常遇到许多相互矛盾和具有事实性错误的文章。出于促进 OI 进一步发展的愿望，本文介绍了数个上述问题的相关算法，并给出了使用上的一些建议。

1.2 基本定义

一个整数 n 的一个非平凡因子 d，是使得 $d|n$ 且 $1 < d < n$ 的数。

素数是指没有非平凡因子的正整数，合数指有非平凡因子的正整数。特别规定 1 既不是素数，也不是合数。

smooth-number 是指每个质因子大小都很小的数。如果一个数的每个质因子大小都不超过 B，则称这个数是 B-smooth 的。

渐进记号 $\tilde{O}(f(n))$ 表示存在常数 k 使得渐进复杂度为 $O(f(n)\log^k f(n))$。

$L_n[\alpha, c]$ 是一种类似大 O 记号的渐进记号，定义为 $L_n[\alpha, c] = \exp((c+o(1))(\ln n)^{\alpha}(\ln\ln n)^{1-\alpha})$，其中 $0 \le \alpha \le 1, c > 0$ 为常数。注意到 $L_n[0, c] = (\ln n)^{c+o(1)}$ 为关于 $\ln n$ 的多项式级别，而 $L_n[1, c] = n^{c+o(1)}$ 为关于 $\ln n$ 的指数级别，可以发现当 $\alpha \in (0, 1)$ 时是关于 $\ln n$ 的一个亚指数级别。

2 素性判定问题

2.1 问题描述

素性判定问题是指，输入一个正整数 p，判断其是否为素数。

2.2 Miller-Rabin 算法

2.2.1 算法原理及流程

Miller-Rabin 算法通过检验一个数是否满足素数应有的性质，来判定一个数是否是素数。更具体的，考虑费马小定理的逆否命题，也即，对于待判定的数 p 和一个整数 a 使得 $1 \le a < p$，若 $a^{p-1} \not\equiv 1 \pmod{p}$，则 p 不是素数。这个过程称为“费马测试”。

费马测试的问题在于，若 $a^{p-1} \equiv 1 \pmod{p}$，并不能据此判定 p 就是素数。然而，注意到若 p 确实是素数，则此时必有 $a^{(p-1)/2} \equiv 1 \pmod{p}$ 或 $a^{(p-1)/2} \equiv p-1 \pmod{p}$。因此，可以进一步检查 $a^{(p-1)/2} \bmod p$，若其不为 1 或 $p-1$，则可以判定 p 不是素数。

于是我们可以费马测试的基础之上，进行二次探测。类似的，若 $(p-1)/2$ 为偶数，而 $a^{(p-1)/2} \bmod p$ 仍为 1，可以通过检查 $a^{(p-1)/4} \bmod p$ 的值，来探测 p 是否不是素数。

总结上述过程并加以推广，便得到 Miller-Rabin 测试。形式化的，称如下过程为“对 p 进行一次以 a 为底数的 Miller-Rabin 测试”：

1. 若 p 是 2，返回 p 是素数；若 p 为不是 2 的偶数或 1，返回 p 不是素数。
2. 初始化 $k = p-1$。
3. 若 $a^k \not\equiv 1 \pmod{p}$，返回 p 不是素数。
4. 计算 $r = a^{k/2} \bmod p$。
5. 若 $r \not\equiv 1 \pmod{p}$ 且 $r \not\equiv -1 \pmod{p}$，返回 p 不是素数。
6. 若 $r \equiv 1 \pmod{p}$ 且 $k/2$ 是偶数，则令 $k \leftarrow k/2$ 回到 4.。
7. 返回 p 是素数。

实现之时，我们需要计算 $a^{p-1}, a^{(p-1)/2}, a^{(p-1)/4}, \ldots$。若直接使用分治快速幂计算每一个复杂度较高。我们可以将 $p-1$ 分解成 $p-1 = 2^l t$，其中 t 是一个奇数。这样，只需使用分治快速幂计算出 a^t 后，逐项递推出 $a^{2t}, a^{4t}, a^{8t} \ldots$，便不必多次调用快速幂。这样，一次测试所需的时间代价是 $O(\log p)$ 次数字乘法和取模所需的时间。

2.2.2 算法性质

Miller-Rabin 测试是一个单边错误的概率算法，当它返回 p 不是素数时，p 一定不是素数；但当它返回 p 是素数时，p 实际上不一定是素数。

一个简单但行之有效的改进方式是，多选取几个底数，分别进行 Miller-Rabin 测试，只有在全部返回 p 是素数时才认为 p 是素数。这便构成了 Miller-Rabin 算法，可是它看上去仍然不能令人信服。好在有如下两条定理来让其成为一个靠谱的算法。

定理 1. 在扩展黎曼猜想 *(extended Riemann hypothesis，ERH)* 成立的假设下，下述素性测试算法是正确的：以 $[1, 2\ln^2 p]$ 中的每个整数 a 作为底数对 p 分别进行 *Miller-Rabin* 测试。[1]

定理 2. 若 p 是合数，在 $[1, p)$ 中至少有 $\frac{3(p-1)}{4}$ 个数可以成为 p 是合数的 *Miller-Rabin* 证据，也即以其作为 *Miller-Rabin* 测试的底数时，算法会返回 p 不是质数。[2]

两条定理的证明过于复杂，在此略去。

虽然扩展黎曼猜想是一个未经证明的猜想，但是大多数学者相信它很有可能是对的。在其成立的假设下，定理 1 指出，Miller-Rabin 算法可以成为一个确定性的，多项式时间的素性测试算法。

不过，Miller-Rabin 算法更为常见的版本是随机选取若干底数进行测试的随机化版本。根据定理 2，只要随机选取 k 个数作为底数分别进行测试，即可获得至少 $1 - 4^{-k}$ 的正确率。

在 OI 中，通常无需对特别大的数进行测试，此时可以选取一些特定的底数来进一步加速测试。例如，选取 2,3,7,61,24251 这组底数，则在 10^{15} 的范围中，只有 46856248255981 这个数会测试出错。有兴趣的读者可以在网络上搜索更多具有其他性质的底数。

2.3 其他算法

Solovay–Strassen 算法是一个比 Miller-Rabin 提出时间更早的算法，具有一定的历史意义。它的性质和 Miller-Rabin 类似，是一个单次测试具有至少 50% 正确率的概率算法。不同的是，它需要计算雅可比符号 (Jacobi symbol)，因而在效率上逊于优化后的 Miller-Rabin 算法，现已经被 Miller-Rabin 淘汰。

AKS 算法是历史上第一个确定性的、不依赖任何未经证明的猜想的、多项式时间复杂度（这里指的是和输入长度呈多项式关系，也即与 $\ln p$ 成多项式关系）的素性判定算法，具有很重要的理论意义。然而由于其步骤复杂，效率低下，很少得到实际应用。

[1] Miller, Gary L. (1976), "Riemann's Hypothesis and Tests for Primality"

[2] Rabin, Michael O. (1980), "Probabilistic algorithm for testing primality"

3 整数分解问题

3.1 问题描述

整数分解问题是指，输入一个合数 n，找到它的一个非平凡因子。

整数分解问题的算法可分为针对特殊情况下的算法，和一般情况下都能工作的算法。我们将从二者中各选取一个具有代表性且较为实用的算法进行介绍。

3.2 Pollard p-1 算法

Pollard p-1 算法是从对 smooth-number 的考虑出发的。如果待分解数 n 有一个不大的质因子 $p \le B$，我们试图构造一个数使得它具有 p 作为因子，然后将其与 n 取 gcd 即分解成功。

费马小定理再次生效了，它指出对于一个随意的 $a \bmod p \neq 0$，$a^{p-1}-1$ 这个数是满足条件的，但是我们不知道 p。注意到取一个 M 使得 $(p-1)|M$ 的话，a^M-1 也是可行的。而我们知道 $p-1<B$，因此可以取 $M=\prod_{prime\ q<B} q^{\lfloor \log_q B \rfloor}$，这样就必定有 $(p-1)|M$ 了。

于是 Pollard p-1 算法的流程是异常简单的，选取合适的 a,B，计算 $\gcd(a^M-1,n)$ 即可。实际应用中常常取 $a=2$。比较值得注意的就是 B 的取值，太大了会导致计算过慢，太小了又可能使得存在 $p \le B$ 的条件遭到破坏，因而可以动态增大 B 来提高成功的可能性。

可以发现，这个算法不一定能成功分解，而在一些特殊情况下（如 n 是两个大素数的乘积），更是毫无用武之地。这就是针对特殊情况下设计的算法的缺点所在。但是，相比于其他通用算法，它的效率十分之高，可以与通用算法结合使用。

3.3 Pollard Rho 算法

由于 Pollard Rho 算法在 OI 中的普及程度已经很高，本文不再仔细介绍 Pollard Rho 的原理，而仅简述其流程和一些性质以助于读者理解。

（使用倍增优化的）Pollard Rho 算法的流程如下：

1. 任意选取常数 c，作为函数 $f(x)=(x^2+c) \bmod n$ 的参数。并确定一正整数常数 B。
2. 初始化 s,t 为 $[0,n)$ 之间的同一个随机整数。初始化倍增长度 $len=2$，循环变量 $i=0$，累乘变量 $q=1$。
3. 令 $t \leftarrow f(t)$，并作累乘 $q \leftarrow q \times |t-s| \bmod n$。
4. 若 $i \equiv 0 \pmod B$，计算 $\gcd(q,n)$。若结果不为 1，返回结果 $\gcd(q,n)$。
5. 令 $i \leftarrow i+1$。若 $i=len$，则令 $len \leftarrow 2 \times len, s \leftarrow t, i \leftarrow 0$ 进入倍增的下一轮。
6. 回到 3.。

注意 Pollard Rho 算法有可能返回 n，此时并未成功找到 n 的非平凡因子，应当视作算法运行失败并重新运行算法。实际运行中，若正确处理了这种情况，则其不会对运行效率产生过大的影响。

算法流程中迭代生成了一个伪随机数列，每个数有唯一后继，因而形成一个“ρ”形，算法名字也由此而来。值得注意的是，Pollard Rho 算法中的伪随机函数不是随意选取的，而是具有一定的性质。具体的，若有一非平凡素因子 $p|n$，使得 $t - s \equiv 0 \pmod{p}$，那么，亦有 $f(t) - f(s) \equiv 0 \pmod{p}$。因此在“$\rho$”形的环上，若存在两个相距为 d 的数，使得算法能找到因子 p 或 p 的倍数，那么任意两个相距为 d 的数也能使算法找到非 1 的因子。本文之所以强调这一性质，是由于作者发现许多 Pollard Rho 的使用者其实并不了解这一性质是 Pollard Rho 算法生效的原理之一。

Pollard Rho 算法的运行时间不稳定，波动也比较大。若其使用的伪随机函数 $f(x) = (x^2 + c) \bmod n$ 足够均匀，而算法流程中参数 B 取 $\Theta(\log n)$ 级别，则其期望时间复杂度为 $O(p^{1/2})$（其中 p 为 n 的最小素因子），也即不超过 $O(n^{1/4})$。

可惜的是，这个伪随机函数是否足够均匀仍是一个开放性问题 (open problem)。不过，在大多数情况下，Pollard Rho 算法的表现都不错，因而成为一种得到广泛应用的通用算法。

3.4 其他算法

一般数域筛选法 (general number field sieve,GNFS) 是目前已知理论复杂度最好的整数分解的传统算法，其复杂度为 $L_n[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}]$。但由于其实现过于复杂，OI 中是不太可能使用的。

Shor 算法是一个具有多项式时间复杂度的量子算法，复杂度可以优化到 $\tilde{O}(\log^2 n)$。由于量子计算机尚未得到足够的发展，目前已知该算法分解的最大数仅为 21。

4 离散对数

4.1 问题描述

在一般群上定义的离散对数问题是，给定一个群 G 和其一个生成元 g，输入一个元素 $b \in G$，要找到一个整数 x 使得 $g^x = b$，也记为 $x = \log_g b$。通常记群的大小 $|G| = n$。

有一类特殊的离散对数问题，即在 $\mathbb{Z}_p^\times$（模素数 p 意义下整数乘法群）内的离散对数问题。此时 g 即为模 p 意义下的一个原根。由于在 OI 中需要解决的离散对数问题绝大多数属于这一类，且这类问题得到了更加深入的研究，我们更加关心这一类问题。同时整数乘法群的性质也是更为我们所熟悉的。后文如无特殊说明，算法可以在一般群上工作，但读者不妨在 $\mathbb{Z}_p^\times$ 中理解，可能会减少一定的理解障碍。

4.2 BSGS 算法

BSGS 是目前 OI 中使用最广泛的离散对数算法，我们先简述其算法流程。

1. 选定一个阈值 B。

2. 将 $g^0, g^1, g^2, \ldots, g^{B-1}$ 插入哈希表中。

3. 枚举 $i = 0, 1, 2, \ldots$，在哈希表中查找 $b \cdot g^{-iB}$。当成功查找到 g^c 时，返回答案为 $iB + c$。

当取 $B = \Theta(\sqrt{n})$ 时，可得到 $O(\sqrt{n})$ 的时间复杂度。

由于其强大的通用性、简单的算法流程和稳定的效率，BSGS 十分受欢迎。可惜的是，它的时间复杂度并不够优秀，因而效率不是很高。

4.3 Pollard Rho 离散对数算法

这里特别要区分先前用于整数分解的 Pollard Rho 算法，它们的名字相同，但是用于解决不同的问题。

如果能找到四个数 $\alpha_1, \beta_1, \alpha_2, \beta_2$ 使得 $g^{\alpha_1} b^{\beta_1} = g^{\alpha_2} b^{\beta_2}$ 而 $\beta_1 \not\equiv \beta_2 \pmod{n}$ 时，由于 $g^x = b$，我们得到 $\alpha_1 - \alpha_2 \equiv x(\beta_2 - \beta_1) \pmod{n}$，就可以通过扩展欧几里得算法解出 x。

Pollard Rho 离散对数算法的精髓是通过构造随机数列产生这样的碰撞。生成一个三元组序列 (v_i, α_i, β_i) 满足 $v_i = g^{\alpha_i} b^{\beta_i}$，其中 (v_i, α_i, β_i) 由 $(v_{i-1}, \alpha_{i-1}, \beta_{i-1})$ 按某种规则生成。如果我们相信这个序列足够随机，则这个序列期望生成 $O(\sqrt{n})$ 个元素后就会出现相同的一对 v_i, v_j，于是可以终止并检查是否能解出答案。

而具体的，该算法由 (v, α, β) 生成下一个三元组的规则如下：

1. 在算法的开始，将整个群 G 预先分成大小大致相等的三部分 S_1, S_2, S_3。

2. 若 $v \in S_1$，生成 $(v \cdot g, \alpha + 1, \beta)$。

3. 若 $v \in S_2$，生成 $(v \cdot b, \alpha, \beta + 1)$。

4. 若 $v \in S_3$，生成 $(v^2, 2\alpha, 2\beta)$。

值得注意的是，虽然第一步划分集合可以十分随意，但仍应保证 $1 \notin S_3$，否则可能会出现算法死循环的情况。

很遗憾的是，作者没能找到相关资料说明这个函数的随机程度如何。不过实际中它确实表现优秀，往往能以比较快的速度出现碰撞。

Pollard Rho 离散对数算法不难实现，效率也还不错，与 BSGS 不相上下，不失为一种优秀的选择。

但是 Pollard Rho 算法有一个十分严重的问题：在算法最后求解方程 $\alpha_1 - \alpha_2 \equiv x(\beta_2 - \beta_1) \pmod{n}$ 时，有可能出现 $\beta_2 - \beta_1$ 与 n 不互质的情况，这会使算法无法求出唯一的解。

在一些 n 为质数的应用场景，这不是什么问题，因为这种情况出现得非常少，仅在 $\beta_1 \equiv \beta_2 \pmod{n}$ 时有，往往重新选择种子、再运行一次算法就能解决。但是在 OI 中，求解的多为 $n = \varphi(p) = p - 1$ 时的问题，n 必然不是质数。此时重新运行很可能也于事无补，因为不互质的情况很多。即使最终幸运的得到了结果，也已经花费了远超预计的时间。

作者实现过一份代码，以 OI 中常用的模数 998244353 和 $10^9 + 7$ 作为乘法群的模数进行测试，发现 Pollard Rho 算法往往重新运行了数千次也不能求出解。有时它好不容易运行结束时，已经花费了比 BSGS 算法高出数个数量级的时间。对随机函数进行细微调整有利于改善这一现象，但仍然很不稳定。总之，不建议在这种情况下使用 Pollard Rho 算法。

4.4 Pohlig-Hellman 算法

Pohlig-Hellman 算法是受群大小为 smooth-number 时的性质启发而产生的。算法首先考虑了 $n = q^k$ 的情况，其中 q 是一个较小的质数，同时考虑将答案 x 在 q 进制下从低到高逐位求出。具体而言，算法的流程如下：

1. 初始化 $x_0 = 0$，计算 $t = g^{q^{k-1}}$。
2. 枚举 $i = 0, 1, 2 \ldots, k - 1$：
   (a) 计算 $v = (b \cdot g^{-x_i})^{q^{k-i-1}}$。
   (b) 求出 d_i 使得 $t^{d_i} = v$ 且 $0 \le d_i < q$。这一步可以通过枚举 d_i 完成。
   (c) 令 $x_{i+1} = x_i + p^i d_i$。
3. 返回 x_k。

算法中的 x_i 即为答案 x 在 q 进制下的低 i 位构成的数。每一轮中，算法通过运算去除已经求出的低位，舍弃其他的高位，而单独求出未知的一位。

至于 n 不是素数幂的情况，Pohlig-Hellman 算法利用了中国剩余定理。先将 n 质因数分解为 $n = \prod_i p_i^{e_i}$，对每个 $p_i^{e_i}$ 分别求解 $x \bmod p_i^{e_i}$，而这只需令 $g' = g^{n/p_i^{e_i}}, b' = b^{n/p_i^{e_i}}$ 然后重新执行上述算法即可。最后再用中国剩余定理合并求出 x。于是，n 为 smooth-number 的情况得到比较完美的解决。

事实上，观察算法中求出 d_i 这一步，我们发现这实际上是在暴力求解一个阶为 q 的离散对数问题，只要将这一步改为其他的离散对数算法（如 BSGS），即可进一步优化算法的时间复杂度。

若 n 的质因子分解为 $n = \prod_i p_i^{e_i}$，则使用 BSGS 优化后，算法的时间复杂度来到 $O(\sum_i e_i(\sqrt{p_i} + \log n))$，在 n 是 smooth-number 或有较多较小的质因子时有非常大的优势（例如，OI 中常见

的 $\mathbb{Z}^{\times}_{998244353}$ 中，群的大小 $998244352 = 2^{23} \times 7 \times 17$）。同时算法不难于实现，可以说是非常实用的算法。

不过相应的，当 n 为质数，或具有一个非常大的质因子时，使用 Pohlig-Hellman 算法基本起不到任何优化效果。

4.5 Index calculus 算法

在这一小节中，我们要解决的问题是 $\mathbb{Z}^{\times}_{p}$ 上的离散对数问题。

4.5.1 算法思想

Index calculus 算法试图把问题全部转化到 smooth-number 上来。对于一个固定的阈值 B，算法通过不断随机选择 l，直至 $b \cdot g^l$ 为 B-smooth 的。这样，就把求解离散对数的问题转化成求解 B 以内质数的离散对数问题。

为了解决后者，算法又通过随机选择 l，使得 g^l 为 B-smooth 的，分解 g^l 从而得到关于 $\log_g p_i$ 的一组方程。通过积累足够多的方程，我们便能通过高斯消元法解出每一个 $\log_g p_i$。

4.5.2 算法流程

1. 选择一个合适的阈值 B。
2. 不断随机选择 l，直至 g^l 为 B-smooth。并找出 g^l 的质因子分解，假设为 $\prod_i p_i^{e_i}$。这样就得到一方程 $\sum_i e_i \log_g p_i = l$，其中 $\log_g p_i$ 为未知数。
3. 反复重复 2.，直至方程足以解出全部未知数，也即全部的 $\log_g p_i (p_i \le B)$。
4. 为了求解 $\log_g b$，不断随机选择 l，直至 $b \cdot g^l$ 为 B-smooth，再分解 $b \cdot g^l$，假设为 $\prod_i q_i^{c_i}$，则得 $\log_g b = (\sum_i c_i \log_g q_i) - l$。

4.5.3 复杂度分析

下记 B 以内的素数个数为 $k = O(B/\ln B)$。

关键在于随机选择 g^l 和 $b \cdot g^l$ 直至 B-smooth 这一步骤所需的随机次数。由于 l 是均匀随机选择的，g^l 和 $b \cdot g^l$ 在 $[1, p-1]$ 中均匀分布。根据参考资料 [3] 的分析，随机至 B-smooth 的概率约为 u^{-u}，其中 $u = \dfrac{\ln p}{\ln B}$。同时，只要随机约 $O(k)$ 个方程，就能解出所有未知数。因而，整个算法的时间复杂度为 $\tilde{O}(k^2 u^u + k^3)$，当 $u \approx \sqrt{\dfrac{2 \ln p}{\ln \ln p}}$，而 $B = L_p[1/2, 2^{-1/2}] = e^{(\sqrt{2}/2 + o(1))\sqrt{\ln p \ln \ln p}}$ 时，算法取到 $L_p[1/2, 3/\sqrt{2}]$ 的时间复杂度。

通过某种方式加速 smooth-number 的判定，可以优化复杂度为 $L_p[1/2, \sqrt{2}]$，本文不再叙述，详见参考资料 [4][5]。

4.5.4 算法优劣

由于复杂度上的巨大优势，Index calculus 算法能求解的离散对数范围远大于其他常见的离散对数算法。例如，其能在未经任何常数优化的情况下，在 6 秒左右求解 200 组左右规模为 3×10^{18} 的离散对数问题[3]，这是其他常见算法难以企及的。

不过，Index calculus 实现时较其他算法略显繁琐，在 OI 中使用可能需要考虑这一代价。

4.6 离散对数问题的一些变体

4.6.1 多次询问的离散对数问题

有时我们会遇到在给定的群中多次求解离散对数的问题，此时可以通过对离散对数算法进行调整来优化运行效率。下面我们认为在 $\mathbb{Z}_p^\times$ 中进行了 T 次离散对数询问。

BSGS 算法中，通过调整阈值 B，可以做到以 $O(B)$ 的复杂度完成预处理，$O(n/B)$ 的复杂度回答单组询问。于是取 $B = \Theta(\sqrt{Tn})$，可以得到 $O(\sqrt{Tn})$ 的总复杂度，优于朴素的 $O(T\sqrt{n})$。

Pollard Rho 算法灵活性较差，很难进行修改，无法进行进一步优化。

Pohlig-Hellman 算法本质上是将原离散对数问题转化成了另外的若干个离散对数问题，在这些另外的离散对数问题中采用相应的方法进行优化即可。例如，使用了 BSGS 以解决这些另外的离散对数问题，则也可以通过调整阈值的方法来优化。

Index calculus 算法也可以通过调整阈值 B 来减少随机产生 smooth-number 的所需步数。总体上而言，将 B 在合理范围内增大可以加速回答单次询问的速度，但是会让预处理变慢。实际应用中，往往针对问题的数据范围，通过调整找到效率较优的 B 值，并使用固定的 B 值，而非一定要使用复杂度分析中确定的 $L_p[1/2, 2^{-1/2}]$。

此外，这里再介绍一种在 OI 中较为实用的离散对数算法。考虑利用带余除法得到等式 $p = qb + r$，其中 $0 \le r < b$。于是有

$$\log_g b \equiv \log_g(-1) + \log_g r - \log_g q \pmod{n}$$

同时我们也有 $p = (q+1)b + (r-b)$，可得

$$\log_g b \equiv \log_g(b-r) - \log_g(q+1) \pmod{n}$$

若 $b > \sqrt{p}$，则有 $q+1 \le \sqrt{p}+1$。于是，倘若事先预处理出了 $\sqrt{p}+1$ 内的离散对数表，则我们可以将求解 $\log_g b$ 的问题递归到求解 $\log_g r$ 或 $\log_g(b-r)$ 的问题，而 $\min(r, b-r) \le b/2$，

[3]在 https://loj.ac/p/6542 上进行测试。

于是可以在一次递归中将问题的规模减半，从而在 $O(\log b)$ 的时间复杂度内计算单个离散对数。

现在，关键问题集中在如何求出 $\sqrt{p}+1$ 内的离散对数表，而通过线性筛法，只需求出 $\sqrt{p}+1$ 内全体素数的离散对数表即可。这可以通过其他离散对数算法解决。倘若使用 BSGS 算法，我们便能在 $O(p^{3/4}/\sqrt{\ln p})$ 的时间复杂度内完成这一步预处理。

在 OI 中常见的模数范围是 10^9 级别，因此算法能在较快的时间内完成预处理，并获得 $O(\log p)$ 单次求解离散对数的优秀复杂度。

这一算法也启发我们不要拘泥于某种特定的算法，而要根据数据范围，灵活的选择合适的方法，或者使用多种方法相结合，从而更好的解决实践中遇到的问题。

4.6.2 底数不为生成元的离散对数问题

目前我们讨论的问题仅限于底数固定为一个给定的生成元 g 的情况。有时我们遇到的离散对数问题形如给定 a,b，求 x 使得 $a^x=b$。

如果已知了群的一个生成元 g，那么这个问题其实并不难处理。我们可以先分别求出 $\log_g a$ 和 $\log_g b$，这样原问题就变成了 $(\log_g a)x \equiv \log_g b \pmod n$，而这是容易使用扩展欧几里得算法解决的。于是问题就转化成了求解 $\log_g a$ 和 $\log_g b$ 的问题，这正是我们熟悉的形式。

这个转化的另一个好处在于，倘若有多组询问具有不同的底数，也能将其转化成相同底数的、多组询问的离散对数问题，从而采用上一小节的方法优化。

在没有生成元的情况下，有的算法仍能求出解或报告无解，如 BSGS 和 Pollard Rho。但除此之外作者尚未发现好的方法解决这种情况下的问题。总之，我们还是尽量将问题转化到生成元上面来。

5 问题意义

5.1 计算理论

AKS 算法标志着素性判定问题存在确定性的多项式时间算法。与之相对的，整数分解问题和离散对数问题至今没有找到多项式时间的传统算法。

然而，也没有证明整数分解问题和离散对数问题属于 NPC 问题。就目前而言，人们认为这两个问题很有可能是 NP-intermediate 的，即既不是 P 的也不是 NPC 的 NP 问题。

这代表着优化这两个问题的时间复杂度很可能需要提出全新的方法，且是具有很高的难度的。

但是，这两个问题在量子计算机上都存在多项式时间的算法。这意味着未来的量子算法可能是解决问题的高效手段。

5.2 与密码学的联系

现代密码学有许多依赖整数分解和离散对数问题的困难性而建立起来的协议和算法，如 RSA 算法，Diffie-Hellman 密钥交换协议，ElGamal 加密算法等等。这意味着这些加密方式的安全性与两个问题的解决效率密切相关，可见两个问题的重要地位。

在离散对数问题上，由于模素数整数乘法群 $\mathbb{Z}_p^\times$ 已经得到了较为深入的研究，存在较快的亚指数时间算法，于是出现了利用精心构造的椭圆曲线群进行加密的算法。在一般群上，目前已知最优的离散对数算法为 BSGS。实践中也常常使用 Pollard Rho 算法求解一般群上的离散对数。

6 总结

本文讨论了三个古老的数论问题：素性判定、整数分解和离散对数问题，同时介绍了数个相关的算法，并简要的提及其与 OI 可能产生的联系。其中有一些算法是 OI 中尚不常见的。计算机科学的知识博大精深，希望本文的内容能为 OI 的发展注入一份新的力量，并吸引更多有能力的读者研究和发展相关的理论。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢雅礼中学屈运华老师多年以来的关心和指导。
感谢父母、朋友对我的支持与鼓励。
感谢褚轩宇等诸位同学在 OI 的学习过程中给我的帮助和陪伴。
感谢罗恺、肖子尧、朱添翼同学为本文提供了宝贵的修改意见。

参考文献

[1] Miller, Gary L. (1976), ”Riemann’s Hypothesis and Tests for Primality”, Journal of Computer and System Sciences, 13 (3): 300−317, doi:10.1145/800116.803773

[2] Rabin, Michael O. (1980), ”Probabilistic algorithm for testing primality”, Journal of Number Theory, 12 (1): 128−138, doi:10.1016/0022-314X(80)90084-0

[3] https://www.csa.iisc.ac.in/~chandan/courses/CNT/notes/lec21.pdf

[4] Carl Pomerance. Fast, rigorous factorization and discrete logarithm algorithms. In Discrete Algorithms and Complexity, pages 119−143. Academic Press, 1987.

[5] Oliver Schirokauer, Damian Weber, and Thomas F. Denny. Discrete logarithms: The effectiveness of the index calculus method. In ANTS, pages 337–361, 1996.

[6] 英文维基的离散对数等相关内容。

浅谈函数的凸性在 OI 中的应用

华东师范大学第二附属中学 郭羽冲

摘 要

利用函数的特殊性质优化算法是 OI 中一种常见且重要的技巧。而函数的凸性是其中一种较为优美且强有力的性质。

本文将围绕着函数的凸性这个主题展开，探讨 OI 中一些利用函数的凸性来优化算法的方法。

1 前置知识

1.1 凸性的定义

函数的凸性分为上凸和下凸两种。

一个定义域为 $[l,r]$ 的函数 $f(x)$ 是上凸的当且仅当

$$\forall x_1,x_2\in[l,r],f\left(\frac{x_1+x_2}{2}\right)\geq\frac{f(x_1)+f(x_2)}{2}$$

一个定义域为 $[l,r]$ 的函数 $f(x)$ 是下凸的当且仅当

$$\forall x_1,x_2\in[l,r],f\left(\frac{x_1+x_2}{2}\right)\leq\frac{f(x_1)+f(x_2)}{2}$$

1.2 凸包与凸壳的定义

一个二维平面上的点集 S（至少包含 3 个不共线的点）的凸包定义为包含所有 S 中的点的最小凸多边形。

与函数的凸性类似，凸壳也分为上凸壳和下凸壳。

设凸包上横坐标最小（如果有多个则取纵坐标最小）的点为 a，横坐标最大（如果有多个则取纵坐标最大）的点为 b，l 为同时经过 a,b 的直线。

S 的上凸壳定义为 S 的凸包在 l 上方的部分。

S 的下凸壳定义为 S 的凸包在 l 下方的部分。

容易发现，上凸壳除去斜率不存在的部分后是一个上凸函数，下凸壳除去斜率不存在的部分后是一个下凸函数。

1.3 (min, +) 卷积与 (max, +) 卷积的定义

设 $f(x)$ 定义域为 $[l_1, r_1]$，$g(x)$ 定义域为 $[l_2, r_2]$，$u(x)$ 为 $f(x), g(x)$ 进行 (min, +) 卷积得到的结果，$v(x)$ 为 $f(x), g(x)$ 进行 (max, +) 卷积得到的结果。

那么有 $u(x), v(x)$ 的定义域为 $[l_1 + l_2, r_1 + r_2]$，且

$$\forall i \in [l_1 + l_2, r_1 + r_2]$$

$$u(i) = \min_{j \in [l_1, r_1], i-j \in [l_2, r_2]} \{f(j) + g(i-j)\}$$
$$v(i) = \max_{j \in [l_1, r_1], i-j \in [l_2, r_2]} \{f(j) + g(i-j)\}$$

1.4 四边形不等式与序列划分问题

一个矩阵 A 满足四边形不等式当且仅当 $\forall i < j \le k < l, A_{i,k} + A_{j,l} \le A_{i,l} + A_{k,j}$。这等价于 $\forall i < j, A_{i,j} + A_{i+1,j+1} \le A_{i+1,j} + A_{i,j+1}$。

定义序列划分问题为：给定 n, m 和一个矩阵 A，求出一个长度为 m 的递增序列 p 满足 $p_0 = 0, p_m = n$ 使得 $\sum_{i=1}^{m} A_{p_{i-1}, p_i}$ 最小。

结论：在序列划分问题中，如果矩阵 A 满足四边形不等式，那么答案关于 m 一定是一个下凸函数。

证明：设 $f(m_0)$ 表示 $m = m_0$ 时的答案，$p(m_0)$ 表示 $m = m_0$ 时的一组最优解对应的 p。

结论等价于 $\forall m_0, d, 2f(m_0) \le f(m_0 - d) + f(m_0 + d)$。

再设 $p_1 = p(m_0 - d), p_2 = p(m_0 + d)$。

根据抽屉原理，一定存在 i 满足

$$i \in [0, m_0 - d), p_{1,i} < p_{2,i+d} < p_{2,i+d+1} \le p_{1,i+1}$$

考虑两组 $m = m_0$ 时的解：$p_{1,0\ldots i} p_{2,i+d+1\ldots m_0+d}$ 以及 $p_{2,0\ldots i+d} p_{1,i+1\ldots m_0-d}$。设这两组解的权值之和为 t。

因为 A 满足四边形不等式，所以可以得到：

$$2f(m_0) \le t$$

$$= f(m_0 - d) + f(m_0 + d) - A_{p_{1,i}, p_{1,i+1}} - A_{p_{2,i+d}, p_{2,i+d+1}} + A_{p_{1,i}, p_{2,i+d+1}} + A_{p_{2,i+d}, p_{1,i+1}}$$

$$\le f(m_0 - d) + f(m_0 + d)$$

结论得证。

2 斜率优化

2.1 简介

斜率优化是一种较为经典的利用凸性优化动态规划的方法。

其主要思想是通过维护二维平面上一些点的上凸壳或下凸壳，并通过在凸壳上求切线来求出最优决策点。

使用斜率优化的关键在于将状态转移转化为求出一个点集中某个斜率的直线的最小截距。

2.2 例题

例 1 给定一个长度为 n 的序列 a 和一个参数 m。定义一个子段 $[l,r]$ 的权值为 $\left(m-\sum_{i=l}^{r}a_i\right)^2$。你需要将序列划分为若干子段使得权值和最小。

$1\le n\le 5\times 10^4, 1\le m,a_i\le 10^7$。

解 设 dp_i 表示前 i 个数的答案，$s_i=\sum_{j=1}^{i}a_j$ 即 a 的前缀和。

有转移式：

$$dp_i=\min_{j=0}^{i-1}\{dp_j+(m-s_i+s_j)^2\}$$

稍作变形可以得到：

$$dp_j+(m-s_i+s_j)^2=(m-s_i)^2+dp_j+s_j^2+2(m-s_i)s_j$$

我们可以对于每个 j 建立一个点 $(s_j,dp_j+s_j^2)$。转移的时候要找出一个当前点集的点使得经过这个点的斜率为 $-2(m-s_i)$ 的直线的截距最小。

容易观察到最优决策点一定在当前点集的下凸壳上。同时由于每次加入的点的横坐标单调不降，且每次询问的斜率也单调不降，所以我们可以用一个队列维护凸壳。

具体来说，队列中的点按照横坐标从小到大排序。加入一个点时从队尾弹出所有不在凸壳中的点。询问切线时从队首不断弹出点，直到当前队首的是切点为止。根据前面的单调性可以得出切点一定在不断向横坐标较大的方向移动，且每次加入的点都是当前横坐标最大的点，因此这个算法的正确性能够保证。

时间复杂度 $O(n)$。

例 2 题意与例 1 相同。

$1\le n\le 5\times 10^4, -10^7\le m,a_i\le 10^7$。

解 动态规划状态的定义以及转移式与例 1 类似，这里不再赘述。

但是现在 a 中出现了负数，因此 s 不再是单调不降的了，因此我们不能直接使用例 1 中提到的方法。

此时我们需要使用平衡树维护凸壳，支持加入一个点，查询某个斜率的直线在凸壳上的切点。

维护凸壳时，平衡树中的点按照横坐标从小到大排序。

加入一个点 (x,y) 时可以先在平衡树上找到前驱后继，然后分别在左右两边删去所有不在凸壳上的点。这一部分的时间复杂度是 $O(n\log n)$。

查询时在平衡树上二分找到切线即可。这一部分时间复杂度是 $O(n\log n)$。

因此这个算法的总时间复杂度为 $O(n\log n)$。

2.3 小结

例 1 是一个斜率优化中的一个较为特殊的情况，可以做到 $O(n)$。

在例 2 这种更一般的情况中需要在凸壳上二分，以及用高级数据结构维护凸壳，一般可以做到 $O(n\log n)$。

3 wqs 二分

3.1 简介

wqs 二分是一种用途广泛的利用凸性的优化方法，可以与动态规划以及其它许多算法同时使用。

这种方法所适用的情况是答案关于某一个参数是一个凸函数，并且我们只需要求出这个凸函数上的某个点而不是求出整个函数。

其主要思想是二分一个斜率 k，用一条斜率为 k 的直线来切这个凸函数，直到找到一个合适的斜率 k 使得凸函数上斜率为 k 的切线经过我们需要求的点。

使用 wqs 二分优化动态规划时一般能够减少一维状态，从而达到优化复杂度的效果。

函数凸性的证明可以直接根据函数定义推导，也可以借助于费用流模型。在费用流模型中，最小费用关于流量是下凸的，最大费用关于流量是上凸的。

3.2 例题

例 3 给定一个长度为 n 的序列 a，要求选出恰好 m 个两两不相邻的数使得和最大。

$1 \le m \le n \le 10^5, -10^9 \le a_i \le 10^9$。

解 先证明答案关于 m 的凸性。考虑借助于费用流模型。

将点按照下标的奇偶性分类，两类点分别放在二分图的两侧，然后将问题转化为二分图最大权匹配。

连边方式如下：

- $\forall i \in [1, n+1], i \equiv 0 \pmod 2, S \to i$ 容量为 1 费用为 0。
- $\forall i \in [1, n+1], i \equiv 1 \pmod 2, i \to T$ 容量为 1 费用为 0。
- $\forall i \in [1, n], i \equiv 0 \pmod 2, i \to i+1$ 容量为 $+\infty$ 费用为 a_i。
- $\forall i \in [1, n], i \equiv 1 \pmod 2, i+1 \to i$ 容量为 $+\infty$ 费用为 a_i。

下图是一个 $n = 4$ 的例子。

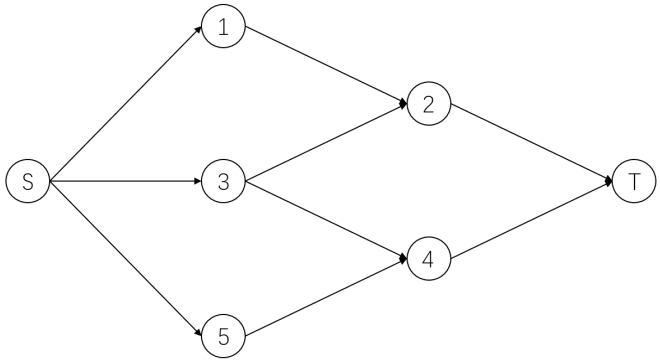


图 1:

这个图中流量恰好为 m 时的最大费用即为答案，因此答案关于 m 是上凸的。因此我们可以二分一个斜率 k，只需要考虑如何求出斜率为 k 的直线在凸函数上的切点。这个切点即为使得斜率为 k 的直线截距最大的点。

我们可以把问题转化为：给定一个序列，要求选出若干个两两不相邻的数使得和最大。新问题中的序列中的第 i 个数为 $a_i - k$。

设 dp_i 表示前 i 个数的答案。

有转移式：

$$dp_i = \max\{dp_{i-1}, dp_{i-2} + a_i\}$$

直接按照上面的式子转移的时间复杂度为 $O(n)$。

但是只求出最大截距是不够的，我们还需要知道切点的横坐标。如果由于出现三点共线而产生了多个切点，我们可以钦定取其中横坐标最大的一个。

具体来说，在上面的转移过程中，我们对每个 i 记录 z_i 表示前 i 个数中取到最优解时最多选取了多少个数。而选取的子段个数实际上就对应了点的横坐标。最终得到的切点坐标即为 $(z_n, dp_n + kz_n)$。z_i 可以在计算 dp_i 的过程中同时计算出来。

这个算法的总时间复杂度为 $O(n \log V)$，其中 V 是 wqs 二分的值域。

例 4 给定一个长度为 n 的序列 a，要求选出恰好 m 个两两无公共点的子段 $[l_i, r_i]$ 使得 $\sum_{i=1}^{m} \sum_{j=l_i}^{r_i} a_j$ 最大。

$1 \le m \le n \le 10^5, -10^9 \le a_i \le 10^9$。

解 先证明答案关于 m 的凸性。与例 3 类似，依然考虑费用流模型。

本题的费用流模型较为简单，下面直接给出连边方式：

- $\forall i \in [1, n+1], S \to i$ 容量为 $+\infty$ 费用为 0。
- $\forall i \in [1, n+1], i \to T$ 容量为 $+\infty$ 费用为 0。
- $\forall i \in [1, n], i \to i+1$ 容量为 1 费用为 a_i。

下图是一个 $n = 5$ 的例子。

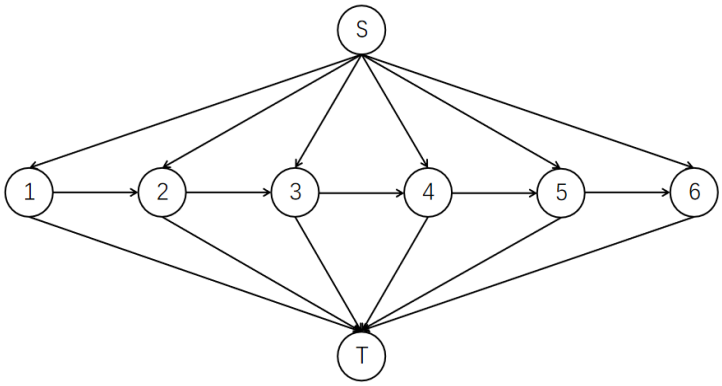


图 2:

这个图中流量恰好为 m 时的最大费用即为答案，因此答案关于 m 是上凸的。

二分一个斜率 k 之后，我们可以把问题转化为：给定一个序列，要求选出若干个不相交的子段，一个子段 $[l, r]$ 的权值为 $-k + \sum_{i=l}^{r} a_i$，求最大总权值。

设 dp_i 表示前 i 个数的答案。

有转移式：

$$dp_i = \max\{dp_{i-1}, \max_{j=0}^{i-1} dp_j - k + s_i - s_j\}$$

通过记录前缀最小值可以做到 $O(n)$。

与例 3 类似，我们对每个 i 记录 z_i 表示前 i 个数中取到最优解时最多选取了多少个子段，它也可以在计算 dp_i 的过程中同时计算出来。

总时间复杂度为 $O(n \log V)$，其中 V 是 wqs 二分的值域。

3.3 小结

例 3 和例 4 都是 wqs 二分最基础的应用方式，将一个 $O(n^2)$ 的动态规划优化到 $O(n \log V)$。在第 5 节中还会提到 wqs 二分与其它凸性优化方法结合起来的一些应用。

4 维护凸函数

4.1 简介

一些在一般函数中难以快速完成的操作在凸函数中存在很高效的算法。例如两个函数的 $(\min, +)$ 卷积，在一般函数中只能做到 $O(n^2)$，但是在下凸函数中可以通过对所有函数中的每一段斜率按照从小到大的顺序进行归并做到 $O(n)$。

可以用高级数据结构维护凸函数中每一段的斜率，需要能支持快速的修改以及查询，有时可能需要支持区间操作。

4.2 例题

例 5 给定两个长度为 n 的序列 a, b。有一个恰好有 m 个 1 的 01 序列 S，设 $c_i = S_i a_i + (1 - S_i) b_i$。你需要求出一个 S 使得 c 的最大子段和（可以为空）最小。

$1 \le n \le 2 \times 10^5, 0 \le m \le n, -10^9 \le a_i, b_i \le 10^9$。

解 一个序列的最大子段和可以用如下过程求出：

- 维护一个变量 now，初始时 $now = 0$。
- 依次考虑序列中的每个数，设当前考虑的是 x，那么将 now 变为 $\max\{now + x, 0\}$。
- 所有时刻中 now 的最大值即为序列的最大子段和。

回到原题，我们可以先二分答案。设当前需要判断答案是否能不超过 mid，这相当于要求任何一个时刻 now 都不超过 mid。设 $dp_{i,j}$ 表示当前考虑前 i 个数，$S_{1...i}$ 中有 j 个 1 时 now 的最小值。

有转移式：

$$dp_{i,j} = \max\{\min\{dp_{i-1,j-1} + a_i, dp_{i-1,j} + b_i\}, 0\}$$

稍作变形可以得到：

$$dp_{i,j} = \max\{\min\{dp_{i-1,j-1} + a_i - b_i, dp_{i-1,j}\} + b_i, 0\}$$

将 dp_i 看作一个关于 j 的函数，我们可以归纳地认为它是下凸的，并考虑这个函数在转移时的变化情况。

将转移的过程分为三步：

- $dp_{i,j} \leftarrow \min\{dp_{i-1,j-1} + a_i - b_i, dp_{i-1,j}\}$
- $dp_{i,j} \leftarrow dp_{i,j} + b_i$。
- $dp_{i,j} \leftarrow \max\{dp_{i,j}, 0\}$。

第一步：在函数中插入一条横向跨度为 1 单位长度且斜率为 $a_i - b_i$ 的线段。

第二步：对整个下凸函数全局加 b_i。

第三步：将下凸函数中小于 0 的点全部改为 0。

需要注意的是，如果某一个状态的值超过了 x，那么需要将其删除，因此有第四步：将下凸函数中大于 x 的部分全部删去。

可以发现这三个操作并不会破坏函数的下凸性，因此归纳成立。

考虑维护函数中每一段横向跨度为 1 单位长度的线段的斜率。

以下凸函数中斜率为 0 的那一段作为分界，左边用 set_1 维护，右边用 set_2 维护。考虑两步操作在这种数据结构上的实现方法。

第一步：如果 $a_i - b_i < 0$，那么往 set_1 中加入一个 $a_i - b_i$，否则往 set_2 中加入一个 $a_i - b_i$。

第二步：维护一个全局加法标记即可，注意这里 set_1, set_2 中的数并不改变。

第三步：在两边不断从内往外弹出线段，直到当前顶部的线段与 x 轴有交点，对这条线段进行特殊处理。

第四步：在两边不断从外往内弹出线段，直到当前没有超过 mid 的点。

以上操作都可以在 $O(n \log n)$ 的时间复杂度内完成。因此本题总时间复杂度为 $O(n \log n \log V)$，其中 V 是二分答案的值域。

例 6 给定一棵 n 个点的有边权的树，要求选出恰好 m 个互不相同的点 $a_{1\ldots m}$ 使得 $\sum_{i=1}^{m} dis(a_i, a_{i+1})$ 最大。其中 $a_{m+1} = a_1$，$dis(u, v)$ 表示树上 u, v 两点间唯一简单路径的边权和。

$1 \le m \le n \le 2 \times 10^5, 1 \le w \le 10^9$。

解 首先考虑 $n = m$ 时的做法。对于一条边 (u, v, w)，设删掉这条边之后形成的两个连通块大小为 $size_1, size_2$，那么这条边最多被经过 $2\min\{size_1, size_2\}$ 次。以树的重心 G 为根，根

的每个儿子的子树大小都不超过 $\frac{n}{2}$，因此一定存在一种将 n 个点排成一个环的方案使得相邻两个点不在根的同一个儿子的子树中。而这种方案恰好能取到之前分析出的上界。

考虑给定 m 个点，如何将它们填入 a 中使得答案最大。做法与 $n = m$ 的情况类似。对于每一条边 (u, v, w)，设删掉这条边之后形成的两个连通块中分别有 cnt_1, cnt_2 个给定的点，那么这条边最多被经过 $2\min\{cnt_1, cnt_2\}$ 次。沿用上面的分析方法同样可以得到这个上界能够取到。

根据上面的分析可以设计一个动态规划：设 $dp_{u,i}$ 表示考虑 u 的子树，子树内选了 i 个点的答案，w_u 表示 u 到父亲的边权。

树形动态规划时依次加入 u 的每一个儿子 v，设 dp' 为加入 v 后 dp 的值，有转移方程：

$$dp'_{u,i} = \max_{j=0}^{i}\{dp_{u,i-j} + dp_{v,j}\}$$

在加入 u 的所有儿子 v 之后，如果 u 不是根，设 dp' 表示最后 dp 的值，那么又有：

$$dp'_{u,i} = \max\{dp_{u,i}, dp_{u,i-1}\} + 2\min\{i, m-i\}w_u$$

将 dp_u 看作一个关于 i 的函数，我们可以归纳地认为它是上凸的，并考虑这个函数在转移时的变化情况。

将转移的过程分为三步：

- $dp_u \leftarrow \prod_{v \in son_u} dp_v$，其中 son_u 表示 u 的所有儿子构成的集合，函数的乘法定义为 $(\max, +)$ 卷积。
- $dp_{u,i} \leftarrow \max\{dp_{u,i}, dp_{u,i-1}\}$。
- $dp_{u,i} \leftarrow dp_{u,i} + 2\min\{i, m-i\}w_u$。

第一步：将所有的函数中的每一段斜率按照从大到小的顺序排序就可以得到 $(\max, +)$ 卷积的结果。

第二步：在函数中插入一条横向跨度为 1 单位长度且斜率为 0 的线段。

第三步：相当于将一段前缀的斜率增加 w_u，并将一段后缀的斜率减少 w_u。

可以发现这三个操作并不会破坏函数的上凸性，因此归纳成立。

考虑维护函数中每一段横向跨度为 1 单位长度的线段的斜率。

以 $\frac{m}{2}$ 作为分界，左边用一个小根堆 $heap_1$ 维护，右边用一个大根堆 $heap_2$ 维护，同时再维护一个当前答案 ans。考虑两步操作在这种数据结构上的实现方法。

第一步：用启发式合并将所有儿子的 $heap_1$ 和 $heap_2$ 合并起来。

第二步：先判断斜率为 0 的线段与 $\frac{m}{2}$ 的位置关系，然后往对应的堆中加入一个 0。

第三步：将 $heap_1$ 全局加 w_u，$heap_2$ 全局减 w_u。对于两个堆分别维护一个全局加法标记即可。

以上操作都可以在 $O(n\log^2 n)$ 的时间复杂度内完成。

例 7 给定二维平面上的 n 个点，第 i 个点为 (x_i, y_i)，你需要从 $(0,0)$ 开始，每次往右或往上走 1 单位长度，形成一条无限长的路径 P。设 $dis((x,y),P)$ 表示 P 上的点与 (x,y) 的切比雪夫距离的最小值。其中 (x_1,y_1) 和 (x_2,y_2) 的切比雪夫距离为 $\max\{|x_1-x_2|,|y_1-y_2|\}$ 求 $\min\limits_{P}\{\sum\limits_{i=1}^{n} dis((x_i,y_i),P)\}$。

$1 \le n \le 8\times 10^5, 0 \le x_i \le y_i \le 10^9$。

解 结论：P 上与 (x,y) 的曼哈顿距离的最小的点一定是 P 上横纵坐标之和与 $x+y$ 相等的点 (x_0,y_0)。

证明：因为 $x+y=x_0+y_0$，所以 $|x-x_0|=|y-y_0|$ 且 $x\ge x_0$ 和 $y\ge y_0$ 至少有一个成立。不妨设 $x\ge x_0$。对于所有 (x_1,y_1) 满足 (x_1,y_1) 在 P 上且 $x+y>x_1+y_1$，一定有 $x_0\ge x_1, y_0\ge y_1$，进一步可以推出 $|x-x_1|=x-x_1\ge x-x_0=|x-x_0|$，即 (x,y) 和 (x_0,y_0) 的切比雪夫距离不大于 (x,y) 和 (x_1,y_1) 的切比雪夫距离。对于 $x+y<x_1+y_1$ 也可以类似地证明。结论得证。

根据上述结论，我们可以考虑根据点的横纵坐标之和从小到大进行动态规划。对于一个给定的点 (x_i,y_i)，只在横纵坐标之和为 x_i+y_i 时考虑它的贡献。

设 $dp_{i,j}$ 表示当前走到 $x+y=i$ 这条直线上，当前走到的点的横坐标为 j 的答案，$w_{i,j}=\sum\limits_{x_k+y_k=i}|j-x_k|$。

有转移式：

$$dp_{i,j}=\min\{dp_{i-1,j},dp_{i-1,j-1}\}+w_{i,j}$$

考虑将 dp_i 看作一个关于 j 的函数，我们可以归纳地认为它是下凸的，并考虑这个函数在转移时的变化情况。

将转移的过程分为两步：

- $dp_{i,j}\leftarrow \min\{dp_{i-1,j},dp_{i-1,j-1}\}$。
- $dp_{i,j}\leftarrow dp_{i,j}+w_{i,j}$。

第一步：在函数中插入一条横向跨度为 1 单位长度且斜率为 0 的线段。

第二步：考虑所有满足 $x_k+y_k=i$ 的 k，执行 $dp_{i,j}\leftarrow dp_{i,j}+|j-x_k|$。这相当于将一段前缀的斜率减少 1，再将一段后缀的斜率增加 1。

可以发现这两个操作并不会破坏函数的下凸性，因此归纳成立。

考虑维护所有斜率变化的位置构成的可重集。如果凸函数在横坐标 x 处右边的斜率相比左边的斜率增加了 Δ，那么集合中就有 Δ 个 x。

以下凸函数中斜率为 0 的那一段作为分界，左边用一个大根堆 $heap_1$ 维护，右边用一个小根堆 $heap_2$ 维护，同时再维护一个当前答案 ans。考虑两步操作在这种数据结构上的实现方法。

第一步：将 $heap_2$ 中的所有数增加 1，ans 不变。维护一个全局加法标记即可。

第二步：设 $heap_1$ 中的最大值为 l。$heap_2$ 中的最小值为 r。按照 x_k 的大小分类讨论：

- 如果 $x_k \in [l, r]$，那么往 $heap_1$ 和 $heap_2$ 中各加入一个 x_k，ans 不变。
- 如果 $x_k < l$，那么先往 $heap_1$ 中加入两个 x_k，然后将 $heap_1$ 中的最大值弹出，加入到 $heap_2$ 中，ans 增加 $l - x_k$。
- 如果 $x_k > r$，那么先往 $heap_2$ 中加入两个 x_k，然后将 $heap_2$ 中的最小值弹出，加入到 $heap_1$ 中，ans 增加 $x_k - r$。

以上操作都可以在 $O(n \log n)$ 的时间复杂度内完成。

4.3 小结

本节中的三道例题中提到了几种比较巧妙的维护凸函数的方法。事实上，三道题中的凸函数都可以使用平衡树在相同时间复杂度内维护。这种方法的代码实现较为复杂、时间常数较大，但是应用范围相比上面提到的几种方法更为广泛。因此，如果题目中需要支持一些难以简单实现的操作时，高级数据结构也是一种很有效的方法。

5 凸性优化的综合应用

5.1 简介

上述的三种凸性优化方法在一些情况下可以结合起来使用来解决一些更为困难的题目，有时还需要使用其它的数据结构方法。本节主要通过几道例题来介绍几种结合应用的方法。

5.2 例题

例 8 给定一个长度为 n 的序列 a，你需要求出一个序列 b 满足 $\sum_{i=1}^{n} |a_i - b_i|$ 最小，且 b 中最多有 m 种不同的数。

$1 \le m \le n \le 10^5, 1 \le a_i \le 10^9$。

解 不妨设序列 a 单调不降。

假设已经确定 b 中的 m 种不同的值，设这 m 个数从小到大排序后为 $c_{1\ldots m}$。对于每个 a_i 找到 $|a_i - c_j|$ 最小的一个 c_j 作为 b_i。

对于每一个 a_i 考虑选择 b_i 的情况：

- 如果 $a_i < c_1$，那么 $b_i = c_1$。
- 如果 $a_i \geq c_m$，那么 $b_i = c_m$。
- 如果 $a_i \in [c_1, c_m)$，那么可以找到唯一的 $j \in [1, m)$ 满足 $a_i \in [c_j, c_{j+1})$。此时如果 $a_i \leq \frac{c_j + c_{j+1}}{2}$，那么 $b_i = c_j$，否则 $b_i = c_{j+1}$。

根据上面的分析可以发现，b 单调不降，因此 b 中相等的值一定是连续的。

假设已知 $b_{l\ldots r}$ 两两相等，当 $\sum_{i=l}^{r} |a_i - b_i|$ 最小时 b_l 的取值范围为 $\left[a_{\lfloor \frac{l+r}{2} \rfloor}, a_{\lceil \frac{l+r}{2} \rceil}\right]$ 中的任意一个数。

设 $dp_{i,j}$ 表示前 i 个数划分为 j 段的答案，$s_i = \sum_{j=1}^{i} a_j$ 即 a 的前缀和，$w_{i,j}$ 表示 $b_{i+1\ldots j}$ 两两相等时 $\sum_{k=i+1}^{j} |a_k - b_k|$ 的最小值。

有转移式：

$$dp_{i,j} = \min_{k=0}^{i-1} \{dp_{k,j-1} + w_{k,i}\}$$

我们可以将这个问题抽象为一个序列划分问题（见 1.4 节），w 即为序列划分问题中的矩阵 A。

结论：矩阵 w 满足四边形不等式。即 $\forall i < j, w_{i,j} + w_{i+1,j+1} \leq w_{i+1,j} + w_{i,j+1}$。

证明：按照 $i + j$ 的奇偶性分类讨论：

- 如果 $i + j$ 为奇数，设 $t = \frac{i + j + 1}{2}$，那么可以得到：

$$w_{i,j} = s_i + s_j - s_t - s_{t-1}, w_{i+1,j+1} = s_{i+1} + s_{j+1} - s_{t+1} - s_t$$

$$w_{i+1,j} = s_{i+1} + s_j - 2s_t, w_{i,j+1} = s_i + s_{j+1} - 2s_t$$

进一步得到：

$$w_{i,j} + w_{i+1,j+1} = s_i + s_{i+1} + s_j + s_{j+1} - 2s_t - s_{t-1} - s_{t+1} \leq s_i + s_{i+1} + s_j + s_{j+1} - 4s_t = w_{i+1,j} + w_{i,j+1}$$

- 如果 $i + j$ 为偶数，设 $t = \frac{i + j}{2}$，那么可以得到

$$w_{i,j} = s_i + s_j - 2s_t, w_{i+1,j+1} = s_{i+1} + s_{j+1} - 2s_{t+1}$$

$$w_{i+1,j} = s_{i+1} + s_j - s_t - s_{t+1}, w_{i,j+1} = s_i + s_{j+1} - s_t - s_{t+1}$$

进一步得到：

$$w_{i,j} + w_{i+1,j+1} = s_i + s_{i+1} + s_j + s_{j+1} - 2s_t - 2s_{t+1} = w_{i+1,j} + w_{i,j+1}$$

根据 1.4 节中的结论，答案关于 m 是下凸的。按照 wqs 二分的方法，我们二分一个斜率 k 然后求出斜率为 k 的直线在函数上的切点。

设 dp_i 表示前 i 个数划分为若干段的答案。

有转移式：

$$dp_i = \min_{j=0}^{i-1}\{dp_j + w_{j,i} - k\}$$

因为 w 满足四边形不等式，所以可以利用决策单调性优化动态规划，进行一轮动态规划的时间复杂度为 $O(n)$ 或者 $O(n\log n)$。总时间复杂度为 $O(n\log V)$ 或者 $O(n\log n\log V)$。其中 V 是 wqs 二分的值域。

实际上还有更简单的方法使得一轮动态规划的时间复杂度为 $O(n)$。

其主要思想是不直接算出每个段的最优权值，而是将这一部分也放入动态规划中进行处理。

具体来说，每个段中我们会钦定其中恰好一个元素 x 出现在 b 中，因此我们把这个段按 x 划分为左右两部分分别计算贡献。这样做的好处是在 x 同侧的元素与 x 的大小关系一致，可以把绝对值拆开。

设 $dp_{i,0/1}$ 表示前 i 个数的答案，第二维是 0 的时候要求 i 是划分出的某个段的结尾，第二维是 1 的时候要求 a_i 在 b 中出现过。

有转移式：

$$dp_{i,0} = \min_{j=1}^{i}\{dp_{j,1} + s_i - s_j - (i-j)a_j\}$$

$$dp_{i,1} = \min_{j=0}^{i-1}\{dp_{j,0} + (i-j)a_j - s_i + s_j\}$$

答案即为 $dp_{n,0}$。

可以利用斜率优化做到 $O(n)$ 的时间复杂度。总时间复杂度为 $O(n\log V)$。

例 9 给定一棵 n 个点的带边权的树，对于每个 $i \in [1,n)$ 求出这棵树中大小恰好为 i 的最大权匹配。

$1 \le n \le 2\times 10^5, -10^9 \le w \le 10^9$。

解 本题中树退化为一条链，且只需求出一个 i 的答案的情况与例 3 是一致的。

不妨设树的根是 1。

设 $dp_{u,i,0/1}$ 表示考虑 u 的子树以及 u 到父亲的边，共选了 i 条边的答案，第三维是 0 的时候要求 u 到父亲的边不能被选择，fa_u 表示 u 的父亲，dep_u 表示 u 的深度，即 u 到根的路径长度，$size_u$ 表示以 u 为根的子树的大小，w_u 表示 u 到父亲的边的边权。

树形动态规划时依次加入 u 的每一个儿子 v，设 dp' 为加入 v 后 dp 的值，有转移方程：

$$dp'_{u,i,0} = \max_{j=0}^{i}\{dp_{u,i-j,0} + dp_{v,j,1}\}$$

$$dp'_{u,i,1} = \max_{j=0}^{i}\{dp_{u,i-j,1} + dp_{v,j,0}\}$$

在加入 u 的所有儿子 v 之后，如果 u 不是根，设 dp' 表示最后 dp 的值，那么又有：

$$dp'_{u,i,1} = \max\{dp_{u,i,0}, dp_{u,i-1,1} + w_u\}$$

直接进行转移的时间复杂度是 $O(n^2)$。

将 $dp_{u,i,0/1}$ 看作一个关于 i 的函数。与之前直接维护凸函数的题不同的是，本题中我们无法直接通过转移的过程归纳证明函数的凸性。但我们可以借助费用流模型来证明它关于 i 的上凸性。

与例 3 类似，将点按照深度的奇偶性分类。

连边方式如下：

- $\forall u \in [1,n], dep_u \equiv 0 \pmod 2, S \to i$ 容量为 1 费用为 0。
- $\forall u \in [1,n], dep_u \equiv 1 \pmod 2, i \to T$ 容量为 1 费用为 0。
- $\forall u \in (1,n], dep_u \equiv 0 \pmod 2, u \to fa_u$ 容量为 $+\infty$ 费用为 w_u。
- $\forall i \in (1,n], dep_u \equiv 1 \pmod 2, fa_u \to u$ 容量为 $+\infty$ 费用为 w_u。

这个图中流量恰好为 i 时的最大费用即为 $dp_{u,i,1}$，因此 $dp_{u,i,1}$ 关于 i 是上凸的，进一步可以得到 $dp_{u,i,0}$ 关于 i 也是上凸的。

但如果直接利用上凸性优化转移，对于每个 u 合并的时间复杂度为 $O(size_u)$，总时间复杂度达到了 $O(\sum_{u=1}^{n} size_u) = O(n^2)$，并没有得到优化。

先考虑弱化版：树退化为一条链。

对链进行分治，设当前处理的区间为 $[l,r]$，我们希望求出一个 2×2 的矩阵，两维 0/1 分别表示 l 到父亲的边和 r 到父亲的边是否被选择。矩阵中的每个元素是一个横向跨度为 $O(r-l)$ 的函数，其中横坐标为 i 时所对应的纵坐标表示从 $[l,r]$ 中恰好选择一个大小为 i 的匹配，并且满足两维 0/1 的限制时的答案。

先递归 $[l,mid]$ 和 $(mid,r]$ 两部分分别计算，然后利用凸性可以在 $O(r-l)$ 的时间内将两部分的答案矩阵合并起来得到 $[l,r]$ 的答案矩阵。时间复杂度 $O(n \log n)$。

对于一般的情况，我们对树进行重链剖分。称 u 是轻的当且仅当 $u = 1$ 或 u 是 fa_u 的轻儿子，反之则称 u 是重的。

结论：所有轻的点的 $size$ 之和不超过 $n \log n$。

证明：对于 u, v，如果 v 是 u 的轻儿子，那么一定有 $size_u \geq 2size_v$。考虑一个点不断往父亲移动直到移动到根的过程，每经过一条轻边，当前点的 $size$ 就至少 $\times 2$，因此对于任意一个点，它到根的路径上的轻边条数都不超过 $\log n$。而所有轻的点的 $size$ 之和等于所有点到根的路径上的轻边条数之和，这个值一定不超过 $n \log n$。结论得证。

设 hv_u 表示 u 的重儿子。

我们对于每个轻的 u 计算答案，对于每个重的 u 计算只考虑它的所有轻儿子的答案。

对于每个 u，我们分治地将所有轻儿子 v 的信息合并起来。根据之前的结论，这一部分的时间复杂度为 $O(n\log^2 n)$。

对于每个轻的 u，我们在以它为顶点上的重链使用弱化版中提到的方法将重链上所有点的信息合并起来就可以得到 u 的答案。因为每条重链只会被在其顶点处访问一次，所以根据之前的结论也可以得到这一部分的时间复杂度为 $O(n\log^2 n)$。

总时间复杂度为 $O(n\log^2 n)$。使用全局平衡二叉树相关技巧可以优化到 $O(n\log n)$，这里不再详细展开。

例 10 给定一个长度为 n 的序列 a，进行 m 次询问，每次给出 l,r,c 要求从 $[l,r]$ 中选出恰好 c 个两两无公共点的子段 $[l_i,r_i]$ 使得 $\sum_{i=1}^{c}\sum_{j=l_i}^{r_i} a_j$ 最大。

$1\le n,m\le 3.5\times 10^4, -10^9\le a_i\le 10^9, 1\le l\le r\le n, 1\le c\le r-l+1$。

解 本题 $m=1$ 的情况与例 4 是一致的。在例 4 中我们已经利用费用流模型分析出了答案关于 c 的上凸性，我们需要继续沿用这一条关键性质。

建立一棵线段树，在线段树上的每一个节点 $[l,r]$ 中维护一个 2×2 的矩阵，两维 0/1 分别表示 l 和 r 是否被选择。矩阵中的每个元素是一个横向跨度为 $O(r-l)$ 的函数，其中横坐标为 i 时所对应的纵坐标表示从 $[l,r]$ 中恰好选择 i 个两两无公共点的子段，并且满足两维 0/1 的限制时的答案。根据之前的性质可以得到矩阵中每个元素都是上凸函数。

对于线段树上的每个节点 $[l,r]$，可以利用凸性可以在 $O(r-l)$ 的时间内将两个儿子的答案矩阵合并起来得到 $[l,r]$ 的答案矩阵。这一部分的时间复杂度为 $O(n\log n)$。

对于每次询问，先将询问区间 $[l,r]$ 分裂成线段树上的 $O(\log n)$ 个节点。但我们显然不能直接像预处理时一样直接合并每个节点的矩阵，否则时间复杂度会达到 $O(nm\log n)$。

考虑使用 wqs 二分，设当前二分的斜率为 k。在每个分裂出的节点中通过二分求出矩阵中每个元素上斜率为 k 的切线，此时矩阵里的元素不再是一个函数，而是一个整数。最后将这些矩阵合并起来即可。时间复杂度降低到 $O(n\log n+m\log^2 n\log V)$，其中 V 是 wqs 二分的值域。

将对每个询问单独二分改为整体二分就可以进一步降低复杂度。具体来说，对于当前需要考虑的 t 个询问，我们只考虑它们在线段树上分裂出的 $O(t\log n)$ 个节点，在这些节点上暴力找出切点。递归处理左侧部分时，只需要保留凸函数上在当前切点右侧的部分。

时间复杂度分析：将整体二分的过程看作一棵二叉树，对于线段树上的每一个节点 $[l,r]$，它在二叉树的同一层节点上产生的时间代价不超过 $O(r-l)$，而线段树上所有节点的长度总和不超过 $O(n\log n)$。因此时间复杂度为 $O((n+m)\log n\log V)$。

5.3 小结

本节中的三道例题中提到了凸性优化方法的几种综合应用，以及一些与其它算法或数据结构结合使用的例子。可见凸性优化用途广泛，是一种非常有效的优化算法的方法。

6 总结

本文介绍了几种凸性优化方法，并通过十道例题较为具体地展现了这些凸性优化方法是如何应用在 OI 问题中的。相信读者们已经体会到了凸性优化方法在 OI 中的广泛用途和强大效力，它们有时可以极大地优化算法的时空复杂度，提升程序运行的效率。

本文中提到的只是凸性优化的冰山一角，希望本文可以起到一个抛砖引玉的作用，激发读者们对凸性优化这一领域继续进行深究的兴趣和热情。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢教练金靖老师、吴申广老师对我的指导。

感谢父母对我的培育和教诲。

感谢所有帮助我的同学、老师。