

数据结果的提炼与压缩

摘要

时间复杂度和空间复杂度是衡量一个数据结构的重要指标，往往，简单存储结构和小存储规模的数据结构会带来较低的时空复杂度。因此，在程序设计中就需要仔细分析问题，化简存储结构，减少存储规模。

关键字

化简存储结构、减少存储规模、“提炼”、“压”、“缩”、DFS 序、BFS 序

引言

作为程序设计的一部分，数据结构在现在的信息学竞赛中起着越来越大的作用。一方面，一个好的数据结构是高效实现算法的基础，例如当算法设计动态序关系时，通过平衡排序二叉树维护序关系就要比普通线性结构高效的多；另一方面，的信息学竞赛中已经涌现出一大批“赤裸裸”的数据结构题，即题目所需要的，就是设计一个能维护特定数据关系，能高效完成特定操作的数据结构，例如 NOI2007 necklace、CEOI2007 necklace 都是这样的问题。

衡量一个程序的优劣，有四个主要的参考度量：时间复杂度，空间复杂度，求解精确度，编程复杂度。时间复杂度衡量程序运行的时间，往往用一个相对输入规模的阶表示，例如冒泡排序的时间复杂度是 $O(n^2)$ ；空间复杂度用来衡量程序所需的除读入外的额外内存空间，往往也用一个相对输入规模的阶表示，例如 KMP 算法的空间复杂度是 $O(n)$ ；求解精确度衡量程序输出解的质量，往往用与最优解之间的相对或绝对误差来描述，现在的信息学竞赛中大多数问题不允许有误差，而另一些问题将求解精确度作为评分尺度之一；编程复杂度衡量正确编程和程序调试的困难程度。

而作为对数据结构优劣的评判，（在编程复杂度不过分高的前提下）一个数据结构的时空复杂度就显得最为重要。本文将主要讨论的通过“化繁为简”的手段优化数据结构的时空复杂度。

“化繁为简”作为一个可以广泛应用的方法，它能给程序设计带来以下两方面的优势。

一方面，直观地看，在规模小、结构简单的数据上进行操作，时空耗费比起规模大、结构复杂的数据在操作规模上“天生”就有着无与伦比的优势。例如，图结构的存储无论是采取邻接矩阵还是邻接表，空间都是 $O(m+n)$ ，只要不是稀疏图，就会占用平方阶的空间，同时，所有基本操作的时间复杂度也在平方阶之上；但对于树结构而言，常用的左儿子右兄弟表示法只需要 $O(n)$ 的空间复杂度，像遍历这样的基本操作，时间复杂度也是 $O(n)$ 。

另一方面，一个简单的数据结构可以有更多的处理手段，也适应于更多的算法。例如，在图结构上，一般只能进行连通性判断，流算法（事实上，流算法

实现中利用到了树结构)等;而在树结构上,就可以实施树型动态规划;更进一步,对于线性结构而言,DP (Dynamic Programming, 动态规划)的形式更加灵活,而且还可以使用线段树,树状数组等工具;另外,如果线性结构辅助序关系,那么各种平衡BST (Binary Search Tree, 二叉搜索树)也能有用武之地。

本文将提出三种常见的化繁为简的手段

- (i) 提炼: 忽略无效信息, 减少存储规模
- (ii) 压: 调整存储方式, 化简存储结构
- (iii) 缩: 合并重复信息, 减少存储规模

1. 二维结构的化简

二维结构主要分两种情况，其一：两维对称，即矩阵的情况；其二：两维不对称，即一个线性表，它的每个元素都是一维结构，串数组是常见的情况。下面我们分别就这两种情况举个例子。

问题一：ural 1568 Train car sorting

问题描述：对于一个序列 $\{a_n\}$ ，定义一种操作，将 a 变成 b ，使得：

$b_1 = a_{x_1}, b_2 = a_{x_2} \dots b_s = a_{x_s}, b_{s+1} = a_{y_1}, b_{s+2} = a_{y_2} \dots b_{s+t} = a_{y_t}$ 。其中 $s+t=n$, $x_1 < x_2 < \dots < x_s, y_1 < y_2 < \dots < y_t, \{x_1, x_2 \dots x_s, y_1, y_2 \dots y_t\} = \{1, 2, 3 \dots n\}$ 。例如：1 2 3 4 5 可转化为 1 3 5 2 4。给出一个序列 $\{a_n\}$ （满足 $\{a_n\}$ 是 1 到 n 的一个排列），求一种方案，通过最少的操作次数是它变成升序序列。

这一题从题面看，操作的定义比较复杂，没有一个明显的切入点，很难设计出一个能有效解决它的算法。其实只要找到题目中涉及的操作对应的不变量，问题就能迎刃而解。

为算法刻画和证明的方便，引入以下定义：

称一个 $p \times q$ 的矩阵 \mathbf{A} 为序列 $\{a_n\}$ 的母矩阵，当且仅当，矩阵 \mathbf{A} 中的所有非零元素，自上到下自左到右逐列读出得到 $\{a_n\}$ ，自左到右自下到上逐行读出得到升序序列。

称序列 $\{a_n\}$ 的所有母矩阵中，行数列数都最小的那个矩阵为序列 $\{a_n\}$ 的最简母矩阵。

例如：当 $n=5$ 时， $\{5, 3, 2, 4, 1\}$ 的最简母矩阵为 $\begin{pmatrix} 5 & 0 \\ 3 & 4 \\ 2 & 0 \\ 0 & 1 \end{pmatrix}$ 。

本题的算法只需要完成以下步骤即可。

- (i) 判断当前 $\{a_n\}$ 是否是升序序列，若是则打印输出结束程序，若否转(ii)
- (ii) 计算 $\{a_n\}$ 的最简母矩阵 A (设 A 是一个 $p \times q$ 的矩阵)
- (iii) 对 $\{a_n\}$ 进行如下题意中的操作： A 的偶数行全部非零元素自左到右自上到下逐行读出得到 $b_1, b_2 \dots b_s$ ， A 的奇数行全部非零元素自左到右自上到下逐行读出得到 $b_{t+1}, b_{t+2} \dots b_{t+s}$ 。重复(i)。

算法证明如下：

只需证明：算法中给出的解是最优的，即操作步骤最少的。

首先： $p \times q$ 的最简母矩阵 A ，经上面的(iii)后，将成为一个 $\left\lceil \frac{p+1}{2} \right\rceil$ 行的矩阵。(这是因为 $\beta_1 = \alpha_1 \cup \alpha_2$ ， $\beta_2 = \alpha_3 \cup \alpha_4 \dots$ ，其中 α_k 表示 A 的第 k 行中非零元素构成的集合， β_k 对应 A 经操作后得到的矩阵 B)。

其次： $\{a_n\}$ 是一个升序序列，当且仅当，他的最简母矩阵是一个 $1 \times n$ 的矩阵。

例如： $a=(5 \ 2 \ 3 \ 1 \ 4)$ ， $b=(2 \ 3 \ 4 \ 5 \ 1)$ ，可得到最简母矩阵 $A=$

$$\begin{pmatrix} 5 & 0 & 0 \\ 2 & 3 & 4 \\ 0 & 1 & 0 \end{pmatrix}, \text{ 则得到 } B = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

由上面这两个事实可以得到，算法中给出的方案含有且只含有 $\lceil \log_2 p \rceil$ 个步骤。

下面证明，这是最优的。这只需证明：若 $\{a_n\}$ 的最简母矩阵有 p 行，则一次操作后得到 $\{b_n\}$ 的最简母矩阵至少有 $\left\lceil \frac{p+1}{2} \right\rceil$ 行。

我们用反证法。假设存在一种操作方案，使得 $\{b_n\}$ 的最简母矩阵 B 只有 p_0

行, $p_0 \leq \left\lceil \frac{p-1}{2} \right\rceil$ 。

则由抽屉原则, 必定存在 B 的一行(不妨设为第 p^* 行), 使得其中含有来自 A 的不同的三行的非零元素 (因为 $2p_0 < p$), 设其中 r_1, r_2, r_3 分别来自第 p_1, p_2, p_3 行。

不妨 $p_1 < p_2 < p_3$, 则由性质二 (在 A 中) 知: $r_1 > r_2 > r_3$ 。这样又由性质二 (在 B 中), r_1, r_2, r_3 在 B 中第 p^* 行中从左到右依次是 r_3, r_2, r_1 。所以, 由性质一 (在 B 中), r_1, r_2, r_3 在 $\{b_n\}$ 中出现的先后顺序为先 r_3 , 再 r_2 , 最后 r_1 。

再由抽屉原则, r_1, r_2, r_3 中必有两个, 同属于以下两个集合之一: $\{b_1, b_2 \dots b_s\}, \{b_{s+1}, b_{s+2} \dots b_{s+t}\}$ 。不妨设 r_1, r_2 同属于 $\{b_1, b_2 \dots b_s\}$ 。

这样, r_1, r_2 在 $\{b_n\}$ 中出现的先后顺序与 $\{a_n\}$ 中相同。即在 $\{a_n\}$ 中也是 r_2 先出现 r_1 后出现。

我们考虑以下这些数: $r_2, r_2+1, r_2+2 \dots r_1-1, r_1$ 。

一方面, 由于 r_2, r_1 在 B 中同行, 由性质二 (在 B 中), 所有这些数在 B 中也同行, 且位置介于 r_2, r_1 之间。所以, 他们在 $\{b_n\}$ 中按照顺序依次 (不一定连续) 出现。同时, 由性质一, 他们一定同属于 $\{b_1, b_2 \dots b_s\}$, 所以, 他们在 $\{a_n\}$ 中也按照顺序依次 (不一定连续) 出现。

另一方面, 由性质二 (在 A 中), 他们在 A 中的行号介于 $[p_1, p_2]$, 且严格不增。

由于 r_2, r_1 在 A 中不同行, 所以其中必有相邻两个数在 A 中不同行, 设为 r_0, r_0+1 。

由性质二, r_0, r_0+1 一定处在相邻两行。(否则, 他们中间的行只能全零,

那这些行就可以省略，这与 A 的最简性是矛盾的。)

另外，由前面的结论，在 $\{a_n\}$ 中 r_0 先出现， $r_0 + 1$ 后出现。所以，在 A 中 r_0 的列号小于 $r_0 + 1$ 的列号。即 r_0 、 $r_0 + 1$ 的位置关系是这样的：

$\begin{pmatrix} 0 & 0 & 0 & r_0 + 1 & \dots & \dots \\ \dots & \dots & r_0 & 0 & 0 & 0 \end{pmatrix}$ 。显然这两行是可以合并的，这与 A 的最简性矛盾！

证毕。

当然，给出这样一个算法，不是本文的关键。注意到这个算法的操作对象是矩阵，如果使用一般的实现方法，不可避免在最坏情况下单次操作的复杂度达到 $O(n^2)$ ，显然，复杂度过高，还有优化余地。

要进一步优化程序，可以着眼数据结构。不难发现，矩阵中的非零元素只有 n 个，因此可以只记录这些元素的位置，这样就等于把矩阵的规模缩小到了 $O(n)$ ，这一优化方法使得时空复杂度都得到了巨大的改进。因此，算法的总时间复杂度是 $O(n \cdot \text{ans})$ ，同时是输出的复杂度，空间复杂度 $O(n)$ ，相比朴素实现矩阵的效果时空复杂度都得到了大大改进。

另外，仔细分析证明，可以发现列数的最优性是可以忽略的。这样，在具体实现时让矩阵的每列只有一个元素，使得每个元素的位置信息可以更加简单，程序编码更加简便。

回顾这一问题的解决，关键在于“提炼”方法的使用忽略了矩阵中的非零元素这一无用信息，从而可以用线性表存储矩阵结构，使时间复杂度从 $O(n^2 \cdot \text{ans})$ 降到 $O(n \cdot \text{ans})$ ，空间复杂度从 $O(n^2)$ 降到 $O(n)$ 。

问题二：CEOI 2007 Day 2 Necklaces

问题描述：要求编译一个库，能够对若干已知的整数串进行两个操作：

(i) 在某个已知串的左端或右端增加或减少一个元素，得到一个新的已知的

串。

(ii) 输出某个已知串的最左端或最右端的数。

在问题的一开始，只有一个已知的串：空串。

这道题是一道典型的“赤裸裸”的数据结构题，若要进行 n 个操作，则最坏情况下，需要维护 $O(n)$ 个长为 $O(n)$ 的字符串。显然，朴素的实现无论在时间上还是空间上总的复杂度都要达到平方阶，这是不能令人满意的。

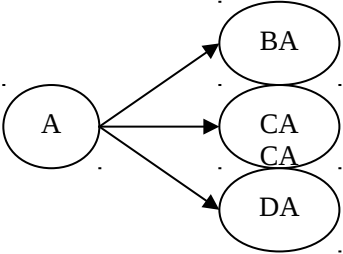
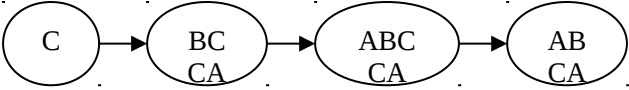
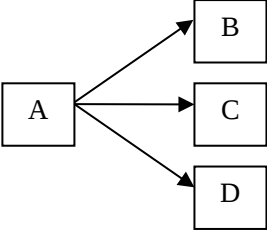
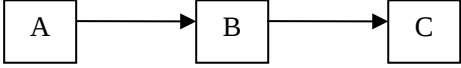
分析这道题的特点，发现由于每个串都要由之前的某个串经过微小加工得到，所以，这些串之间一定有很多公共部分。不妨利用这一点来优化数据结构。

首先利用两种特殊情况来帮助思维（见表 1）：

(i) 在某一串左侧，分别加上不同的元素，得到不同的串。这一特例容易让我们想到星形存储。

(ii) 在某一串左侧，加上一列元素，在从右段依次删除。这一特例容易让我们想到链形存储加上首尾指针。

表 1 两种特殊情况和它们的存储方式

| | | |
|------|---|--|
| 特殊情况 |  |  |
| 存储方式 |  |  |

这样，我们就需要设计一种数据结构，上面的星形和链形统一起来。很明显树形结构可以胜任。有了树形的构想，整个数据结构的设计就不难了。

根据上面的分析，可以维护一个数据结构，称它 **left-right tree**。 **left tree** 和 **right tree** 都是 Trie(字母树)。

对于每个已知的串 **s**，我们把它任意（实际上不是任意的）分成左右两部分，左串存储在 **left tree** 中，右串存储在 **right tree** 中。这样，我们只需要四个量：**left_root**, **left_leave**, **right_root**, **right_leave**，就可以描述 **s**：**s** 就由 **left tree** 中 **left_leave** 到 **left_root** 的路径代表的串，与 **right tree** 中 **right_root** 到 **right_leave** 的路径代表的串组成。

下面给出一个 left-right tree 的例子（见图 1）

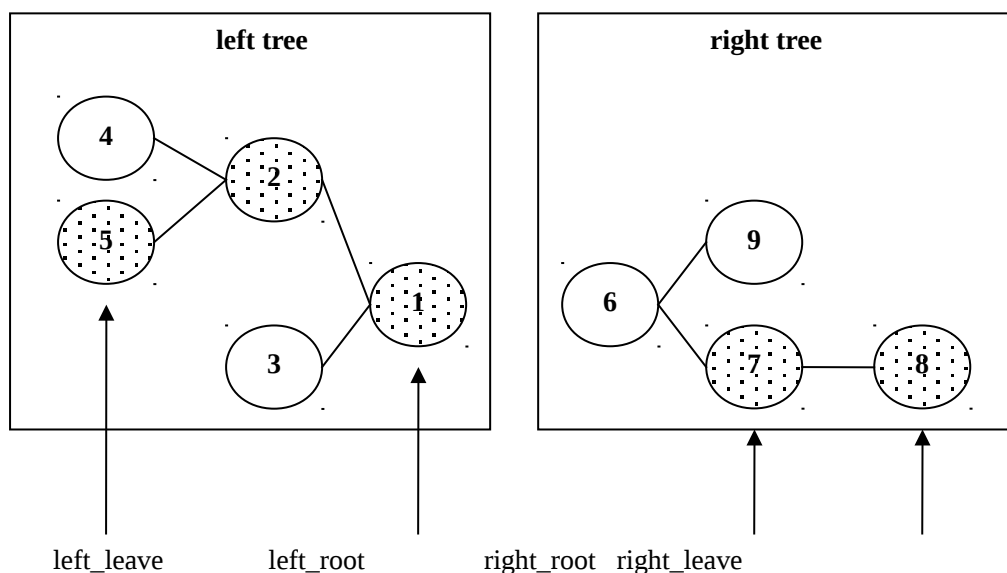


图 1 整数串 (5, 2, 1, 7, 8) 在一棵 left-right tree 中的表示

同时，我们允许，左串和右串中任意一串为空，甚至都为空，这时实际 **s** 就为空。并规定：若在左树中的串为空，则 **left_root=0**；同样若在右树中的串为空，则 **right_root=0**。

有了这样的数据结构，各个操作的实现已经不难了，只要实现好下面几个情况即可：

- (i) 已知树中某链的两端点，求底部端点的父亲
- (ii) 已知树中某链的两端点，求顶部端点在链中的儿子

这些看起来都是 **Trie** 的基本操作，但由于本题中 **Trie** 的元素规模不受限制，故(ii)的实现并不平凡，因此仍需要探讨，关于树形结构的化简将在第二部分着重讨论。

总结本题，关键在于，用“缩”的方法，利用 **Trie**，将串数组中的重复数据压缩，用树形结构简化储存，空间复杂度从 $O(n^2)$ 到 $O(n)$ ，有了质的飞跃。对于时间复杂度，要看(i)(ii)的实现，但也是离不开这个树形结构的前提的。

2. 树形结构的化简

树结构是一种被许多人深入研究过的数据结构，树结构也被用作各种高级数据结构的基本模型，例如并查集、堆等等。本文主要讨论的是图论树。同样以两个问题为例：

问题三：浙江 **2007** 年省选 捉迷藏

问题描述：给定一棵树，每个节点要么是黑色，要么是白色，能执行两个操作：把某一个点取反色，返回距离最远的黑色点对。

这也是一道“赤裸裸”的数据结构题。应该说这是众多数据结构题中的一道难题。

仔细读题，不难发现这是一个树结构上的“局部参数调整+整体属性返回”的问题。这一类特点的问题要求在线性数组上完成在信息学竞赛中式十分常见的，例如如“每次改变一个值，动态维护：最大值、和最大的子串、所有相邻数对中最小的差”等等，堆、线段树、树状数组都是常用手段。但对于本体中的树形树形结构，这些方法都难以直接使用。不妨尝试把它转化成线性结构来解决。

定义一种对一棵树的括号编码。这种编码方式很直观，所以，这里不给出严格的定义，用以下这棵树为例（图 2）。

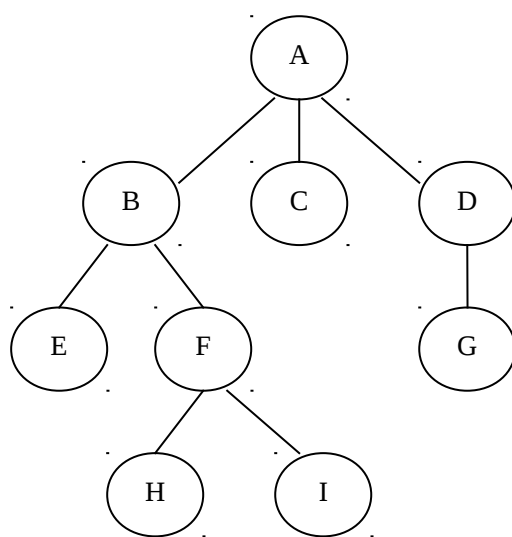


图 2 一棵有根树

可以先序遍历后写成: $[A[B[E][F[H][I]]][C][D[G]]]$

去掉字母后的串: $[[[[[[[]]]]][]]]$ 就称为这棵树的括号编码。(这个编码本质上是由深度优先遍历得到的)

考察两个结点, 如 E 和 G, 取出介于它们之间的那段括号编码: $[[[[[]]][]]$ 。

把匹配的括号去掉(用小括号和大括号表示), 得到: $([[[]]])$

我们看到 2 个 $)$ 和 2 个 $[$, 也就是说, 在树中, 从 E 向上爬 2 步, 再向下走 2 步就到了 G。

注意到, 题目中需要的信息只有这棵树中点与点的距离, 所以, 贮存编码中匹配的括号是没有意义的。因此, 对于介于两个节点间的一段括号编码 S, 可以用一个二元组 (a,b) 描述它, 即这段编码去掉匹配括号后有 a 个 $)$ 和 b 个 $[$ 。所以, 对于两个点 PQ, 如果介于某两点 PQ 之间编码 S 可表示为 (a,b) , PQ 之间的距离就是 $a+b$ 。

也就是说, 题目只需要动态维护: $\{a+b | S'(a,b) \text{ 是 } S \text{ 的一个子串, 且 } S' \text{ 介于两个黑点之间}\}$, 这里 s 是整棵树的括号编码。我们把这个量记为 $\text{dis}(s)$ 。

现在, 如果可以通过左边一半的统计信息和右边一半的统计信息, 得到整段编码的统计, 这道题就可以用熟悉的线段树解决了。

这需要下面的分析。

考虑对于两段括号编码 $S_1(a_1,b_1)$ 和 $S_2(a_2,b_2)$, 如果它们连接起来形成 $S(a,b)$ 。注意到 S_1 、 S_2 相连时又形成了 $\min\{b,c\}$ 对成对的括号, 合并后它们会被抵消掉。所以:

当 $a_2 < b_1$ 时第一段 $[$ 就被消完了, 两段 $)$ 连在一起, 例如:

$[[[] +]]][] = [](())[] = []]][]$

当 $a_2 \geq b_1$ 时第二段 $)$ 就被消完了, 两段 $[$ 连在一起, 例如:

$[[[] +]]][] = [][(())[] = [][[[]$

这样，就得到了一个十分有用的结论：

当 $a_2 < b_1$ 时 $(a,b) = (a_1 - b_1 + a_2, b_2)$ ，当 $a_2 \geq b_1$ 时 $(a,b) = (a_1, b_1 - a_2 + b_2)$ 。
(*)

由此，又得到几个简单的推论：

$$(i) \quad a+b = a_1+b_2+|a_2-b_1| = \max\{(a_1-b_1)+(a_2+b_2), (a_1+b_1)-(a_2+b_2)\}$$

$$(ii) \quad a-b = a_1-b_1+a_2-b_2$$

$$(iii) \quad b-a = b_2-a_2+b_1-a_1$$

由(i)式，可以发现，要维护 $\text{dis}(s)$ ，就必须对子串维护以下四个量：

right_plus: $\max\{a+b \mid S'(a,b) \text{ 是 } S \text{ 的一个后缀, 且 } S' \text{ 紧接在一个黑点之后}\}$

right_minus: $\max\{a-b \mid S'(a,b) \text{ 是 } S \text{ 的一个后缀, 且 } S' \text{ 紧接在一个黑点之后}\}$

left_plus: $\max\{a+b \mid S'(a,b) \text{ 是 } S \text{ 的一个前缀, 且有一个黑点紧接在 } S \text{ 之后}\}$

left_minus: $\max\{b-a \mid S'(a,b) \text{ 是 } S \text{ 的一个前缀, 且有一个黑点紧接在 } S \text{ 之后}\}$

这样，对于 $S=S_1+S_2$ ，其中 $S_1(a,b)$ 、 $S_2(c,d)$ 、 $S(e,f)$ ，就有

$$\text{当 } b \geq c \text{ 时 } (e,f) = (a, b-c+d), \text{ 当 } b < c \text{ 时 } (e,f) = (a-b+c, d) \quad ①$$

$$\text{dis}(S) = \max\{\text{right_plus}(S_1) + \text{left_minus}(S_2), \text{right_minus}(S_1) + \text{left_plus}(S_2), \text{dis}(S_1), \text{dis}(S_2)\} \quad ②$$

那么，增加这四个参数是否就够了呢？是的，因为：

$$\text{right_plus}(S) = \max\{\text{right_plus}(S_1) -$$

$c+d$
, $\text{right_plus}(S_2)$
 $\text{right_plus}(S_1) -$

mi
nus
(S1
) + c
+ d,
rig
ht_
plu
s(S
2))
③

$$\text{right_minus}(S) = \max\{\text{right_minus}(S1) + c - d, \text{right_minus}(S2)\} \quad ④$$

$$\text{left_plus}(S) = \max\{\text{left_plus}(S2) - b + a, \text{left_minus}(S2) + b + a, \text{left_plus}(S1)\} \quad ⑤$$

$$\text{left_minus}(S) = \max\{\text{left_minus}(S2) + b - a, \text{left_minus}(S1)\} \quad ⑥$$

这样一来，由①②③④⑤⑥，就可以用线段树处理编码串了。实际实现的时候，在编码串中加进结点标号会更方便，对于底层结点，如果对应字符是一个括号或者一个白点，那么 `right_plus`、`right_minus`、`left_plus`、`left_minus`、`dis` 的值都是 `-maxlongint`；如果对应字符是一个黑点，那么 `right_plus`、`right_minus`、`left_plus`、`left_minus` 都是 0，`dis` 是 `-maxlongint`。

现在这个题得到圆满解决，回顾这个过程，可以发现用一个串表达整棵树的信息是关键，这一“压”使得线段树这一强大工具得以利用。

问题四：2005 年国家集训队何林论文 树的统计

问题描述：给定一棵含有 n 个节点的树，所有的节点分别编号为 $1, 2, 3, \dots, n$ 。对于编号为 v 的节点，定义 $t(v)$ 为 v 的后代中所有编号小于 v 的节点个数。求 $t(1), t(2), t(3), \dots, t(n)$ 。

对于本题，可以毫不费力的得出一个 $O(n^2)$ 的朴素算法，但这个时间复杂

度太高。下面介绍来看一种别出心裁的算法。（注：此题的题目和题解摘自 2005 年国家集训队何林论文，命题者和算法提供者是雷涛同学）

先引入一下定义：记 $T(v)$ 表示 v 的后代构成的集合， $Y(v)$ 表示顺序在 v 之后 v 的兄弟构成的集合， $E(v)$ 表示顺序在 v 之后 v 的兄弟构成的集合， $P(v)$ 表示 v 的祖先构成的集合。

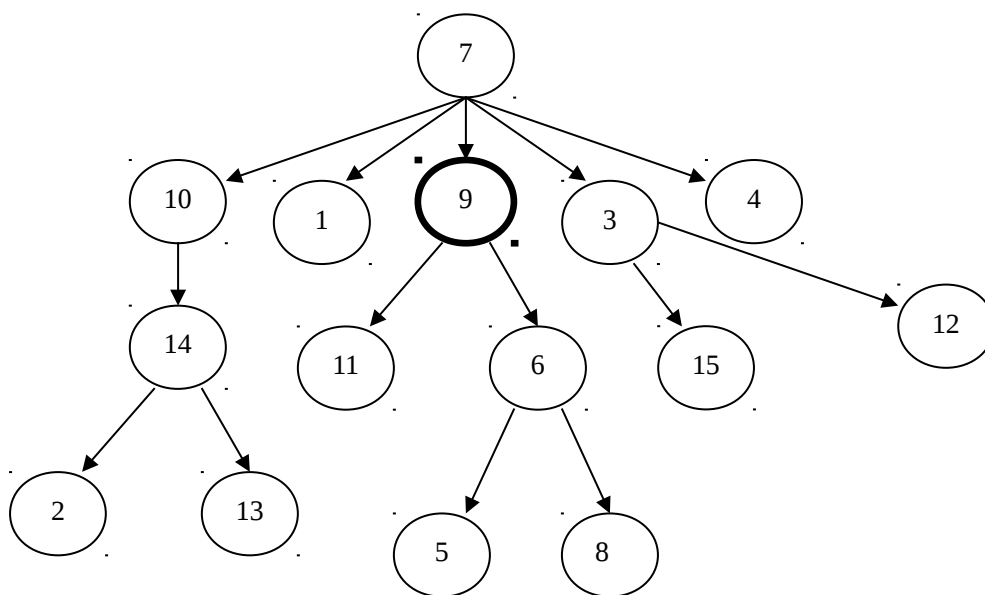


图 3

如图 3 中，

$T(3)=\{12,15\}$, $Y(5)=\{8\}$, $E(6)=\{11\}$, $P(8)=\{6,7,9\}$ 。

下面描述算法的操作，仍以图 3 中这棵树为例。

DFS 遍历（Deep First Search，深度优先遍历）该树，然后按照访问到的先后顺序把节点依次写下来：

7 10 14 2 13 1 **9** 11 6 5 8 3 15 12 4（该序列称为“**DFS** 序列”）

这个序列中数字 9 后面的部分已在图 4 中用实线框出来。

然后把每个节点的儿子先后顺序倒过来，重新遍历，得到的序列如下：

7 4 3 12 15 **9** 6 8 5 11 1 10 14 13 2（该序列称为“逆 **DFS** 序列”）

这个序列中数字 9 后面的部分已在图 4 中用虚线框出来。

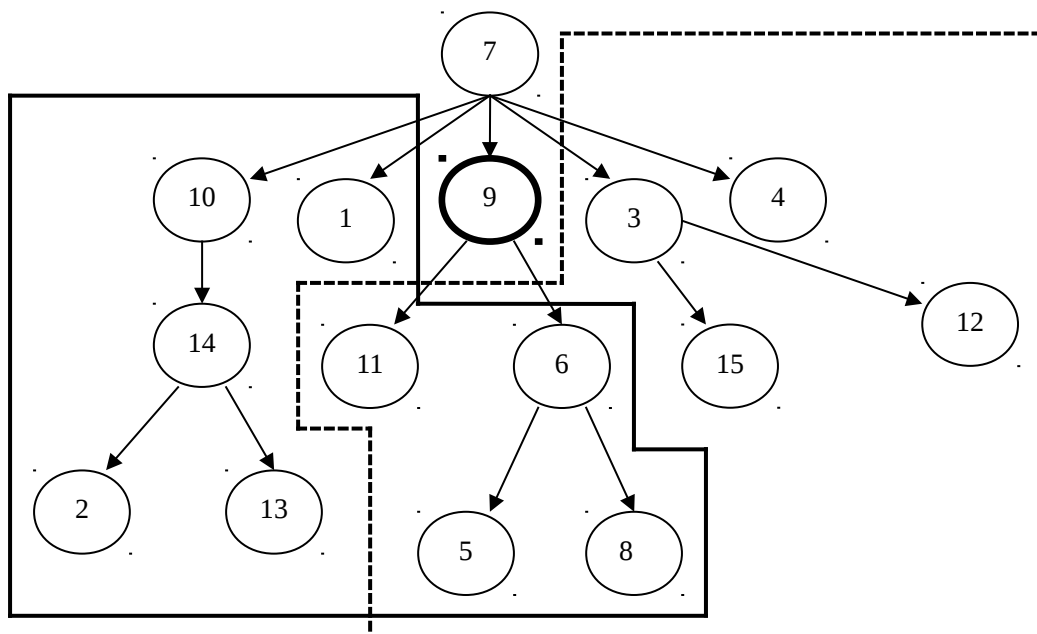


图 4

注意到实线框和虚线框有一个重叠区域，而这个区域正是“9 的所有后代”。另外，除了实线框和虚线框囊括的区域外，还有一个未被触及的盲区：那就是 9 本身和它的直系祖先。

定义 $f(v, S)$ 表示在 S 所描述的集合或者区域中，是小于 v 的节点数量。从图上直观地看，可以得到如下结论： $f(v, T(v)) = f(v, \text{DFS 序列中 } v \text{ 之后的部分}) + f(v, \text{逆 DFS 序列中 } v \text{ 之后的部分}) + f(v, p(v)) - f(v, \text{整棵树})$ 。

图上看很直观，下面给出它的严格证明。

因为，DFS 序列中 v 之后的部分 =

$$\bigcup_{w \in P(v) \cup \{v\}, u \in Y(w) \cup \{v\}} T(u) \cup \bigcup_{w \in P(v) \cup \{v\}} Y(w)。$$

对应的，逆 DFS 序列中 v 之后的部分 =

$$\bigcup_{w \in P(v) \cup \{v\}, u \in E(w) \cup \{v\}} T(u) \cup \bigcup_{w \in P(v) \cup \{v\}} E(w)。$$

$$\text{由于 } \bigcup_{w \in P(v) \cup \{v\}, u \in E(w) \cup \{v\}} T(u) \cap \bigcup_{w \in P(v) \cup \{v\}} Y(w) = \phi ,$$

$$\bigcup_{w \in P(v) \cup \{v\}, u \in Y(w) \cup \{v\}} T(u) \cap \bigcup_{w \in P(v) \cup \{v\}} E(w) = \phi , ,$$

$$\bigcup_{w \in P(v) \cup \{v\}} E(w) \cap \bigcup_{w \in P(v) \cup \{v\}} Y(w) = \phi .$$

所以:

$$\left(\bigcup_{w \in P(v) \cup \{v\}, u \in Y(w) \cup \{v\}} T(u) \cup \bigcup_{w \in P(v) \cup \{v\}} Y(w) \right) \cap \left(\bigcup_{w \in P(v) \cup \{v\}, u \in E(w) \cup \{v\}} T(u) \cup \bigcup_{w \in P(v) \cup \{v\}} E(w) \right)$$

$$= \bigcup_{w \in P(v) \cup \{v\}, u \in Y(w) \cup \{v\}} T(u) \cap \bigcup_{w \in P(v) \cup \{v\}, u \in E(w) \cup \{v\}} T(u) = T(v)$$

$$(\text{最后一个等号是因为 } \left(\bigcup_{w \in P(v) \cup \{v\}} E(w) \cup \{v\} \right) \cap \left(\bigcup_{w \in P(v) \cup \{v\}} Y(w) \cup \{v\} \right) = \{v\})$$

容易观察到 $f(v, \text{整棵树}) = v - 1$, 而 $t(v) = f(v, T(v))$, 所以:

$$\mathbf{t(v) = f(v, \text{DFS 序列中 } v \text{ 之后的部分}) + f(v, \text{逆 DFS 序列中 } v \text{ 之后的部分}) + f(v, P(v)) - v + 1}$$

这样, 通过一次 DFS 遍历联合树状数组就能以 $O(n \log n)$ 的时间复杂度求出所有节点的直系祖先中有多少个比它本身小。对于 DFS 序列和逆 DFS 序列, 可以采用树状数组求出序列每个元素后面有多少个比它本身小, 这一步时间复杂度也是 $O(n \log n)$ 。这两部分用线段树实现也是可行的, 只是树状数组在时间空间效能上更优一些 (准确地说, 有一个常数)。

回顾整个问题的解决, 这个算法最巧妙的地方, 在于用“缩”的方法, 忽略树中的一些拓扑信息, 把树对应到了两个序列: DFS 序列和逆 DFS 序列。用线性表存储了树状结构中的有效信息。另外, 这两个序列的构造是极具创意的。

上面两个问题的解决, 都利用 DFS 序将树结构简化到了线性结构, 使得线段树和树状数组这样的工具得以应用。事实上, BFS 遍历 (宽度优先遍历) 在“提炼”“压”“缩”数据结构时也有它的功用。这将在第三部分涉及到。

问题五: 问题二的遗留问题

问题描述：给定一棵有根树，在线回答两种询问：

(i) 已知树中某链的两端点，求底部端点的父亲

(ii) 已知树中某链的两端点，求顶部端点在链中的儿子

显然，如果用普通的左儿子右兄弟方法存储会对操作(ii)力不从心。这里不妨先做这样一个小转化。在已知两端点深度的情况下，求顶部端点在链中的儿子可以转化为求底端结点的一个“超级父亲”。有了这样的转化，就可以对树结构进行一个改进，忽略掉结点上所有的后辈信息，而只储存包括父亲在内的祖先信息。

在这样的树结构上，“超级父亲”的实现方法就相对简单。设 $SF(x,1)=f(x)$ ， $SF(x,k+1)=f(SF(x,k))$ 。同时，在树中每个点 x 上储存： $d(x)$ —— x 的深度； $Su(x,k)$ —— $SF(x,2^k)$ 。

这样，可以在 $O(\log n)$ 的时间内计算 $SF(x,k)$ 。而维护 Su 也只需要 $O(\log n)$ 的时间。从而，整个算法的时间复杂度为 $O(n \log n) - O(\log n)$ 。

总结这个问题的解决，通过“提炼”树结构，跳出“左儿子右兄弟”常规存储思路，忽略在本题中没有价值的节点的后代信息，从而完成了“超级父亲”这个优秀的数据结构。

3 . 图结构的化简

图结构是一种复杂的结构，直接在图上动手往往很困难，所以一般情况下都要“压”到一些简单的结构，例如树结构。下面两个题是用不同手段“压”图的例子。

问题六：ural 1557 Network Attack

问题描述：给定一个无向连通图，若从中删去两条边能使它不连通，求所有这样的方案的总数。图点数 n 边数 m 。

显然，必须通过预处理先找到所有的 **bridge**（指那些边如果删除一条图就不连通的边），并把它们删除（**bridge** 的两个端点合并）。这样，就可以假定现在图中没有 **bridge**。

作为一个图的问题，直接处理十分困难，因此，解决这道题需要利用图的 DFS 遍历帮助思维。为算法叙述和分析的方便，引入下面定义：

设 n 个点为 $P_1 P_2 P_3 \dots P_n$ 。则任取一点，不妨设为 P_1 ，我们构建以 P_1 为根的 **DFS** 树。设 $T(i)$ 表示以 P_i 为根的子树中结点构成的集合， $R(i)$ 表示 P_i 的所有祖先构成的集合。深度 $d(i)$ 表示结点 P_i 的深度，其中规定 $d(1)=1$ ， $d(i)=d(j)+1$ 若 j 是 i 的父亲。 $U(A,B)$ 表示点集 A 中的点与点集 B 中的点所连边构成的集合。 $W(A,B)$ 表示点集 A 中的点与点集 B 中的点所连回边构成的集合。 $s(i)$ 表示 $T(i)$ 与 $R(i)$ 之间连的边数，即 $s(i)=|U(T(i),R(i))|$ 。 $t(i)$ 表示 $W(T(i),R(i))$ 中，相关深度最大的 $R(i)$ 节点的深度。

为解释上面这些定义我们以图 5 为例：

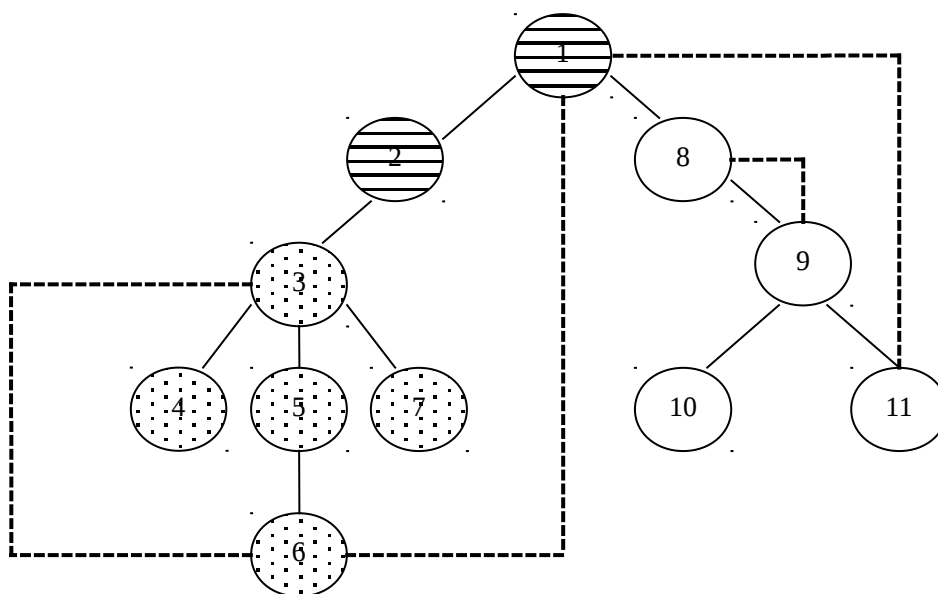


图 5

图 5 中浅色阴影的点构成了 $T(3)$ ，横条阴影的点构成了 $R(3)$ ， $d(8)=2$ ， $d(5)=4$ ， $s(5)=3$ ， $s(8)=2$ ， $t(3)=1$ ， $t(9)=2$ ， P_8P_9 之间连了一条树边一条回边。

熟知，在无向图的 DFS 遍历中有这样一个性质：图中不存在 DFS 遍历树的横向边。利用这一性质，不难发现这样一个重要的事实：（分割后的两部分中）不包含根（即 P_1 ）的那部分，一定具有形式： $T(i)$ 或 $T(i) \setminus T(j)$ （其中 P_i 是

P_j 的祖先）。

对这两种情况分类讨论：

若它的形式为 $T(i)$ ，则 2-cuts 的两条边一定属于集合 $U(T(i), T(1) \setminus T(i))$ 。由前面无向图的性质， $U(T(i), T(1) \setminus T(i)) = U(T(i), R(i))$ 。所以此时必需 $s(i) = |U(T(i), R(i))| = |U(T(i), T(1) \setminus T(i))| = 2$ 。

若它的形式为 $T(i) \setminus T(j)$ ，则 2-cuts 的两条边一定分别属于集合

$U(T(i), T(1) \setminus T(i))$ 和 $U(T(j), T(i) \setminus T(j))$ ，注意到 P_i 、 P_j 与各自父亲所连的树边，一定分别属于上面这两个集合。所以 2-cuts 必须只有这两条边。因此， $W(T(i), T(1) \setminus T(i)) = \emptyset$ ， $W(T(j), T(i) \setminus T(j)) = \emptyset$ ，所以 $W(R(i), T(i)) = W(R(j), T(j))$ 。

所以，满足条件的两条边有且只有以下两种情况：

(i) $s(i)=2$ 时， $U(T(i), R(i))$ 中的两条边。

(ii) P_i 是 P_j 的祖先，且 $W(R(i), T(i)) = W(R(j), T(j))$ 时， P_i

P_j 与各自父亲所连的边。

有了这个结论，程序不难编了，具体的实现这里不再展开（可以参看附录）。

从解决这个题的过程中可以看到，DFS 遍历使得原来普通的无向图有了更多可以利用的性质。可以说，DFS 遍历作为一种分析方式十分有效。

问题七：ural 1569 Networking the “Iset”

问题描述：输入一个无向图 $G=(V,E)$ ，求这个图的直径最小生成树。

这个题的题面很简洁，但和多跟图有关的问题一样，要解决这个问题却不好下手。因此，必须进行深入的分析。

为了叙述的方便，引入下面定义： $l(v) = \max\{d(u, v) | u, v \text{ 是一个图中的点}\}$ 。树的中心：树中 $l(v)$ 最小的点。对于一棵树，条件 X 成立，当且仅当，存在一个满足 $d(v_i, v_j) = 1$ 点 v_j ，使得，对于任意满足 $d(v_i, v_k) = d_i$ 的点 v_k ，都有 v_j 是 v_k 的祖先，这里 v_i, v_j, v_k 都是树中的节点。

这里指出关于 BFS 树的两条性质：一个图中以某一定点为根的 BFS 树不一定唯一的，但以某一定点为根的 BFS 的深度都相等。这些性质是很显然的，不再证明。

先从一棵树的直径入手分析这个问题。

尽管一棵树的直径是不唯一的（直径的长是确定），但对于树的中心有两个结论：当直径长为偶数，树的中心是唯一；当树的直径长为奇数，树的中心是唯二的。

下面证明这两个结论：

当树的直径 D 是偶数。设有一条直径是 AB ， AB 中点是 P 。一方面，对于任意一个点 C ，设 AB 上距离 C 最近的点为 Q ，不妨 Q 在 AP 上，则 $CP=BC-BP$

$BP \leq AB-BP = \frac{D}{2}$ ，同时 $AP=BP = \frac{D}{2}$ ，所以 $l(P) = \frac{D}{2}$ 。另一方面，对于任意一个不是 P 的点 C ，设 AB 上距离 C 最近的点为 Q ，不妨 Q 在 AP 上，则

$BC=BP+PQ+QC > \frac{D}{2}$ ，所以 $l(C) > \frac{D}{2}$ 。所以， P 就是这棵树的唯一的中心。

当树的直径 D 是奇数。设有一条直径是 AB ， AB 中央两点是 PQ ，其中 A 在 P 一侧， B 在 Q 一侧。一方面，对于任意一个点 C ，设 AB 上距离 C 最近的点为

R ，若 R 在 AP 上，则 $CP=BC-BP \leq AB-BP = \frac{D-1}{2}$ ，若 R 在 BP 上，则 $CP=AC-$

$AP \leq AB-AP = \frac{D+1}{2}$ 。同时 $BP = \frac{D+1}{2}$ ，所以 $l(P) = \frac{D+1}{2}$ 。同理 $l(Q) = \frac{D+1}{2}$ 。另一

方面，对于任意一个不是 P 、 Q 的点 C ，设 AB 上距离 C 最近的点为 R ，不妨 Q

在 AP 上，则 $BC=BP+PR+RC = \frac{D+1}{2} + PR + RC > \frac{D+1}{2}$ ，所以 $l(C) > \frac{D+1}{2}$ 。所以，

P 、 Q 就是这棵树的唯二的中心。

设 T 是图的某一棵直径最小生成树。直观地看会发现：当 T 的直径为偶数，且 P 是它的唯一的中心，图中任意一棵以 P 为根的 BFS 树都是该图的一棵直径最小生成树。当 T 的直径为奇数，且 PQ 是它的唯二的中心，则存在至少一棵以

P 为根的 BFS 树满足 X 条件，且所有满足 X 条件以 P 为根的 BFS 树都是该图的一棵直径最小生成树。证明如下：

先证明 T 的直径是偶数的情况。设 AB 是 T 的一条直径。有前面的证明对任意点 $CP \leq AP$ 且当 $C=A$ 或 B 是这个等号能取到。所以， $l(P) = AP$ 。注意到， $l(P)$ 就是以 P 为根的 BFS 树的深度。所以在一棵以 P 为根的 BFS 树 T' 中，任意两点之间的距离不超过 $2 \cdot l(P)$ ，即 $2 \cdot AP = T$ 的直径。所以 T' 的直径都不超过 T 的直径，即 T' 一定是该图的一棵直径最小生成树。

再证明 T 的直径是奇数的情况。设 AB 是 T 的一条直径，AB 长为 D。其中 A 在 P 一侧，B 在 Q 一侧。类似的偶数部分的推理，可以得到 $l(P) = BP = \frac{D+1}{2}$ 。由

先前的证明，对于任意 C，若 $PC = \frac{D+1}{2}$ 则 $QC = \frac{D-1}{2} = l(P) - 1$ ，而 $PQ = 1$ ，所以 $PC = PQ + QC$ 。又因为 $l(P)$ 就是以 P 为根的 BFS 树的深度，所以一定存在一棵以 P 为根的 BFS 树 T'' ，使得 T'' 中所有满足 $PC = l(P)$ 的点 C，都是 Q 的子孙。那么， T'' 是符合 X 条件的。另一方面，对于任意一棵符合 X 条件的以 P 为根的 BFS 树 T_0 ，设所有满足 $PC = l(P)$ 的点 C，都是 Q_0 的子孙。那么显然 T_0 的直径为 $2 \cdot l(P) - 1 = D$ ，且 P 和 Q_0 是它的唯二的中心。

这样，就只要统计以 V_i 点为中心的生成树的最小直径即可。由前面的结论，一棵以 V_i 为根的 BFS 树一定是以 V_i 点为中心的直径最小的生成树。

所以，只需要分析所有的 BFS 树：对于每个图中的 V_i ，求出以它为根 BFS 树 T_i 。树是不唯一的，优先考虑满足条件 X 的那一棵树。计算 T_i 直径时，当条件 X 对于 T_i 不成立时， T_i 的直径 $= 2d_i$ ；当条件 X 对于 T_i 成立时， T_i 的直径 $= 2d_i - 1$ 。从而，取出使得 T_i 的直径最大的 T_i ，就是答案。

这个题算是一个简单题，但从中可以看出 BFS 树对于图结构处理的作用。

上面这两个关于无向图的题，一个用 DFS 解决，一个用 BFS 解决。以这两个题目为代表，可以总结出这两个方法各自的特点。

其一：性质上（结构决定性质），DFS 遍历后的可以得到 DFS 树，图中的边在 DFS 树中要么是树边要么是回边；而 BFS 遍历后往往得到层状结构，图中

的边要么连接同一层中的两个点，要么连接相邻两层的两个点。

其二：用途上（性质决定用途），**DFS** 能较有效解决与连通性相关的问题（因为任意一棵子树只与它的祖先相联）；**BFS** 能较有效解决与点对距离相关的问题（由前面的性质，两点的距离，与所在层数密切相关）。

当然这些区别只是一般而言的，例如点对距离问题，当图十分稀疏时，极端情况下退化到树，那显然是 **DFS** 更有效了。

小结

本文通过几个例子说明了数据结构的“提炼”“压”“缩”在信息学中的应用。这三种方法一方面能减少存储规模，一方面能简化存储结构，归纳起来就是一个“减”和一个“简”。对于一些问题，数据结构的压缩使原算法得到优化，时空复杂度大大降低；对于另一些问题，不压缩数据结构就会在设计算法是一筹莫展，压缩后的数据结构提供了解决问题的突破口。

一般而言，如果一个算法中存储了无用信息（如前面例子中的矩阵零元素、树结构中结点后代信息、树结构拓扑关系等），或者重复信息就可以通过数据结构的化繁为简优化程序。另一方面，如果一个题目中的数据较为无序，杂乱无章就需要重新组织这些数据，往往，都是把它们组织为较原来更简明的数据结构，使得这些数据建有更多可以利用的性质，以帮助分析问题（如前面例子中，把一个无向图放到它的 DFS 树中，就有了“不存在横向边”的性质）。

所以说，数据结构的化繁为简不仅仅是一种优化手段，还是一种思维方式。

参考文献

1. 《算法艺术与信息学竞赛》，刘汝佳、黄亮著，清华大学出版社，2004 年 1 月第一版
2. 《数据结构与算法分析》，（美）[维斯](#) 著 [冯舜玺](#) 译，[机械工业出版社](#), 2004 年 1 月 1 日第一版
3. 2007 年浙江省省选试题及解答
4. 2005 年国家集训队何林论文
5. Timus Online Judge <http://acm.timus.ru/>

附录

题目来源

ural 1568 Train car sorting

原题

1568. Train car sorting

Time Limit: 1.0 second Memory Limit: 64 MB

There are several towers being built simultaneously in the city of Ekaterinburg. A lot of high quality hardware and materials is needed for the construction, and most materials are being shipped to the city via railroad. Railroad delivery isn't always as fast as contractors would like it to be. Trains spend too much time at the intermediate stations, being sorted and directed to different regions of the country.

As you know, freight train cars are sorted in the following way: the train is driven to a two-way switch, where each individual car can follow either left or right track. After that, the cars are joined back together. For example, if the order of the cars in the train is “1 2 3 4 5 6 7”, they can be split in two parts: “1 3 5” (left track) and “2 4 6 7” (right track), and then joined: “1 3 5 2 4 6 7”.

Help railroad workers speed up the sorting process. Write a program to rearrange cars according to the given order using the minimum number of join operations.

Input

The first line of input contains a single integer N — the number of cars in the train ($1 \leq N \leq 10000$). The second line contains N numbers — the initial ordering of the cars. Each car has an unique number from 1 to N . The cars have to be reordered so their numbers are increasing, starting from 1.

Output

The first line of output shall contain the integer K — minimum number of times the join must be done. The following $K + 1$ lines shall contain N numbers each. Output the initial ordering of the cars on the first of these lines; each following line shall contain the ordering achieved with the next join operation.

Samples

| input | output |
|------------------|---|
| 5 5 1 3 2 4 | 2 5 1 3 2 4 1 2 5 3 4 1 2 3 4 5 |
| 6 6 5 2 4 1 3 | 3 6 5 2 4 1 3 6 4 1 5 2 3 6 1 2 3 4 5 1 2 3 4 5 6 |

Problem Author: Sergey Pupyrev

Problem Source: The XIIth USU Programing Championship, October 6, 2007

问题描述

对于一个序列 $\{a_n\}$ ，定义一种操作，将 a 变成 b ，使得：

$$b_1 = a_{x_1}, \quad b_2 = a_{x_2} \dots b_s = a_{x_s}, \quad b_{s+1} = a_{y_1}, \quad b_{s+2} = a_{y_2} \dots b_{s+t} = a_{y_t}。 \text{ 其中}$$

$s+t=n$ ， $x_1 < x_2 < \dots < x_s$ ， $y_1 < y_2 < \dots < y_t$ ， $\{x_1, x_2 \dots x_s, y_1, y_2 \dots y_t\} = \{1, 2, 3 \dots n\}$ 。例如：1 2 3 4 5 可转化为 1 3 5 2 4。

给出一个序列 $\{a_n\}$ （满足 $\{a_n\}$ 是1到 n 的一个排列），求一种方案，通过最少的操作次数是他变成升序序列。

核心算法

为算法刻画和证明的方便，引入以下定义：

称一个 $p \times q$ 的矩阵 A 为序列 $\{a_n\}$ 的母矩阵，当且仅当，矩阵 A 中的所有

非零元素，自上到下自左到右逐列读出得到 $\{a_n\}$ （后称性质一），自左到右自

下到上逐行读出得到升序序列（后称性质二）。

称序列 $\{a_n\}$ 的所有母矩阵中，行数列数都最小的那个矩阵为序列 $\{a_n\}$ 的最简母矩阵。

例如：当 $n=5$ 时， $\{5, 3, 2, 4, 1\}$ 的最简母矩阵为 $\begin{pmatrix} 5 & 0 \\ 3 & 4 \\ 2 & 0 \\ 0 & 1 \end{pmatrix}$ 。

本题的算法只需要完成以下步骤即可。

- (i) 判断当前 $\{a_n\}$ 是否是升序序列，若是则打印输出结束程序，若否转
- (ii)
- (ii) 计算 $\{a_n\}$ 的最简母矩阵 A （设 A 是一个 $p \times q$ 的矩阵）
- (iii) 对 $\{a_n\}$ 进行如下题意中的操作： A 的偶数行全部非零元素自左到右自上到下逐行读出得到 $b_1, b_2 \dots b_s$ ， A 的奇数行全部非零元素自左到右自上到下逐行读出得到 $b_{t+1}, b_{t+2} \dots b_{t+s}$ 。重复(i)。

算法证明

只需证明：算法中给出的解是最优的，即操作步骤最少的。

首先： $p \times q$ 的最简母矩阵 A ，经上面的(iii)后，将成为一个 $\left\lceil \frac{p+1}{2} \right\rceil$ 行的矩

阵。(这是因为 $\beta_1 = \alpha_1 \cup \alpha_2$, $\beta_2 = \alpha_3 \cup \alpha_4 \dots$, 其中 α_k 表示 A 的第 k 行中非零元素构成的集合, β_k 对应 A 经操作后得到的矩阵 B)。

其次: $\{a_n\}$ 是一个升序序列, 当且仅当, 他的最简母矩阵是一个 $1 \times n$ 的矩阵。

由上面这两个事实可以得到, 算法中给出的方案只含有 $\lceil \log_2 p \rceil$ 个步骤。

下面证明, 这是最优的。这只需证明: 若 $\{a_n\}$ 的最简母矩阵有 p 行, 则一次操作后得到 $\{b_n\}$ 的最简母矩阵至少有 $\left\lceil \frac{p+1}{2} \right\rceil$ 行。

我们用反证法。假设存在一种操作方案, 使得 $\{b_n\}$ 的最简母矩阵 B 只有 p_0 行, $p_0 \leq \left\lceil \frac{p-1}{2} \right\rceil$ 。

则由抽屉原则, 必定存在 B 的一行(不妨设为第 p^* 行), 使得其中含有来自 A 的不同的三行的非零元素(因为 $2p_0 < p$), 设其中 r_1 r_2 r_3 分别来自第 p_1 p_2 p_3 行。

不妨 $p_1 < p_2 < p_3$, 则由性质二(在 A 中)知: $r_1 > r_2 > r_3$ 。这样又由性质二(在 B 中), r_1 r_2 r_3 在 B 中第 p^* 行中从左到右依次是 r_3 r_2 r_1 。所以,

由性质一（在 B 中）， r_1, r_2, r_3 在 $\{b_n\}$ 中出现的先后顺序为先 r_3 ，再 r_2 ，最后 r_1 。

再由抽屉原则， r_1, r_2, r_3 中必有两个，同属于以下两个集合之一： $\{b_1, b_2 \dots b_s\}$ ， $\{b_{s+1}, b_{s+2} \dots b_{s+t}\}$ 。不妨设 r_1, r_2 同属于 $\{b_1, b_2 \dots b_s\}$ 。

这样， r_1, r_2 在 $\{b_n\}$ 中出现的先后顺序与 $\{a_n\}$ 中相同。即在 $\{a_n\}$ 中也是 r_2 先出现 r_1 后出现。

我们考虑一下这些数： $r_2, r_2+1, r_2+2 \dots r_1-1, r_1$ 。

由于 r_2, r_1 在 B 中同行，由性质二（在 B 中），他们在 B 中都同行，且位置介于 r_2, r_1 之间。所以，他们在 B 中按照顺序依次（不一定连续）出现。同时，由性质一，他们一定同属于 $\{b_1, b_2 \dots b_s\}$ ，所以，他们在 A 中也按照顺序依次（不一定连续）出现。

另一方面，由性质二（在 A 中），他们在 A 中的行号介于 $[p_1, p_2]$ ，且严格不增。

由于 r_2, r_1 在 A 中不同行，所以其中必有相邻两个数在 A 中不同行，设为 r_0, r_0+1 。

由性质二， r_0 、 r_0+1 一定处在相邻两行。（否则，他们中间的行只能全零，那这些行就可以省略，这与 A 的最简性是矛盾的。）

另外，由前面的结论，在 $\{a_n\}$ 中 r_0 先出现， r_0+1 后出现。所以，在 A 中

r_0 的列号小于 r_0+1 的列号。即 r_0 、 r_0+1 的位置关系是这样的：

$\begin{pmatrix} 0 & 0 & 0 & r_0+1 & \dots & \dots \\ \dots & \dots & r_0 & 0 & 0 & 0 \end{pmatrix}$ 。显然这两行是可以合并的，这与 A 的最简性矛盾！

证毕！

算法实现

从证明可以看出，列的最简性是不重要的，我们在实现算法的时候，就忽略列的最简性，只要求矩阵满足行的最简性。这样，我们可以这样实现：每列只有一个数。这样的话，我们就不需要存储整个矩阵了（所有的零元素都省去了）

我们用 **a** 数组，保存序列。用 **b** 数组保存每个数所在的列号。用 **L** 数组保存每个数的行号。这样步骤(i)(ii)(iii)的时间复杂度都是 $O(n)$ 的，由于答案是 $O(\log n)$ 的，所以总的时间复杂度为 $O(n \log n)$ 。

题目来源

CEOI 2007 Day 2 Necklaces

原题

CEOI 2007 Day 2 Necklaces
Time limit: 3 s Memory limit: 128 MB

And Alice said: "I have the most beautiful necklace. From left to right, it consists of two red pearls, two green ones and one more red one." Beatrix was quick to answer: "Mine is even better. It looks almost like yours, but you would need to remove the two rightmost pearls and replace them by two blue ones." No sooner did she stop, when Caroline jumped in: "That is no match for my necklace. I have one more yellow pearl on the left." You probably will not be surprised when I tell you that Dominica also did not stay silent: "That is all boring. To get my necklace, you would have to take the Beatrice's one, remove the leftmost and the rightmost pearls, and add two black pearls to the left." And so it went on, until Zaida asked: "I got a bit confused; what was the color of the leftmost pearl of the Eugenie's necklace?" And silence answered her.

Task

Your task is to write a library (unit in Pascal) that is able to help with the conversations of this type. The interface of the library is described in what follows; you can find templates for the libraries in the directories `/mo/public/necklace/c`, `/mo/public/necklace/cpp`, and `/mo/public/necklace/pas` (the subdirectories for C and C++ also contain the header file `necklace.h`). There is no input or output.

Your library will work with a set of necklaces. A necklace is a sequence of integers between 0 and 1 000 000, ordered from left to right. Each necklace is identified by a nonnegative integer. Initially, only a necklace 0 that consists of an empty sequence is present.

In the beginning, the evaluation system will call exactly once the procedure `init`. Then, it will repeatedly call in an arbitrary order the functions `create` and `pearl`. In total, your library will be called at most 1 000 000 times in each test run.

The call to the procedure `create` corresponds to creating a new necklace. The number of the necklace is by one higher than the largest number of an existing necklace, i.e., the first call to `create` creates a necklace with number 1, the second one a necklace with number 2, etc. The new necklace is derived from the necklace number from by the operation specified by the parameters `operation`, `on left` and `param`:

- If the parameter operation is the letter R (as “remove”), a single integer is removed from the end of the necklace. The parameter param is ignored in this case.
- If the parameter operation is the letter A (as “add”), the integer param is added to the end of the necklace.

If on left is true (nonzero in C), then the operation is performed at the left end of the necklace, otherwise it is performed at the right end. The from parameter of the create call will always be smaller than the number of the necklace created by the call, i.e., it will refer to an existing necklace. You can always assume that no call will attempt to remove a pearl from an empty necklace.

The function pearl should return the integer on the left end of the necklace neck id if on left is true (nonzero in C/C++) and the integer on the right end of the necklace neck id, otherwise. The function pearl will be only called for non-empty necklaces that were created by the call of the function create before. This function does not alter the necklaces.

Example

The following example shows one possible sequence of calls of the functions and the returned values:

In Pascal:

```
init;
```

```
create (0, 'A', true, 5);
```

```
create (1, 'A', true, 3);
```

```
pearl (2, false); { returns 5 }
```

```
create (2, 'R', false, 0);
```

```
pearl (3, false); { returns 3 }
```

```
pearl (2, false); { returns 5 }
```

Testing your library

In addition to the templates for your library, the subdirectories of /mo/public/necklace contain the files neck_main.c, neck_main.cpp, and neck_main.pas which are the main source files that can be compiled with your library and will access the library routines

in the way shown in the example (the program will inform you whether the return values of the function `pearl` are the correct ones).

The script compile with the parameter `necklace` will compile your library with the default main file (`neck_main.c`, `neck_main.cpp` or `neck_main.pas`) and the resulting program `necklace` will allow you to test your library as described further¹. If the program is run without parameters, it calls your library as described in the example and checks whether the return values of the function `pearl` are the correct ones (if not, a message is written to the standard error output). If the program is called with the parameter `-i`, it switches to the interactive mode. It reads requests from the standard input, passes them to the library and writes the responses to the standard output.

In the interactive mode, the program first calls the procedure `init` and after this procedure terminates, it outputs `Initiated`. Then, the following three types of requests are expected (each on a separate line):

- `create from operation on left param`

This request causes the program to call the procedure `create` with the parameters `from`, `operation`, `on left` and `param`. After the procedure is finished, the program outputs `Done`.

- `pearl neck id on left`

This request causes the program to call the function `pearl` with the parameters `neck id` and `on left`. After the function is finished, the program outputs `Returns X` where `X` is the number returned by the function.

- `quit`

This request causes the program to terminate.

Note that the program in the interactive mode only provides you with responses of your library and it does not check whether the responses are correct.

The following shows the requests and responses of the program running in the interactive

mode corresponding to the sequence of calls to the library from the example.

In Pascal:

Initiated

create 0 A true 5

Done

create 1 A true 3

Done

pearl 2 false

Returns 5

create 2 R false 0

Done

pearl 3 false

Returns 3

pearl 2 false

Returns 5

quit

问题描述

要求编译一个库，能够对若干已知的整数串进行两个操作：

(i) 在某个已知串的左端或右端增加或减少一个元素，得到一个新的已知的串。

(ii) 输出某个已知串的最左端或最右端的数。

在问题的一开始，只有一个已知的串：空串。

核心算法

维护一个数据结构：left-right tree。Left tree 和 right tree 都是字母树，Left tree 储存所有从左端插入的元素，right tree 储存所有右端插入的元素。

对于每个已知的串 s ，储存四个量：
 $left_root, left_leave, right_root, right_leave$ 。表示 s 由 $left\ tree$ 中 $left_leave$ 到 $left_root$ 的路径代表的串，与 $right\ tree$ 中 $right_root$ 到 $right_leave$ 的路径代表的串组成。
 若在左树中的串为空，则 $left_root=0$ ；同样若在右树中的串为空，则 $right_root=0$ 。

对于操作：在已知串 $s(lr, ll, rr, rl)$ 的左端插入一个元素 x 得到 $s'(lr', ll', rr', rl')$ 。

当 $lr=0$ 时，在 $left\ tree$ 中添加一个根的儿子 t ，他的值为 x 。
 $lr'=ll'=t, rr'=rr, rl'=rl$ 。

当 $lr \neq 0$ 时，在 $left\ tree$ 中添加一个 ll 的儿子 t ，他的值为 x 。
 $lr'=lr, ll'=t, rr'=rr, rl'=rl$ 。

对于操作：在已知串 $s(lr, ll, rr, rl)$ 的左端删除一个元素得到 $s'(lr', ll', rr', rl')$ 。

当 $lr=0, rr=rl$ 时， $lr'=ll'=rr'=rl'=0$ 。

当 $lr=0, rr \neq rl$ 时， $lr'=ll'=0, rr'=son(rr), rl'=rl$ 。（ $son(rr)$ 表示在 $rl-rr$ 路径上 rr 的儿子）

当 $ll=lr \neq 0$ 时， $ll'=lr'=0, rr'=rr, rl'=rl$ 。

当 $ll \neq 0, ll \neq lr$ 时， $lr'=lr, ll'=f(ll), rr'=rr, rl'=rl$ 。（ $f(ll)$ 表示 ll 的父亲）

对于操作：输出已知串 $s(lr, ll, rr, rl)$ 的左端结点。当 $lr=0$ 时，输出结点 rr 的值。
 当 $lr \neq 0$ 时，输出结点 ll 的值。

右侧的操作完全类似。

算法证明

一个明显的事实：本题中所有得到的串都能分为左右两部分，左边部分的元素全都是在一系列(i)操作中从左端插入的，右边部分的元素全都是在一系列(i)操作中从右端插入的。

这几乎是显然的，算法正确性可以由此推出。

算法实现

算法实现的关键在于高效的在字母树中实现 son 函数。我的程序中使用了这样一个结构，我称他为 Super_Father。即，设 $SF(x,1)=f(x)$ ， $SF(x,k+1)=f(SF(x,k))$ 。我们在树中每个 x 上储存： $d(x)$ —— x 的深度； $Su(x,k)$ —— $SF(x,2^k)$ 。

这样，可以在 $O(\log n)$ 的时间内计算 $SF(x,k)$ ，注意到 $son(root)=SF(leave,d(leave)-d(root)-1)$ ，son 也可以在 $O(\log n)$ 的时间内给出。而维护 Su 也只需要 $O(\log n)$ 的时间就可以了。故，每次调用库的时间复杂度均为 $O(\log n)$ 。

题目来源

浙江 2007 年省选 捉迷藏

原题

Jiajia 和 Wind 是一对恩爱的夫妻，并且他们有很多孩子。某天，Jiajia、Wind 和孩子们决定在家里玩捉迷藏游戏。他们的家很大且构造很奇特，由 N 个屋子和 $N-1$ 条双向走廊组成，这 $N-1$ 条走廊的分布使得任意两个屋子都互相可达。

游戏是这样进行的，孩子们负责躲藏，Jiajia 负责找，而 Wind 负责操纵这 N 个屋子的灯。在起初的时候，所有的灯都没有被打开。每一次，孩子们只会躲藏在没有开灯的房间中，但是为了增加刺激性，孩子们会要求打开某个房间的电灯或者关闭某个房间的电灯。为了评估某一次游戏的复杂性，Jiajia 希望知道可能的最远的两个孩子的距离（即最远的两个关灯房间的距离）。

我们将以如下形式定义每一种操作：

| | |
|--------------|---------------------------------------|
| C(hange) i | 改变第 i 个房间的照明状态，若原来打开，则关闭；若原来关闭，则打开。 |
| G(ame) | 开始一次游戏，查询最远的两个关灯房间的距离。 |

【输入文件】

输入文件 `hide.in` 第一行包含一个整数 N ，表示房间的个数，房间将被编号为 $1,2,3,\dots,N$ 的整数。接下来 $N-1$ 行每行两个整数 a, b ，表示房间 a 与房间 b 之间有一条走廊相连。接下来一行包含一个整数 Q ，表示操作次数。接着 Q 行，每行一个操作，如上文所示。

【输出文件】

对于每一个操作 **Game**，输出一个非负整数到 `hide.out`，表示最远的两个关灯房间的距离。若只有一个房间是关着灯的，输出 0；若所有房间的灯都开着，输出 -1。

【样例输入】

8
1 2
2 3
3 4
3 5

3 6
6 7
6 8
7
G
C 1
G
C 2
G
C 1
G

【样例输出】

4
3
3
4

【数据规模】

对于 20%的数据, $N \leq 50, M \leq 100$;
对于 60%的数据, $N \leq 3000, M \leq 10000$;
对于 100%的数据, $N \leq 100000, M \leq 500000$ 。

问题描述

给定一棵树, 每个节点要么是黑色, 要么是白色, 能执行两个操作: 把某一个点取反色, 返回距离最远的黑色点对。

核心算法

首先, 我们定义一种对一棵树的括号编码。这种编码方式很简单, 所以, 我们就不给出严格的定义, 用以下这棵树为例:

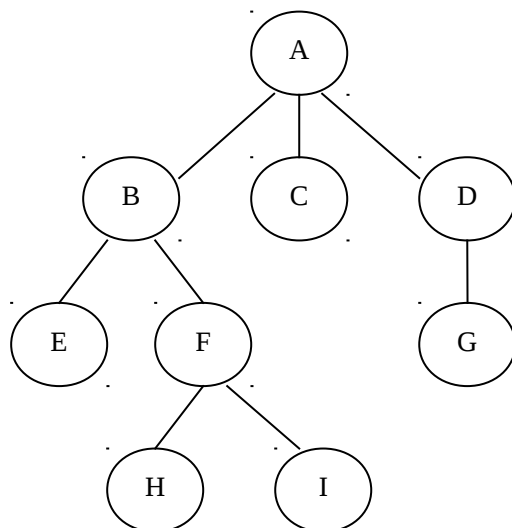


图 6 一棵有根树

可以先序遍历后写成: $[A[B[E][F[H][I]][C][D[G]]]$

去掉字母后的串: $[[][[[]]][][[[]]]$ 就称为这棵树的括号编码。

那么, 考察两个结点, 如 E 和 G, 取出介于它们之间的那段括号编码: $[[][[[]]][]$

把匹配的括号去掉 (用小括号和大括号表示), 得到: $\{()()()\}$

我们看到 2 个 $\}$ 和 2 个 $\{$, 也就是说从 E 向上爬 2 步, 再向下走 2 步就到了 G。

注意到, 我们只需要返回这棵树中点与点的距离, 所以, 贮存匹配的括号是没有意义的。因此, 对于介于两个节点间的一段括号编码 S, 可以用一个二元组 (a,b) 描述它, 即这段编码去掉匹配括号后有 a 个 $\}$ 和 b 个 $\{$ 。

所以, 对于两个点 PQ, 如果介于某两点 PQ 之间编码 S 可表示为 (a,b) , PQ 之间的距离就是 $a+b$ 。

要解决原问题, 我们比较自然的想到使用线段树处理这棵树的括号编码。

对于每个线段数中的节点：编码段 S ，我们储存以下量：

dis : $\max\{a+b \mid S'(a,b) \text{ 是 } S \text{ 的一个子串, 且 } S' \text{ 介于两个黑点之间}\}$

right_plus : $\max\{a+b \mid S'(a,b) \text{ 是 } S \text{ 的一个后缀, 且 } S' \text{ 紧接在一个黑点之后}\}$

right_minus : $\max\{a-b \mid S'(a,b) \text{ 是 } S \text{ 的一个后缀, 且 } S' \text{ 紧接在一个黑点之后}\}$

left_plus : $\max\{a+b \mid S'(a,b) \text{ 是 } S \text{ 的一个前缀, 且有一个黑点紧接在 } S \text{ 之后}\}$

left_minus : $\max\{b-a \mid S'(a,b) \text{ 是 } S \text{ 的一个前缀, 且有一个黑点紧接在 } S \text{ 之后}\}$

这样，我们可以根据儿子结点计算父结点，对于结点 $S=S_1+S_2$ ，其中 $S_1(a,b)$ 、 $S_2(c,d)$ 是 $S(e,f)$ 的儿子。

当 $b \geq c$ 时 $(e,f)=(a,b-c+d)$ ，当 $b < c$ 时 $(e,f)=(a-b+c,d)$ ①

$\text{dis}(S)=\max\{\text{dis}(S_1), \text{dis}(S_2), \text{right_plus}(S_1)+\text{left_minus}(S_2), \text{right_minus}(S_1)+\text{left_plus}(S_2)\}$ ②

$\text{right_plus}(S)=\max\{\text{right_plus}(S_1)-c+d, \text{right_minus}(S_1)+c+d, \text{right_plus}(S_2)$ ③

$\text{right_minus}(S)=\max\{\text{right_minus}(S_1)+c-d, \text{right_minus}(S_2)\}$ ④

$\text{left_plus}(S)=\max\{\text{left_plus}(S_2)-b+a, \text{left_minus}(S_2)+b+a, \text{left_plus}(S_1)\}$ ⑤

$\text{left_minus}(S)=\max\{\text{left_minus}(S_2)+b-a, \text{left_minus}(S_1)\}$ ⑥

最后，每次改变树中某结点状态，只须自底向上用儿子更新父亲，调整所有线段树中的相关节点。返回最远的两个黑色点之间的距离，只须输出 $\text{dis}(S^*)$ ，其中 S^* 是整棵树的编码。故整个算法的复杂度为 $(m \log n)$ (m 为操作次数)

算法证明

要证明算法的正确性，只需要证明①②③④⑤⑥是正确的即可。

先证明一个基本的结论：对于两段括号编码 $S_1(a_1, b_1)$ 和 $S_2(a_2, b_2)$ ，如果它们连接起来形成 $S(a, b)$ ，则：

当 $a_2 < b_1$ 时 $(a, b) = (a_1 - b_1 + a_2, b_2)$ ，当 $a_2 \geq b_1$ 时 $(a, b) = (a_1, b_1 - a_2 + b_2)$ 。 (*)

这是因为， S_1 、 S_2 相连时形成 $\min\{b, c\}$ 对成对的括号，合并后他们会被抵消掉。所以，当 $a_2 < b_1$ 时第一段[就被消完了，两段]连在一起，例如：

]] [[+]]] [[=]](0) [[=]]] [[

当 $a_2 \geq b_1$ 时第二段]就被消完了，两段[连在一起，例如：

]] [[[+]]] [[=]](0) [[=]]] [[

从而 (*) 式得证。(*) 式其实就是①式，故①得证。

这样，我们得到几个简单的推论：

$$(iv) \quad a+b=a_1+b_2+|a_2-b_1|=\max\{a_1+b_2+a_2-b_1, a_1+b_2-a_2+b_1\}$$

$$(v) \quad a-b=a_1-b_1+a_2-b_2$$

$$(vi) \quad b-a=b_2-a_2+b_1-a_1$$

接下来我们证明②。用 $S_1(a_1, b_1)$ 、 $S_2(a_2, b_2)$ 表示线段树中 $S(a, b)$ 的儿子。

对于 dis ，即 $\max\{a'+b' | S'(a', b') \text{ 是 } S \text{ 的一个子串，且 } S' \text{ 介于两个黑点之间}\}$ ，我们要对 S' 分类讨论。

情况 1: S' 是 S_1 的子串，这种情况下 $\max\{a'+b'\}=\text{dis}(S_1)$

情况 2: S' 是 S_2 的子串，这种情况下 $\max\{a'+b'\}=\text{dis}(S_2)$

情况 3: S' 具有形式 $S1'+S2'$, 其中 $S1'$ ($a1',b1'$) 是 $S1$ 的后缀, $S2'$ ($a2',b2'$) 是 $S2$ 的前缀。这种情况下, 由 (i) 式, $\max\{a'+b'\}=\max\{\max\{a1'+b2'+a2'-b1',a1'+b2'-a2'+b1'\}|$
 $S1'(a1',b1'),S2'(a2',b2')\}=\max\{\max\{a1'+b2'+a2'-b1'\},\max\{a1'+b2'-$
 $a2'+b1'\}\}=\max\{\text{right_plus}(S1)+\text{left_minus}(S2), \text{right_minus}(S1)+\text{left_plus}(S2)\}$

dis 就是以上三个数的最大值, 故②式得证。

③④⑤⑥的证明是完全类似的, 利用(i)(ii)(iii)三式即可。

算法实现

实现中, 在编码串中加进结点标号使编程更方便, 对于底层结点, 如果对应字符是一个括号或者一个白点, 那么 right_plus right_minus left_plus left_minus dis 的值就都是 $-\text{maxlongint}$; 如果对应字符是一个黑点, 那么 right_plus right_minus left_plus left_minus 都是 0, dis 是 $-\text{maxlongint}$ 。

题目来源

ural 1557 Network Attack

原题

1557. Network attack

Time Limit: 2.0 second Memory Limit: 64 MB

In some computer company, Mouse Inc., there is very complicated network structure. There are a lot of branches in different countries, so the only way to communicate with each other is the Internet. And it's worth to say that interaction is the key to the popularity and success of the Mouse Inc.

The CEO of this company is interested now to figure out whether there is a way to attack and devastate whole structure. Only two hackers are capable to perpetrate such an outrage — Vasya and Petya, who can destroy any two channels. If after that there are at least two servers without connection between them, then they succeed.

In other words, the company is a set of servers, some of them connected with bidirectional channels. It's guaranteed that all the servers are connected directly or indirectly. The hackers' goal is to divide network into at least two parts without any connection between them. Each hacker can destroy exactly one channel. And they can't destroy the same channel together. You are asked to count the number of ways for hackers to win.

Input

There are two integer numbers (N, M) in the first line of input: the number of servers and channels respectively ($1 \leq N \leq 2000$; $0 \leq M \leq 100000$). In the each of the next M lines there are exactly two numbers — the indices of servers connected by channel. Channels can connect a server to itself. There can be multiple channels between one pair of servers. The servers are numbered from 1 to N.

Output

There must be exactly one integer — the answer to the question described in the problem.

Sample

| input | output |
|-------|--------|
| 3 3 | 3 |

| | |
|-----|--|
| 1 2 | |
| 2 3 | |
| 3 1 | |

Problem Source: Novosibirsk SU Contest. Petrozavodsk training camp, September 2007

问题描述

给定一个无向连通图，若从中删去两条边能使他不连通，求所有这样的方案的总数。图点数 n 边数 m 。

核心算法

算法的主体流程是：

(i) 求出所有这样的边：删去他后图不连通（我们称这样的边为 **bridge**）。设有 s 条。

(ii) 缩去所有 **bridge**，即将 **bridge** 连接的两点并成一个。

(iii) 在修改后的图（此时图中已不含 **bridge**）中找 2-cuts，即两边组 $(E1, E2)$ 删去他们后图不连通。设有 r 对 2-cuts。

(iv) 输出 $r + s*(m-s) + s*(s-1)/2$ 就是答案。

显然，(i)(ii)(iv)都是平凡的，下面我们描述(iii)的算法。

设 n 个点为 $P_1 P_2 P_3 \dots P_n$ 。则任取一点，不妨设为 P_1 ，我们构建以 P_1 为

根的 DFS 树。设 $T(i)$ 表示以 P_i 为根的子树中结点构成的集合， $R(i)$ 表示 P_i 的所

有祖先构成的集合。深度 $d(i)$ 表示结点 P_i 的深度，其中规定 $d(1)=1$, $d(i)=d(j)+1$ 若 j 是 i 的父亲。 $U(A,B)$ 表示点集 A 中的点与点集 B 中的点所连边构成的集合。 $W(A,B)$ 表示点集 A 中的点与点集 B 中的点所连回边构成的集合。 $s(i)$ 表示 $T(i)$ 与 $R(i)$ 之间连的边数，即 $s(i)=|U(T(i),R(i))|$ 。 $t(i)$ 表示 $W(T(i),R(i))$ 中，相关深度最大的 $R(i)$ 节点的深度。

为解释上面这些定义我们以下图为例：

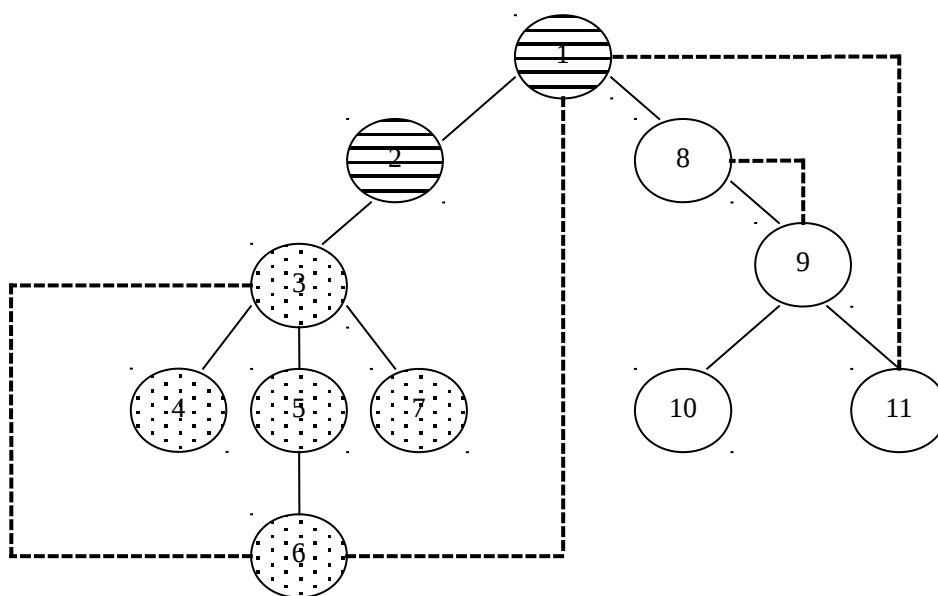


图 7

浅色阴影的的点构成了 $T(3)$ ，横条阴影的点构成了 $R(3)$, $d(8)=2$, $d(5)=4$, $s(5)=3$, $s(8)=2$, $t(3)=1$, $t(9)=2$, P_8P_9 之间连了一条树边一条回边。

r 初值 0。若由某个点 P_i 满足 $s(i)=2$, 则 r 加 1。若某个点对 P_i P_j

满足 P_i 是 P_j 的祖先, 且 $W(R(i), T(i)) = W(R(j), T(j))$, 则 r 加 1。

算法证明

只需要证明, 2-cuts 有且只有两种情况:

(i) $s(i)=2$ 时, $U(T(i), R(i))$ 中的两条边。

(ii) P_i 是 P_j 的祖先, 且 $W(R(i), T(i)) = W(R(j), T(j))$ 时, P_i

P_j 与各自父亲所连的边。

先说明无向图的一个重要性质: 无向图不存在 DFS 遍历树的横向边。这应该是熟知的。

利用这一性质我们可以推出一个基本的事实: (分割后的两部分中) 不包含根 (即 P_1) 的那部分, 一定具有形式: $T(i)$ 或 $T(i) \setminus T(j)$ (其中 P_i 是

P_j 的祖先)。(证明很简单, 这里略去)

若他的形式为 $T(i)$, 则 2-cuts 一定是 $U(T(i), T(1) \setminus T(i))$ 。由前面无向图的性质, $U(T(i), T(1) \setminus T(i)) = U(T(i), R(i))$ 。这就是情况(i), 显然这时必需 $s(i)=2$ 。

若他的形式为 $T(i) \setminus T(j)$, 则 2-cuts 一定是

$U(T(i), T(1) \setminus T(i)) \cup U(T(j), T(i) \setminus T(j))$ ，注意到 P_i 与 P_j 与各自父亲所连的树边，一定属于上面这个集合。所以 2-cuts 必须只有这两条边。所以 $W(T(i), T(1) \setminus T(i)) \cup W(T(j), T(i) \setminus T(j)) = \emptyset$ ，所以 $W(R(i), T(i)) = W(R(j), T(j))$ 。

算法实现

步骤(i)的实现：做一次 DFS，当某一棵子树 $T(i)$ 的遍历全部结束后，利用他的子树的 s 值，统计 $s(i)$ 。若 $s(i)=1$ ，则 P_i 与他的父亲的连边就是我们要找的一条边。

步骤(iii)的实现：

建立辅助数组 $l(i,j)=|U(T(i), D(j))|$ ，其中 $D(j)=\{P_i | d(i)=j\}$ 。

r 初值 0。对这棵树做一次 DFS，当某一棵子树 $T(i)$ 的遍历全部结束后，进行一下操作：

(i) 利用关于 P_i 儿子的 l 值，统计 $l(i,1), l(i,2) \dots l(i,d(i)-1)$ 。

(ii) 利用 l 数组统计 $s(i)$ 和 $t(i)$ 。

(iii) 若 $s(i)=2$ ，则 r 加 1。

(iv) (若 P_i 与他的父亲只有一条连边则做这一操作，否则不做) 扫描

$T(i)$ 中非 P_i 的节点，若有某点 P_j 满足， P_j 与他的父亲只有一条连边、

$s(j)=s(i) \setminus t(j) < d(i)$, 则 r 加 1。(这三个条件, 其实就是 $W(R(i), T(i))=W(R(j), T(j))$ 的判断)

整个算法中用邻接矩阵存储所有的边, 整个时间复杂度为 $O(n^2+m)$

题目来源

ural 1569 Networking the “Iset”

原题

1569. Networking the “Iset”

Time Limit: 1.0 second Memory Limit: 64 MB

There’s not much time left until the “Iset” tower opens for business, but a computer network is not yet installed in the building. The network is expected to be very robust and should have lots of branches. There are N nodes in the tower that should be connected with this network. These nodes were planned to be connected with M direct links, with no more than one direct link between each pair of nodes. To save some time, it was decided that only the links that are required to make a connected network will be installed; all the remaining wires are going to be laid after the opening ceremony. In order to be efficient network should have satisfy one more requirement: the maximal distance between it’s nodes must be as small as possible. Distance between a pair of nodes A and B is defined as the number of intermediate nodes on the path from the node A to the node B .

Input

The first line contains two integers N ($2 \leq N \leq 100$) and M ($1 \leq M \leq 10000$). The following M lines describe the initial planned network layout. Each of these lines contains a pair of integers — numbers of nodes that are connected with a direct link. Nodes are numbered from 1 to N . This network layout is guaranteed to be connected, and there are no links connecting a node with itself.

Output

The new network layout (in the same format).

Sample

| input | output |
|-------|--------|
| 4 4 | 1 2 |
| 1 2 | 2 3 |
| 2 3 | 2 4 |
| 2 4 | |
| 3 4 | |

Problem Author: Sergey Pupyrev

Problem Source: The XIIth USU Programing Championship, October 6, 2007

问题描述

输入一个图 $G=(V,E)$ ，求这个图的直径最小生成树。（树的直径：距离最远的两点之间的路径）

核心算法

对于每个图中的 V_i ，求出以他为根的 BFS 树 T_i ，设深度为 d_i 。由于这棵 BFS 树是不唯一的，我们优先考虑满足条件 X 的那一棵树。

条件 X 成立，当且仅当，存在一个满足 $d(V_i, V_j)=1$ 点 V_j ，使得，对于任意满足 $d(V_i, V_k)=d_i$ 的点 V_k ，都有 V_j 是 V_k 的祖先。

在规定当条件 X 不成立， $D_i=2d_i$ ；当条件 X 成立， $D_i=2d_i - 1$ 。

我们取出使得 D_i 最大的点 V_i 的 BFS 树 T_i ，就是答案。

算法证明

为了叙述的方便，引入下面定义： $l(v)=\max\{d(u,v)|u,v \text{ 是一个图中的点}\}$ 。树的中心：树中 $l(v)$ 最小的点。

这里指出关于 BFS 树的两条性质：一个图中以某一定点为根的 BFS 树不一定唯一的，但以某一定点为根的 BFS 的深度都相等。这些性质是很显然的，不再证明。

尽管一棵树的直径是不唯一的（直径的长是确定），但对于树的中心有两个结论：当直径长为偶数，树的中心是唯一；当树的直径长为奇数，树的中心是唯二的。

下面证明这两个结论：

当树的直径 D 是偶数。设有一条直径是 AB ， AB 中点是 P 。一方面，对于任意一个点 C ，设 AB 上距离 C 最近的点为 Q ，不妨 Q 在 AP 上，则 $CP=BC-BP \leq AB-BP = \frac{D}{2}$ ，同时 $AP=BP = \frac{D}{2}$ ，所以 $l(P) = \frac{D}{2}$ 。另一方面，对于任意一个不是 P 的点 C ，设 AB 上距离 C 最近的点为 Q ，不妨 Q 在 AP 上，则 $BC=BP+PQ+QC > \frac{D}{2}$ ，所以 $l(C) > \frac{D}{2}$ 。所以， P 就是这棵树的唯一的中心。

当树的直径 D 是奇数。设有一条直径是 AB ， AB 中央两点是 PQ ，其中 A 在 P 一侧， B 在 Q 一侧。一方面，对于任意一个点 C ，设 AB 上距离 C 最近的点为 R ，若 R 在 AP 上，则 $CP=BC-BP \leq AB-BP = \frac{D-1}{2}$ ，若 R 在 BQ 上，则 $CP=AC-AP \leq AB-AP = \frac{D-1}{2}$ 。同时 $BP=PQ = \frac{D-1}{2}$ ，所以 $l(P) = \frac{D-1}{2}$ 。同理 $l(Q) = \frac{D-1}{2}$ 。另一方面，对于任意一个不是 P 、 Q 的点 C ，设 AB 上距离 C 最近的点为 R ，不妨 R 在 AP 上，则 $BC=BP+PR+RC = \frac{D-1}{2} + PR + RC > \frac{D-1}{2}$ ，所以 $l(C) > \frac{D-1}{2}$ 。所以， P 、 Q 就是这棵树的唯二的中心。

设 T 是图的某一棵直径最小生成树。接下去我们证明：当 T 的直径为偶数，且 P 是它的唯一的中心，图中任意一棵以 P 为根的 BFS 树都是该图的一棵直径最小生成树。当 T 的直径为奇数，且 PQ 是它的唯二的中心，则存在至少一棵以 P 为根的 BFS 树满足 X 条件，且所有满足 X 条件以 P 为根的 BFS 树都是该图的一棵直径最小生成树。

先证明 T 的直径是偶数的情况。设 AB 是 T 的一条直径。有前面的证明对任意点 C 有 $CP \leq AP$ 且当 $C=A$ 或 B 是这个等号能取到。所以， $l(P) = AP$ 。注意到， $l(P)$ 就是以 P 为根的 BFS 树的深度。所以在一棵以 P 为根的 BFS 树 T' 中，任意两点之间的距离不超过 $2 \cdot l(P)$ ，即 $2 \cdot AP = T$ 的直径。所以 T' 的直径都不超过 T 的直径，即 T' 一定是该图的一棵直径最小生成树。

再证明 T 的直径是奇数的情况。设 AB 是 T 的一条直径， AB 长为 D 。其中 A 在 P 一侧， B 在 Q 一侧。类似的偶数部分的推理，可以得到 $l(P)=BP=\frac{D+1}{2}$ 。由先前的证明，对于任意 C ，若 $PC=\frac{D+1}{2}$ 则 $QC=\frac{D-1}{2}=l(P)-1$ ，而 $PQ=1$ ，所以 $PC=PQ+QC$ 。又因为 $l(P)$ 就是以 P 为根的 BFS 树的深度，所以一定存在一棵以 P 为根的 BFS 树 T'' ，使得 T'' 中所有满足 $PC=l(P)$ 的点 C ，都是 Q 的子孙。那么 T'' 是符合 X 条件的。另一方面，对于任意一棵符合 X 条件的以 P 为根的 BFS 树 T_0 ，设所有满足 $PC=l(P)$ 的点 C ，都是 Q_0 的子孙。那么显然 T_0 的直径为 $2l(P)-1=D$ ，且 P 和 Q_0 是它的唯二的中心。证毕。

所以算法中的 D_i 实际就是以 i 点为中心的生成树的最小直径。

算法实现

用 Floyd 算法与处理所有两点间的距离。总算法复杂度是 $O(n^3)$ 。