取出对应的 $seg(T,x)$ 值，并适当截取。由于保证最终答案也是一个等差数列，只需要简单地对截取过后的 $seg(T,x)$ 直接合并。

这样，我们成功地把算法中用到的 *succ* 和 *pred* 查询替代为在 *seg* 的哈希表中的每次期望 $O(1)$ 查询。

综合前面的分析，子串周期查询问题做到了期望 $O(n\log n)$ 的时间空间预处理，一次查询期望 $O(\log n)$ 的复杂度。

5 基本子串字典的扩展与应用

本节主要介绍基本子串字典的一些扩展和应用。由于基本子串字典和后缀数组具有很多联系，也可以看作后缀数组向基本子串，即长度为 2 的幂次的子串方向的应用。

5.1 子串比较

基本子串字典的一个基础应用，是可以 $O(1)$ 比较两个子串的大小。尽管这个问题可以被后缀数组、高度数组和区间最值查询 (RMQ) 做到相同的复杂度，甚至可以做到更优的 $O(|S|)$ 预处理，但是基本子串字典的做法简单，在我看来有一定的启发性，而且可扩展性要强于 RMQ 做法。

5.1.1 问题描述

给定串 S ，每次查询给出 i,j,i',j' ，要求比较 $S[i\dots j]$ 和 $S[i'\dots j']$ 的大小。

5.1.2 做法

如果 $j-i\neq j'-i'$ ，那么两个子串不可能相等，可以把较长的子串的后面若干位删去使得和较短的子串长度相同，并做一次新询问。若新询问的回答是不相等，则原询问的回答和新询问的回答相同，否则原询问的回答是较长的子串更大。

现在假设 $j-i=j'-i'$ 。找到正整数 m 满足 $2^m<j-i+1\le 2^{m+1}$ ，那么两个子串的比较可以由比较 $S[i\dots i+2^m-1]$ 与 $S[i'\dots i'+2^m-1]$ ，以及比较 $S[j-2^m+1\dots j]$ 与 $S[j'-2^m+1\dots j']$ 完成。然而前两者可以通过比较 $Name_t[i]$ 与 $Name_t[j]$ 和比较 $Name_t[j-2^m+1]$ 与 $Name_t[j'-2^m+1]$ 完成。

综上所述，得到了一个 $O(n\log n)$ 预处理，$O(1)$ 查询的做法。

5.1.3 简单扩展

把原问题改为：给定一棵大小为 n 的 trie 树，定义 $up(i,l)$ 表示 i 节点向上走 l 条边，并把每条边上的字符依次连接形成的字符串。每次询问给出 i,l,i',l' ，比较 $up(i,l)$ 和 $up(i',l')$ 的大小。

此时倍增求后缀数组的算法仍然适用，但是线性求高度数组的算法却失效了。一般遇到这种情况，只能使用常数较大的二分 + 字符串哈希做法，或是较难理解的广义后缀树（后缀自动机）。

然而，基于基本子串字典的做法仍然有效，甚至还可以和 RMQ 做法结合，得到新的求高度数组的方法：由于二分 + 哈希中的哈希仅仅是用来判断字符串是否相等的，可以直接把其替换为基于基本子串字典的字符串比较，得到更优的常数。

当然，可以发现上述所有算法不可避免地需要快速求某个点 x 的 k 级祖先 $anc(x,k)$ 。记录 $f(i,k)=anc(i,2^k)$ ，并用转移式 $f(i,k)=f(f(i,k-1),k-1)$ 递推得到所有的 f 值，这样 anc 查询可以在 $O(\log n)$ 时间内解决，不过这样每次询问的复杂度增大为 $O(\log n)$ 。要做到 $O(1)$ 查询，还需要结合长链剖分 +ladder 的做法，具体做法可参考相关资料。

5.2 整周期相关

在一些有关整周期的问题中，幂串 (String Powers) 是一个重要的研究对象。

定义 6：如果一个字符串 T 有周期 $\frac{|T|}{k},k\in\mathbb{Z},k>1$ ，则称 T 是一个 k 次幂串。此时 T 可以表示为 $\left[pref(T,\frac{|T|}{k})\right]^k$ 。若还满足 $k\cdot minper(T)=|T|$ ，则称 T 是一个本原 k 次幂串。特别地，当 $k=2$ 时，T 是平方串（本原平方串），当 $k=3$ 时，T 是立方串（本原立方串）。如果没有满足条件的 k ，则称 T 是本原串。

5.2.1 整周期的简单性质

性质 1：如果 T 既是 k_1 次幂串，又是 k_2 次幂串，则 T 是 $\frac{k1\cdot k2}{\gcd(k_1,k_2)}$ 次幂串。

性质 2：T 是本原串当且仅当 T^k 是本原 k 次幂串。特别地，当 $k=2$ 时，T 是本原串当且仅当 T^2 中 T 只作为前缀和后缀出现。

这两个性质容易使用 WPL 证明。

5.2.2 k 次幂子串查找

给定一个字符串 S ，找到一个 S 的子串 $S[l\ldots r]$ 满足其是 k 次幂串。我们直接给出做法，并说明其是正确的。注意算法可能会给出多个相同的 k 次幂子串。

Algorithm 1 Detect k-th Powers

Require: S, k

Require: Function $IsPower_k(pos, root)$

Ensure: Report if S contains a k-th power

```
Name ← DBF(S)
for t = 0 to ⌊log2 |S|⌋ do
    Prev ← (0, 0, ... 0)
    for i = 1 to |S| − 2^t + 1 do
        name ← Name_t(i)
        pos ← Prev[name]
        root ← i − pos
        if IsPower_k(pos, root) then
            Report that S contains a k-th power S[pos ... pos + k · root − 1]
        end if
        Prev[name] ← j
    end for
end for
if No k-th power detected then
    Report that S does not contain a k-th power
end if
```

算法中的函数 $IsPower_k(pos, root)$ 返回 $S[pos \dots pos + k \cdot root - 1]$ 是否是一个 k 次幂串。这可以通过判断 $S[pos \dots pos + (k-1) \dots root - 1]$ 和 $S[pos + root - 1 \dots pos + k \cdot root - 1]$ 是否相等来解决，从而可以套用之前的 $O(1)$ 判断方法。

显然如果算法回答存在 k 次幂子串则一定是正确的，我们只需说明如果算法回答不存在，则 S 中一定没有 k 次幂子串。

定理 1：当 $k = 2$ 时，若 S 存在平方子串，则 **Algorithm 1** 一定会找到所有长度最短的平方子串。

引理 6：若 $w = S[i \dots j] = S[i' \dots j']$，且区间 $[i, j]$ 和 $[i', j']$ 的交非空，则 S 中存在一个长度不超过 $2|w| - 1$ 的平方串。

证明：令 $l = \min(i, i'), r = \max(j, j')$。考虑串 $S[l \dots r]$，w 是它的 border，$r - l + 1 - |w|$ 是它的周期。而 $r - l + 1 < 2|w|$，因此 $S[l \dots l - 1 + 2 \cdot (r - l + 1 - |w|)]$ 是一个平方串。 □

定理 1 的证明：令 $S[j \dots j + 2 \cdot r - 1]$ 为某个长度最短的平方子串。令 $v = S[j \dots j + r - 1]$，找到 s 满足 $2^s \le |v| < 2^{s+1}$，那么考虑 **Algorithm 1** 运行到 $t = s, i = j + r$ 时的情况。由于 $S[j \dots j + 2^s - 1] = S[i \dots i + 2^s - 1]$，可以得到 $i > pos \ge j$，因此只需证明 $pos = j$ 即可。

令 $w = S[i \dots i + 2^s - 1]$。假设 $pos > j$，则有 $S[i \dots i + 2^s - 1] = S[pos \dots pos + 2^s - 1] = S[j \dots j + 2^s - 1] = w$。由于 $j - i = |v| < 2^{s+1} = 2|w|$，区间 $[pos, pos + 2^s - 1]$ 必定和 $[i, i + 2^s - 1]$ 与 $[j, j + 2^s - 1]$ 其中之一有交。根据**引理 6** 就得到了一个长度比 $2|v|$ 更短的平方串，产生了矛盾。 □

定理 2：当 $k \ge 3$ 时，**Algorithm 1** 会找到 S 中的所有本原 k 次幂子串。

证明：令 $S[j \dots j + k \cdot r - 1]$ 为某个本原 k 次幂子串。令 $v = S[j \dots j + r - 1]$，找到 s 满足 $2^{s-1} < |v| \le 2^s$，那么考虑 **Algorithm 1** 运行到 $t = s, i = j + r$ 时的情况。由于 $S[j \dots j + 2^s - 1] = S[i \dots i + 2^s - 1]$，可以得到 $i > pos \ge j$，因此只需证明 $pos = j$ 即可。

考虑 $S[j \dots j + 2 \cdot r - 1] = v^2$，若 $i > pos > j$ 则 v 在 pos 位置再次出现，根据**性质 1** 和**性质 2**，v 不是本原串，与 v^k 是本原 k 次幂串矛盾。 □

综合**定理 1** 和**定理 2**，我们就证明了 **Algorithm 1** 的正确性。

5.2.3 Runs 和 Cubic Runs

与整周期有关的另一个重要概念是 Runs 的概念。

定义 7：字符串 S 的一个 Run 是一个三元组 (i, j, l) 满足 $l \in per(S[i \dots j])$，$l \le \frac{j-i+1}{2}$，且 $S[i-1 \dots j]$ 和 $S[i \dots j+1]$ 均未定义或不满足前述条件。若 $l \le \frac{j-i+1}{3}$，则三元组 (i, j, p) 被称为一个 Cubic Run。

根据 WPL 也不难得到下述结论：字符串 S 的所有 Run 对应的区间 $[i, j]$ 没有包含关系。

可以把 Run 理解成至少重复两次，且无法向左右扩展的周期子串。下面的重要定理说明了 Run 个数的上界。定理的证明可以参考相关资料。

定理 3：(The Runs Theorem) 一个字符串 S 至多有 $|S|$ 个 Run。

求字符串 S 的所有 Runs 的最优复杂度是 $O(|S|)$，不过需要使用后缀数组的线性构造，以及基于 Method of Four Russians 的、线性预处理 $O(1)$ 查询的 RMQ，因而十分繁琐，不适合在算法竞赛中实现。这个问题有较为容易理解且易于实现的 $O(|S|\log|S|)$ 的做法，有兴趣的读者可以阅读参考文献。

在一些问题中，我们关心的并不是所有的 Runs，而是具有更强性质的 Cubic Runs。由于 Cubic Runs 是 Runs 的子集，所以 Cubic Runs 的个数也至多是 $|S|$。不难发现，任何一个本原立方串都恰好被一个周期与其相等的 Cubic Run 包含，且一个 Cubic Run 一定包含至少一个本原立方串。这提示我们可以通过求出本原立方串来解决这类问题。我们不加证明地给出下面一个定理，这使得我们可以直接用 **Algorithm 1** 来求出所有的本原立方串。

定理 4： **Algorithm 1** 中求出的所有 k 次幂串均是本原的。

求出了所有的本原立方串，就容易得到所有 Cubic Runs 了，只要每次取出出现位置最早的本原立方串，并尽量扩展得到一个 Cubic Run 即可。

6 总结

在本文中，我们完整地解决了子串周期查询问题这一经典问题。问题虽然解决了，但是它的做法，以及所用到的三个引理和基本子串字典这个数据结构，还值得更多的研究。

希望本文能起到一个抛砖引玉的作用，带领读者初步领略有关字符串周期的魅力，并更加深入地了解不仅限于本文的相关知识和研究成果。

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张瑞喆的指导。
感谢南京外国语学校李曙老师的指导和支持。
感谢杨骏昭和王泽远同学帮我审稿。
感谢所有曾经给予我帮助的老师和同学。
感谢我的父母一直以来的全力支持和鼓励。

参考文献

[1] 金策，2017 年 CCFNOI 冬令营《字符串算法选讲》

[2] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń. Efficient Data Structure for the Factor Periodicity Problem

[3] Maxime Crochemore, Costas Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Krzysztof Stencel, Tomasz Waleń. New Simple Efficient Algorithms Computing Powers and Runs in Strings

[4] Manber Udi, Myers Gene. Suffix Arrays: a new method for on-line string searches

[5] Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, Kazuya Tsuruta. The "Runs" Theorem

[6] 徐明宽，IOI2018 中国国家候选队论文《非常规大小分块算法初探》

《公园》命题报告

福建省福州第一中学 吴作同

摘 要

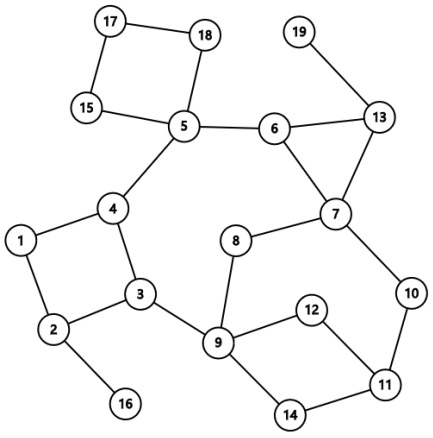
《公园》是我在 2019 年集训队互测中命制的一道试题。本文引入了“广义串并联图”的概念，并证明了任意广义串并联图可以通过“删 1 度点操作”、“缩 2 度点操作”、“叠合重边操作”使图转化为只包含一个节点的图。本文介绍了本题在不同的特殊条件下的解决思路与算法选择，其中正解算法为了支持快速修改参数，建立了表达式树来表示本题中广义串并联图的动态规划的过程，使用矩阵的形式表示转移，并用链分治线段树来加速修改。我希望《公园》这道题能够带给大家更多关于广义串并联图以及缩 2 度点操作的相关思考。

1 试题

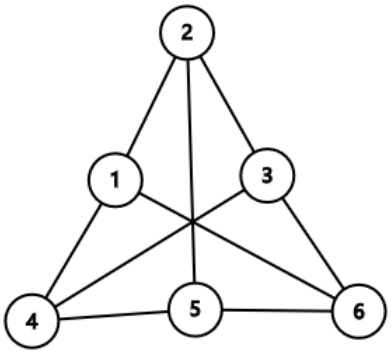
1.1 题目大意

我们称满足对于任意 4 个节点都不存在 6 条两两没有公共边的路径连接这 4 个节点中的每一对节点的无向连通图为广义串并联图。

下图是一张广义串并联图：



下图不是一张广义串并联图：



因为对于第 $1,4,5,6$ 号节点，存在六条两两没有公共边的路径 $1\to 4, 4\to 5, 5\to 6, 6\to 1, 4\to 3\to 6, 1\to 2\to 5$ 连接这四个节点的每一对节点。

给定一张 n 阶简单（无自环无重边）广义串并联图 $G=(V,E)$ 。每个节点有两种点权 b_i 和 w_i ，每条边有两种边权 s_i 和 d_i 。要求把每个节点染成黑白两色之一，求一种权值最大的染色方案的权值。定义染色之后节点 i 的颜色为 c_i ，即若 $c_i=0$ ，表示节点 i 为黑色；若 $c_i=1$ ，表示节点 i 为白色。一种染色方案的权值定义为

$$\sum_{v\in V}(b_v\cdot[c_v=0]+w_v\cdot[c_v=1])+\sum_{e=(u,v)\in E}(s_e\cdot[c_u=c_v]+d_e\cdot[c_u\neq c_v])$$

其中约定 $[x]$ 的含义为，若表达式 x 为真则 $[x]=1$ ，否则 $[x]=0$ 。

要求支持动态修改某个节点的两种点权或某条边的两种边权，在所有修改前与每次修改后输出答案。修改共 Q 次。

1.2 输入格式

第一行两个正整数 n,m ，其中 n 表示图 G 的节点数， m 表示图 G 的边数。

接下来 n 行，每行两个非负整数，其中第 $i(1\le i\le n)$ 行表示 b_i,w_i 。

接下来 m 行，每行四个正整数 x_i,y_i,s_i,d_i ，其中第 $i(1\le i\le m)$ 行表示第 i 条边的两个端点的编号为 x_i,y_i 且第 i 条边的两种边权为 s_i,d_i 。

第 $n+m+2$ 行一个非负整数 Q 。

接下来 Q 行每行三个正整数 x,a,b 表示一次修改，若 $x\le n$ 则表示把 b_x 改为 a ，把 w_x 改为 b ，否则表示把 s_{x-n} 改为 a ，把 d_{x-n} 改为 b 。

1.3 输出格式

输出 $Q+1$ 行，其中第 1 行表示所有修改之前问题的答案，第 $i(2\le i\le Q+1)$ 行表示第 $i-1$ 次修改后问题的答案。

1.4 样例输入

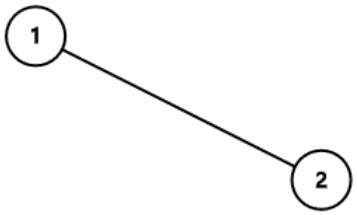
```
2 1
2 3
4 7
1 2 5 7
1
1 2 6
```

1.5 样例输出

```
16
18
```

1.6 样例解释

公园路线图如下图：



修改前，$b_1 = 2, w_1 = 3, b_2 = 4, w_2 = 7, s_1 = 5, d_1 = 7$ 。
最优染色方案为 $c_1 = 0, c_2 = 1$ ，权值为 $b_1 + w_2 + d_1 = 16$ 。
修改后，$b_1 = 2, w_1 = 6, b_2 = 4, w_2 = 7, s_1 = 5, d_1 = 7$ 。
最优染色方案为 $c_1 = 1, c_2 = 1$ ，权值为 $w_1 + w_2 + s_1 = 18$ 。

1.7 数据规模与约定

保证 $2 \le n \le 10^5, Q \le 10^5$ 。
保证 $x_i, y_i \le n$ ， $x \le n + m$ ， $b_i, w_i, s_i, d_i, a, b \le 10^6$ 。

子任务	分值	图的特殊形态	n	Q	其它限制
1	2	树		$=0$	
2	3		≤ 17	≤ 17	
3	7	仙人掌		$=0$	
4	5			$=0$	
5	13		≤ 1000	≤ 1000	
6	19	仙人掌			$b_i = w_i = 0, x > n$
7	17				
8	23	树			
9	11				

树即无环的连通图。

仙人掌即每条边属于不超过 1 个简单环的连通图。

留空部分表示没有额外限制。

1.8 时空限制

时间限制：6s

空间限制：1GB

2 初步分析

2.1 算法一

问题需要求出一种权值最大的染色方案。显然对于一种确定的染色方案，可以通过枚举边统计贡献的方式在 $O(m)$ 的时间内计算出染色方案的权值。

对于子任务 2 ，n, Q 都不超过 17 。对于一次询问，染色方案的情况总数较少，只有不超过 $2^{17} = 131072$ 种，所以可以考虑枚举所有的染色方案，分别求出权值后取最大值。

该算法时间复杂度为 $O(2^n \times Qm)$ 。题目中并没有给出 m 的取值范围，但是可以证明 m 是 $O(n)$ 级别的，证明见 5.3.1 。

期望得分 3 分。

该算法效率低下，是因为该算法没有利用到 G 是广义串并联图的性质，并且对于每一次修改都要重新计算问题的答案，没有利用到修改前后大量数据的相似性。

接下来考虑先从一些特殊的情况入手，来分析这个问题。

3 树上的问题

3.1 算法二 树，没有修改

对于此类树上最优化问题，通常可以使用树形动态规划的方式解决。

将无根树转为以 1 为根的有根树，记 $f(i,j)(j\in\{0,1\})$ 为 $c_i=j$ 时节点 i 子树的答案。

设 $C(u)$ 为 u 的子节点集合，则有

$$f(u,0)=b_u+\sum_{v\in C(u)}\max\{f(v,0)+s_{(u,v)},f(v,1)+d_{(u,v)}\}$$

$$f(u,1)=w_u+\sum_{v\in C(u)}\max\{f(v,0)+d_{(u,v)},f(v,1)+s_{(u,v)}\}$$

答案为 $\max\{f(1,0),f(1,1)\}$ 。

时间复杂度 $O(n)$ 。

可以通过子任务 1 ，期望得分 2 分。

3.2 算法三 树，带修改

考虑利用算法二的动态规划算法。

根据转移方程，可以发现只有被修改的节点/边到树的根的一条链上的节点的 f 值[43]可能会被修改。对于这类问题，可以考虑使用基于链的分治。

在树上，我们把度数为 1 的非根节点称为**叶子**或**叶子节点**，否则称为**非叶子节点**。

修改点的 f 值或修改边的父亲节点的 f 值可以直接计算，其余部分可以利用树链剖分[44] 拆成 $O(\log n)$ 条重链，然后只需考虑如何快速维护重链上的 f 值。

设 son_u 为节点 u 的重子节点。

[43]我们把一个节点 u 的 $f(u,0)$ 和 $f(u,1)$ 的值统称为节点 u 的 f 值。

[44]本文中所有提到的“树链剖分”为轻重链剖分，是摘要中提到的“链分治”的表现形式。具体地，定义点 u 的**重子节点**为 $C(u)$ 中满足 $size(v)$ 最大的 v ，若有多个则取编号最小的，其中 $size(v)$ 表示以 v 为根的子树中节点的个数，并把一个节点与其重子节点之间的连边称为**重边**，把树上不是重边的边称为**轻边**。定义**重链**为所有重边的边导出子图的一个连通块或是满足 u 与 u 的父亲节点之间的边为轻边的叶子节点 u 所构成的只包含一个节点的图。定义点 u 的**轻子节点**为 $C(u)$ 中不是 u 的重子节点的节点。

对于非叶子节点 u ，我们改写一下转移方程：

$$\begin{aligned} f(u,0) &= b_u + \sum_{v\in C(u)} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} \\ &= b_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} \\ &\quad + \max\{f(son_u,0)+s_{(u,son_u)}, f(son_u,1)+d_{(u,son_u)}\} \\ f(u,1) &= w_u + \sum_{v\in C(u)} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} \\ &= w_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} \\ &\quad + \max\{f(son_u,0)+d_{(u,son_u)}, f(son_u,1)+s_{(u,son_u)}\} \end{aligned}$$

使用已更新的 son_u 的 f 值更新 u 的 f 值时，只有最后一项中的 $f(son_u,0)$ 和 $f(son_u,1)$ 可能改变，其余都可视为常数。于是转移方程可以写成：

$$\begin{aligned} f(u,0) &= \max\{f(son_u,0)+A_u, f(son_u,1)+B_u\} \\ f(u,1) &= \max\{f(son_u,0)+C_u, f(son_u,1)+D_u\} \end{aligned}$$

其中

$$\begin{aligned} A_u &= b_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} + s_{(u,son_u)} \\ B_u &= b_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} + d_{(u,son_u)} \\ C_u &= w_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} + d_{(u,son_u)} \\ D_u &= w_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} + s_{(u,son_u)} \end{aligned}$$

于是重链上的转移可以用类似矩阵的形式来表示。

3.2.1 矩阵表示

令

$$f_u = \begin{bmatrix} f(u,0) \\ f(u,1) \end{bmatrix} = \begin{bmatrix} \max\{f(son_u,0)+A_u, f(son_u,1)+B_u\} \\ \max\{f(son_u,0)+C_u, f(son_u,1)+D_u\} \end{bmatrix}$$

观察表达式的形式，发现它很类似一个矩阵乘法的结果，只是原本矩阵乘法中元素的加法在这里为 max 运算，原本矩阵乘法中元素的乘法在这里为加法运算。

由于加法、max 运算满足交换律、结合律且加法对 max 运算满足分配律，所以这样新定义的矩阵乘法仍然满足结合律 [3]。

本题中所有矩阵乘法都在该定义下进行。

令

$$g_u = \begin{bmatrix} A_u & B_u \\ C_u & D_u \end{bmatrix}$$

则有

$$\begin{aligned} g_u f_{son_u} &= \begin{bmatrix} A_u & B_u \\ C_u & D_u \end{bmatrix} \begin{bmatrix} f(son_u, 0) \\ f(son_u, 1) \end{bmatrix} \\ &= \begin{bmatrix} \max\{f(son_u, 0) + A_u, f(son_u, 1) + B_u\} \\ \max\{f(son_u, 0) + C_u, f(son_u, 1) + D_u\} \end{bmatrix} \\ &= f_u \end{aligned}$$

3.2.2 分治

初始状态下，每个 A_u, B_u, C_u, D_u 的值，可以在预处理时利用算法二计算出。之后当且仅当修改 u 到某个子节点之间的边的边权，或修改 u 或 u 的某个轻子节点的子树内的点的点权时，A_u, B_u, C_u, D_u 的值才需要更新。由于一个点到根的简单路径上只有 $O(\log n)$ 条轻边，所以一次修改只有 $O(\log n)$ 个点的 A_u, B_u, C_u, D_u 被更新。但是为了更新一个点 u 的 A_u, B_u, C_u, D_u 的值或计算答案（即计算 $f(1,0)$ 与 $f(1,1)$ 的值），就需要计算出某一条重链的顶端节点[45]的 f 值。我们可以用线段树来维护该 f 值。

有了 3.2.1 中提到的的矩阵乘法以及结合律，就可以使用线段树来维护重链上一段区间内的矩阵 g 的乘积，在修改某个点 u 的 g_u 时，直接在对应的线段树上更新即可。

注意 g_u 对叶子节点 u 是没有定义的，所以建线段树时只需要维护非叶子节点的信息。

因此重链顶端节点 u 的 f 值可以按如下方法计算：

若 u 是叶子，则 $f_u = \begin{bmatrix} b_u & w_u \end{bmatrix}^T$ [46]，否则可以利用线段树维护的 g 矩阵计算。

设重链上的节点为 $u_1, u_2, \cdots, u_l$，其中 $u_1 = u$，且对于 $1 \le i < l$，u_i 是 u_{i+1} 的父亲节点，且 u_l 一定为叶子[47]。

那么有

$$f_{u_1} = g_{u_1} g_{u_2} \cdots g_{u_{l-1}} f_{u_l}$$

[45] 重链的顶端节点即重链上距离根最近的节点。底端节点即重链上距离根最远的节点。

[46] 符号 ^{T} 表示矩阵的转置。

[47] 根据树链剖分的方法可知，重链的底端一定是叶子。

于是直接在线段树上查询整条重链上的 g 矩阵的乘积，再右乘上 f_{u_l} 即可得到 f_u 的值。

总结一下，在一次修改中，这个算法的过程是这样的：

1、修改被修改的点或边的权值，令 u 为被修改边的父亲节点或被修改点。

2、如果 u 不是叶子，更新 g_u ，并更新对应重链上的线段树。

3、令 u 为原来 u 对应重链的顶端节点。

4、重新计算 f_u 的值并令 v 为 f_u 。

5、若 u 为根，返回 v 作为答案，否则令 u 为 u 的父亲节点。

6、利用计算好的 v 的值更新 g_u 。

7、跳转到第 3 步。

在一次修改中，需要进行 $O(\log n)$ 次线段树单点修改操作，所以该算法的时间复杂度为 $O(n+Q\log^2 n)$ 。

可以通过子任务 $1,8$ ，期望得分 25 分。

4 仙人掌上的问题

4.1 算法四 仙人掌，没有修改

对于仙人掌上的问题，一种经典的方法是仙人掌上的动态规划。

时间复杂度 $O(n)$ ，可以通过子任务 $1,3$ ，获得 9 分。

该部分内容与本文主题无关，因此不再赘述。相关内容可以参考陈俊锟的论文《〈神奇的子图〉命题报告及其拓展》。

子任务 6 有一个“所有点权均为 0 ”的限制。这个限制可以对解题有什么样的帮助呢？先分析一下树上的情况。

4.2 算法五 树，点权均为 0

当点权均为 0 时，染色方案的权值只和每条边两端的点的颜色是否相等有关。

我们称两端节点颜色相同的边为同色边，两端节点颜色不同的边为异色边。

有一种思路是，枚举每条边是同色边还是异色边，然后判断是否存在一种染色方案满足条件。

对于树上的情况，假设我们已经确定了每条边是同色边还是异色边，那么我们可以令 $c_1=0$ ，然后进行一次深度优先搜索。对于每个非根节点，可以根据父亲节点的颜色以及与父亲节点之间的边是同色边还是异色边来确定自己的颜色。这证明了对于任意一种枚举的情况，均存在一种染色方案满足条件。所以对于树上点权均为 0 的问题，我们可以对每条边贪心地取对答案贡献较大的情况，答案为

$$\sum_{e\in E}\max\{s_e,d_e\}$$

接下来考虑仙人掌上点权均为 0 的问题。

4.3 算法六 仙人掌，点权均为 0

仍然可以使用深度优先搜索来判断是否存在一种满足条件的染色方案。搜索出任意一棵 DFS 树，按照 4.2 中的方法确定每个点的颜色，再判断所有的非树边是否均满足条件。

可以发现存在合法的染色方案，当且仅当不存在一个简单环使得环上异色边的数量为奇数。

由于仙人掌上每条边最多属于一个简单环，所以可以把每个简单环以及割边独立计算出答案再加起来。

对于割边，直接把 $\max\{s_i, d_i\}$ 计入答案即可。对于环，考虑先把所有边的 $\max\{s_i, d_i\}$ 计入答案，若异色边的数量为偶数，则取到理论最大值并且合法；否则可以找到使 $|s_i - d_i|$ 最小的一条边，把它两端颜色的异同性反转，即可得到最优解。修改的时候把上一次询问的答案加上修改边所在环或割边的答案的增量即可。

时间复杂度 $O((n+Q)\log n)$ 。

可以通过子任务 6 ，期望得分 19 分。

结合以上所有树和仙人掌的部分（算法三、算法四、算法六）可以得到 51 分。

5 较为一般的图的形态

广义串并联图的性质为，对于任意 4 个节点都不存在 6 条两两没有公共边的路径连接这 4 个节点中的每一对节点。之前的算法对于图的形态都有更高的要求，所以不适合用于这种较一般的情况，但是仍然可以从树与仙人掌出发，寻找一些更为通用的做法。为了方便分析，这里暂且不考虑需要修改的情况。

5.1 树上问题的另一种解决方法

我们先来看一道例题。

例题 1. 树上最大权独立集问题[48]

给定一棵 n 个节点的树 T ，节点 i 有点权 a_i 。求一个点集的子集 S 使得 S 中任意两个节点不相邻且 S 中节点的权值和最大。

数据范围 $1 \le n \le 10^5, 1 \le a_i \le 10^9$

时间限制 1 秒

空间限制 128 MB

[48]题目来源：经典问题

当然，这道题有类似算法二的动态规划算法，但是我们现在考虑另一种算法。

当所有 a_i 均为 1 时，有一种经典的贪心做法：

每次找到一个度数不超过 1 的节点 u ，把它加入 S（即贪心地认为 u 一定在 S 中），并在 T 中删去与 u 相邻的点 v（如果存在这样的 v ，根据题意若 $u \in S$ 则必有 $v \notin S$ ）以及 u 本身，直到 T 中不存在任何节点，此时 S 就为一组最优解。

这个算法的正确性证明如下：

证明. 假设存在一个最优解为 $S^* \neq S$ 。

当 a_i 均为 1 时，S 中节点的权值和最大相当于 $|S|$ 最大。

如果 u 的度数为 0 ，且不在 S^* 中，那么显然可以把 u 加入 S^* 来增加 S^* 的大小，所以 S^* 不是问题的最优解，与假设矛盾。

如果 u 的度数为 1 ，且不在 S^* 中，若 u 的相邻节点 $v \notin S^*$ ，则把 u 加入 S^* 同样可以得到一组更优的解，与假设矛盾；若 $v \in S^*$ ，则可以把 S^* 中的 v 替换成 u ，仍然是一组合法的最优解。

于是可以求出删去 u 和 v（如果存在）后的问题的最优解 S' ，然后令 $S = S' \cup \{u\}$ 就可以得到原问题的最优解。递归边界是图为空，此时显然 $S = \varnothing$ 就是问题的最优解。

这样每次操作至少会删去一个点。而树的非空子图一定存在一个度数不超过 1 的节点，因为树的非空子图的每个连通块必然是树，并且根据抽屉原理，树一定存在一个度数不超过 1 的节点。所以经过有限次操作后，算法一定会结束。 □

但是，如果去掉 a_i 均为 1 的限制，这个贪心算法就是错误的，原因在于在上面的证明中把 v 换成 u 时，可能会使解变劣，但是证明的其余部分仍然适用。

于是可以考虑改进一下这个贪心算法，在把 u 加入 S 后给一个撤销该操作并把 v 加入 S 的机会。这相当于之后可以考虑把 v 加入 S ，但是需要额外花费 a_u 的代价把原来已经在 S 中的 u 移出 S 。这一步可以通过把 a_v 改为 $a_v - a_u$ 来实现。若 $a_v < a_u$ ，那么最优解中一定包含 u 而不包含 v ；因为若包含 v ，则把 v 替换为 u 后可以得到更优的解。于是在这种情况下可以直接把 u 加入 S 并删去 v 。改进后的算法可以处理一般的情况。

把改进后的算法用在本题的树上问题上也是类似的。若遇到一个度数为 1 的节点 u ，那么找到它的相邻节点 v 。若把 v 染成黑色，那么 u 以及边 (u,v) 最多可以贡献的权值为 $\max\{s_{(u,v)} + b_u, d_{(u,v)} + w_u\}$ ；若把 v 染成白色，那么 u 以及边 (u,v) 最多可以贡献的权值为 $\max\{d_{(u,v)} + b_u, s_{(u,v)} + w_u\}$ 。于是可以把 b_v 改为 $b_v + \max\{s_{(u,v)} + b_u, d_{(u,v)} + w_u\}$ ，把 w_v 改为 $w_v + \max\{d_{(u,v)} + b_u, s_{(u,v)} + w_u\}$ ，并删去节点 u 和边 (u,v) 。这样操作后与操作前的问题的答案相等，但从问题规模上前者比后者少了一个节点。我们称这个把一个 1 度点[49]删去并修改其相邻点的点权的操作为**删 1 度点操作**。

一棵点数大于 1 的树一定有 1 度点，并且删去一个 1 度点后仍为一棵树。所以不断地对一棵树进行删 1 度点操作，可以使树转化为只包含一个节点的图，且答案不变。而仅包

[49]本文中把“度数为 k 的节点”简称为“ k 度点”

括单个点 u 的图的答案显然为 $\max\{b_u, w_u\}$ ，从而一个树上的静态问题可以在 $O(n)$ 的时间内得到解决。

删 1 度点操作对图的整体形态并没有要求，只要图中有一个度数为 1 的节点就可以进行该操作。

解决了树上的问题，现在可以去仙人掌上寻找更多的启发。

5.2 在仙人掌上的应用

这里不加证明地给出一些要用到的关于点双连通分量的知识：

定理 5.1. *对于任意的非空无向图 G ，一定存在一个 G 的点双连通分量 B ，使得 B 中只有不超过 1 个节点是 G 的割点。其中，若 B 中没有 G 的割点，则有 $B = G$ 。*

定理 5.2. *若一个点双连通分量不为 K_2 [50]，则该点双连通分量中至少有一个简单环。*

引理 5.1. *在仙人掌上的每个点双连通分量要么是 K_2 ，要么是一个简单环。*

引理 5.2. *对于一个不是 K_2 的点双连通分量中的任意一个点 u ，一定存在一个简单环 C 使得 u 在 C 上。*

树上的问题可以使用删去 1 度点（叶子）的方式简化问题，那么仙人掌能否用类似的方法呢？

首先若存在度数为 1 的节点，则同样可以使用删 1 度点操作。

否则可以考虑定理 5.1 ，找到一个点双连通分量 B 使得 B 中**只有不超过一个节点**是 G 的割点；再结合引理 5.1 ，B 一定是一个简单环。

此时 B 在图中的地位与度数不超过 1 的节点在树上的地位类似。那么类比树上的做法，能否将 B 转化为一个节点，来减少 G 中点双连通分量的数量呢？

答案是肯定的。

由于点双连通分量 B 是一个环，所以 B 中的节点中除了割点（如果割点存在），度数均为 2 。

在遇到 1 度点时，我们根据其相邻点的染色方案计算其带来的贡献，在遇到 2 度点时同样可以使用这个方法。

设 2 度点 x 的相邻两个节点（编号）为 u, v 。

当 $c_u = 0, c_v = 0$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{00} = \max\{s_{(u,x)} + b_x + s_{(x,v)}, d_{(u,x)} + w_x + d_{(x,v)}\}$$

当 $c_u = 0, c_v = 1$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{01} = \max\{s_{(u,x)} + b_x + d_{(x,v)}, d_{(u,x)} + w_x + s_{(x,v)}\}$$

[50] 定义 K_n 为包含 n 个节点的完全图。

当 $c_u = 1, c_v = 0$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{10} = \max\{d_{(u,x)} + b_x + s_{(x,v)}, s_{(u,x)} + w_x + d_{(x,v)}\}$$

当 $c_u = 1, c_v = 1$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{11} = \max\{d_{(u,x)} + b_x + d_{(x,v)}, s_{(u,x)} + w_x + s_{(x,v)}\}$$

由于该贡献同时与 c_u, c_v 的取值有关，所以不能像删 1 度点时那样考虑更新 u, v 的点权，而是可以考虑新建一条连接 u, v 的边，在边上记录 c_u, c_v 的每种取值的情况下对答案的贡献。

我们用 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 表示该边对答案的贡献为 $w_{c_u c_v}$ 。我们称它的边权为

$$\begin{bmatrix} w_{00} & w_{01} & w_{10} & w_{11} \end{bmatrix}^T$$

输入中给出的边是 $(u, v, (s, d))$ 的形式。为了使格式一致，可以把它改为 $(u, v, (s, d, d, s))$ 。

我们将把一个 2 度点和其相邻两条边转化为一条边的操作称为**缩 2 度点操作**。

可以注意到，缩 2 度点操作不会使 u, v 的度数发生改变。

于是对于一个 n $(n \geq 3)$ 元环，可以通过使用一次缩 2 度点操作使其转化为一个 $(n-1)$ 元环。特殊地，对于一个 3 元环，使用一次缩 2 度点操作后会得到两条重边。

由于贡献是可以叠加的，所以若遇到两条重边 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 和 $(u, v, (w'_{00}, w'_{01}, w'_{10}, w'_{11}))$ ，可以将它们合并为一条边

$$(u, v, (w_{00} + w'_{00}, w_{01} + w'_{01}, w_{10} + w'_{10}, w_{11} + w'_{11}))$$

我们称把两条重边合并为一条边的操作为**叠合重边操作**[51]。

设 B 上有 k 个节点，因为 B 中只有一个 G 上的割点，而其他点都是 2 度点，且缩 2 度点操作和叠合重边操作不会使 2 度点的度数变得大于 2 ，所以在经过 $(k-2)$ 次缩 2 度点操作和一次叠合重边操作之后， B 就转化成了 K_2 。再进行一次删 1 度点操作，G 中就减少了一个点双连通分量。

所以经过 $O(n)$ 次操作之后，可以把 G 转化为一个节点。

这样就可以在 $O(n)$ 的时间内解决仙人掌上的静态问题。

5.3 广义串并联图

下面证明，不仅是仙人掌，任意广义串并联图都可以在若干次缩 2 度点、叠合重边、删 1 度点的操作后变为一个只包含一个节点的图。

[51]重边的方向有可能是相反的，即可能遇到 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 和 $(v, u, (w'_{00}, w'_{01}, w'_{10}, w'_{11}))$ ，此时只要改变其中一条边的方向再按上述方法叠合即可，叠合后的边为 $(u, v, (w_{00} + w'_{00}, w_{01} + w'_{10}, w_{10} + w'_{01}, w_{11} + w'_{11}))$ 。

5.3.1 证明

考虑证明它的逆否命题，即证明任意一个不能用上述操作使其变为一个只包含一个节点的图的无向连通图不是广义串并联图。

这里先证明两个结论：

定理 5.3. *若一张无向连通图 G 中存在 3 个不同的 1 度点 x,y,z，则一定存在一个点 $u \notin \{x,y,z\}$ 使得存在 3 条两两没有公共边的简单路径满足其中一个端点均为 u 且另一个端点分别为 x,y,z。*

证明. 先求出 G 的任意一棵生成树 T。

由于 x,y,z 是 1 度点，所以在 T 中 x,y,z 一定是 1 度点。

由于任意图中度数为奇数的点必有偶数个，所以 G 中一定存在一个异于 x,y,z 的节点。

取一个异于 x,y,z 的点作为根，把 T 转化为有根树，令 u 为 x 和 y 的最近公共祖先，即令 $u = \mathrm{LCA}(x,y)$。因为 x,y,z 均为叶子且不为根，所以有 $u \neq x, u \neq y, u \neq z$。

若 z 不在 u 的子树中或 $\mathrm{LCA}(x,z), \mathrm{LCA}(y,z)$ 均为 u，那么在 T 上 u 到 x,y,z 的三条路径两两没有公共边。

否则 $\mathrm{LCA}(x,z)$ 和 $\mathrm{LCA}(y,z)$ 恰有一个不为 u。

不失一般性，假设 $w = \mathrm{LCA}(x,z) \neq u$，那么 y 不在 w 的子树中，又因为 $\mathrm{LCA}(x,z) = w$，所以在 T 上 w 到 x,y,z 的三条路径两两没有公共边。 □

定理 5.4. *对于一个无向图 G，若进行若干次删 1 度点操作，缩 2 度点操作以及叠合重边操作后得到的图不是广义串并联图，那么 G 也不是广义串并联图。*

证明. 考虑还原所进行的操作。

设操作后的图中 4 个使图不满足广义串并联图的性质的节点为 a,b,c,d，6 条两两边不相交的路径为 $p(a,b), p(a,c), p(a,d), p(b,c), p(b,d), p(c,d)$。

删 1 度点操作的逆操作为选择一个点 $u \in V$，加入新点 v 和边 (u,v)。

叠合重边操作的逆操作为选择一条边 $e = (u,v) \in E$，加入一条新的重边 (u,v)。

由于这两个逆操作不会删除图中的节点或边，所以若删 1 度点操作或叠合重边操作后的图不是广义串并联图，那么操作前的图也不是广义串并联图。

缩 2 度点操作的逆操作为选择一条边 $e = (u,v) \in E$，删去 e 并加入新点 w 以及新边 $(u,w), (w,v)$。

若还原时没有删去这 6 条路径中任何一条路径上的边，则原来的 $p(a,b), p(a,c), p(a,d), p(b,c), p(b,d), p(c,d)$ 同样能够作为判断操作前的图不是广义串并联图的依据。否则不失一般性，假设要删去 $p(a,b)$ 上的一条边。那么可以在 $p(a,b)$ 中删去那条边并加入逆操作需要加入的两条边和一个节点，这样仍然满足存在 $p(a,b), p(a,c), p(a,d), p(b,c), p(b,d), p(c,d)$ 两两没有公共边。

因此，若任意多次操作之后的图不是广义串并联图，那么操作之前的图也一定不是广义串并联图。 □

一张简单无向图不能再进行操作，当且仅当每个连通块是一个 0 度点或一个所有点的度数都大于等于 3 的图。下面给出定理：

定理 5.5. *任意一张所有点的度数都大于等于 3 的简单无向连通图，一定不是广义串并联图。*

证明. 假设有一个满足所有点的度数都大于等于 3 的简单无向连通图 G 。

根据定理 5.1，可以在 G 中找到一个点双连通分量 B ，使得 B 中只有不超过 1 个节点使得该节点是 G 的割点。

这个点双连通分量 B 一定不是 K_2 ，否则 G 中就会存在一个 1 度点。

若 B 中有 G 的割点，根据引理 5.2，B 中一定存在一个包含该唯一割点的简单环，设其为 C 。

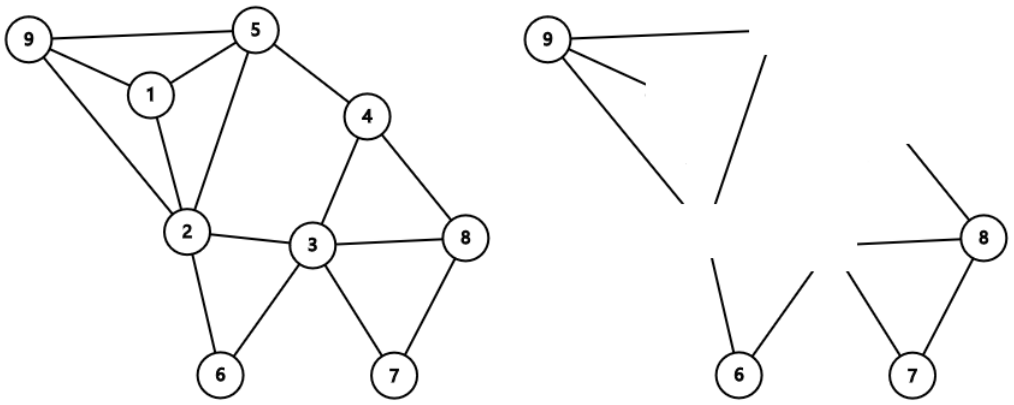
否则根据定理 5.2，B 中一定存在一个简单环。任取一个 B 中的简单环，并设为 C 。

设 C 上的点依次为 $v_0, v_1, \cdots, v_{l-1}$ ，即 v_0 和 v_{l-1} 相邻，对于所有的 $0 \le i < l-1$ ，v_i 与 v_{i+1} 相邻。

设 $V' = V \setminus V_C, E' = E \setminus E_C$ ，在 $V' \cup E'$ 上定义二元关系 $\sim$ ：对于 $u, e \in V' \cup E'$ ，$u \sim e$ 当且仅当 $u \in V', e \in E'$ ，且 u 是 e 的一个端点。再定义 $V' \cup E'$ 上的等价关系 $\leftrightarrow$ 是二元关系 $\sim$ 的最小等价关系。直观地说，如果 a, b 是 G 中的顶点或边，且不在 C 中，那么 $a \leftrightarrow b$ 当且仅当从 a 出发可以经过一系列不在 C 中的点和边到达 b ，这类似图中的连通性。

我们称等价关系 $\leftrightarrow$ 的等价类为**片**。对于片 U ，定义它的**点集**为 $U \cap V$ ，它的**边集**为 $U \cap E$ ，它对 C 的**联系点集**为 $A(U) = \{v \in V_C \mid \exists u \in V,\ e = (u, v) \in U \cap E\}$ ，直观地说，在 G 中删去所有在 C 上的顶点和边（但是那些不在 C 上的边即使一个或两个顶点被删去也应该保留），“连通”的部分就是片；$A(U)$ 就是片 U 的没有或者只有一个端点的边所缺少的端点集合。

以下是一个例子：（左边的图删去环 $1 \to 2 \to 3 \to 4 \to 5 \to 1$ 后会得到右边的图）



右图中共有 4 个片，这 4 个片包含的边的数量分别为 $1,2,3,4$ ，联系点集的大小分别为 $2,2,3,2$ 。

由于 G 中每个节点的度数都大于等于 3 ，所以对于环上的任意一个点 v_i ，至少存在一个片 U 使得 $v_i \in A(U)$ 。

若 B 中存在 G 的割点则设该割点为 v_k （此时割点唯一且在 C 上），若不存在割点则 $k=-1$ 。

若存在一个 U 使得 $|A(U)|=1$ ，由于删去其中的唯一元素之后 G 不连通，即它是 G 的割点，所以对于所有满足 $|A(U)|=1$ 的 U ，均满足 $A(U)=\{v_k\}$ （显然当 $k=-1$ 时 v_k 不存在，这也意味着当 $k=-1$ 时不存在 U 使得 $|A(U)|=1$ ）。

接下来分三种不同的情况讨论说明 G 一定不是广义串并联图。

情况 1. 存在一个 U 使得 $|A(U)|\geq 3$ 。

假设 U 中有一条边 e 满足其两个端点均不在 U 中，那么 e 与 V' 中的任意一个元素 u 均不可能满足 $u\sim e$ 。自然地，对于 $V'\cup E'$ 中的任意一个元素 a ，均不满足 $e\leftrightarrow a$ 。所以有 $U=\{e\}$ ，且 $A(U)$ 中的元素为 e 的两个端点，与 $|A(U)|\geq 3$ 矛盾。因此，U 中不可能有一条边 e 满足 e 的两个端点均不在 U 中。

可以考虑把 $A(U)$ 中的所有元素都加入 U ，得到一张图 $U'=((U\cap V)\cup A(U),U\cap E')$ 。在 U' 中，$U\cap V$ 是连通的，这是因为 U 是等价关系 $\leftrightarrow$ 的等价类，而 U 中存在一个端点不属于 U 的边不传导 U 的点集在 U' 中的连通性。

在 U' 中，删去若干条与 $A(U)$ 的元素相邻的边使得 $A(U)$ 的元素的度数均为 1 。此时 $A(U)$ 中的点仍然通过一条 U' 中的边与 U 的点集连通，因此 U' 是连通图。

这样根据定理 5.3 ，设 $x,y,z\in A(U)$ ，U 中一定存在一个节点 u 使得 U' 中有 3 条两两没有公共边的路径分别连接 u 与 x ，u 与 y ，u 与 z 。又因为 x,y,z 在同一个环 C 上，所以在 C 上存在 3 条两两没有公共边的路径分别连接 x 与 y ，y 与 z ，z 与 x 。因为 U' 与 C 没有公共边，所以找到了 4 个节点 x,y,z,u 使得存在 6 条两两没有公共边的路径分别连接这 4 个节点的每一对，即说明了 G 不是广义串并联图。

情况 2. 对于所有的 U 都有 $|A(U)|\leq 2$ ，并且存在 U_1,U_2 使得 $A(U_1)=\{v_x,v_y\},A(U_2)=\{v_z,v_w\}$ ，其中 $x<z<y<w$ 。

此时在环 C 上可以找到 4 条两两没有公共边的路径连接 v_x 与 v_z ，v_z 与 v_y ，v_y 与 v_w ，v_w 与 v_x ，且 G 中存在一条连接 v_x 与 v_y 的路径满足其边集包含于 U_1 的边集，存在一条连接 v_z 与 v_w 的路径满足其边集包含于 U_2 的边集。又因为 U_1 的边集、U_2 的边集、C 的边集两两不交，所以找到了 6 条两两没有公共边的路径连接 4 个节点 x,y,z,w 中的每一对，即说明了 G 不是广义串并联图。

情况 3. 对于所有的 U 都有 $|A(U)|\leq 2$ ，并且存在 U_0,i 使得 $A(U_0)=\{v_i,v_{i+1}\}$ 。

以下说明，情况 $1,2,3$ 涵盖了所有可能情况。

如果不满足情况 1 ，我们有以下事实：

1. 对于环上的任意一个点 v_i ，至少存在一个片 U 使得 $v_i \in A(U)$ ；

2. 所有满足 $|A(U)| = 1$ 的 U 均满足 $A(U) = \{v_k\}$ ；

3. 简单环 C 至少包含 3 个节点；

4. 不存在任何 U 使得 $A(U) \geq 3$ 。

综合上述四个事实，我们知道一定能找到一个片 U 使得 $|A(U)| = 2$ 。

假设 $A(U) = \{v_x, v_y\}(x < y)$ ，由于环可以按照顺时针或逆时针来定向，所以可以不失一般性地假设 $k \notin (x, y)$ 。

下面证明若不满足情况 1 和情况 2 ，则一定存在 U_0, i 使得 $A(U_0) = \{v_i, v_{i+1}\}$ 。

若 $y - x = 1$ ，则取 $U_0 = U, i = x$ 可以满足要求。

否则找到 U_1, z 使得 $A(U_1) = \{v_{x+1}, v_z\}$ 。根据事实 *1,2,4* ，且 $k \notin (x, y)$ ，这样的 U_1, z 一定存在。

若 $z = x$ ，则取 $U_0 = U_1, i = x$ 可以满足要求；若 $z \notin [x, y]$ ，则 U 与 U_1 满足情况 2 ，矛盾。

所以若 $z \neq x$ ，则必有 $z \in (x + 1, y]$ 。

令新的 $x' = x+1, y' = z$ ，则有 $y'-x' = y'-x-1 \leq y-x-1$ ，并且仍有 $y' > x', k \notin (x', y')$ 。

每一次 x, y 到 x', y' 的迭代都会使 $y - x$ 的值减小，所以经过有限次迭代后一定能够找到满足条件的 U_0, i 。

所以若不满足情况 1 和情况 2 ，则一定存在 U_0, i 使得 $A(U_0) = \{v_i, v_{i+1}\}$ 。

令 $G' = (V_{G'}, E_{G'})$ 为 U_0 加入节点 v_i, v_{i+1} 以及边 (v_i, v_{i+1}) 后的无向图。那么有 $deg(v_i) \geq 2, deg(v_{i+1}) \geq 2, \forall u \in U_0, deg(u) \geq 3$ 。

所以，G' 中不存在度数不超过 1 的点，G 中 2 度点的个数不超过 2 ，并且若存在 2 个 2 度点，则这 2 个 2 度点必然相邻。

可以证明 $V_{G'} \subseteq V_B$ 成立：若 $k \neq -1$ ，假设存在 $u \in V_{G'}$ 且 $u \notin V_B$ ，则连接 u 和 B 中任意一点的任意一条路径必然经过 v_k ，而 $v_k \in V_C$ ，所以有 $A(U_0) = \{v_k\}$ ，与 $|A(U_0)| = 2$ 矛盾；若 $k = -1$ ，则根据定理 5.1 有 $B = G$ ，所以有 $V_{G'} \subseteq V = V_B$ 。

下面说明 G' 是点双连通图，即 G' 不存在割点。

首先 v_i 和 v_{i+1} 不是 G' 的割点，否则 U_0 不连通。

其次 G' 中其余节点也不是 G' 的割点，否则它同时也是 G 的割点。已知当 $k = -1$ 时 B 中没有 G 的割点，而 $k \neq -1$ 时 B 中只有 v_k 是 G 的割点，然而因为 $V_{G'} \cap V_C = \{v_i, v_{i+1}\}$ ，所以一定有 $v_k \notin V_{G'} \setminus \{v_i, v_{i+1}\}$ 。

所以，G' 不存在割点，即 G' 是点双连通图。

可以通过枚举得到，对于所有节点数不超过 3 的简单图，均不能同时满足以上两个性质[52]。所以 G' 至少包含 4 个节点。

考虑对 G' 进行缩 2 度点、叠合重边操作。缩 2 度点操作不会影响与 2 度点相邻的两个点 u, v 的度数，而之后的叠合重边操作（如果需要）只会使 u, v 的度数减少 1 。

[52]这两个性质分别为“是一个点双连通图”和“不存在度数不超过 1 的点且 2 度点的数量不超过 2 。

所以，若 G' 只有一个 2 度点，那么在操作后，G' 中不存在度数不超过 1 的节点，并且 2 度点数量仍然不超过 2 ，并且若有 2 个 2 度点，则这 2 个 2 度点必然相邻。

若 G' 有两个相邻的 2 度点 x,y ，设与 x 相邻的另一个节点为 z ，与 y 相邻的另一个节点为 w ，则 $z \neq w$ ，否则有 z 的度数为 2 或 z 是 G' 的割点。不失一般性，假设接下来对 x 进行缩 2 度点操作，由于 y 与 z 之间没有边，所以不会进行叠合重边操作，y 与 z 的度数均不变，所以操作之后 G' 中只有 1 个 2 度点，没有 1 度点。

接下来证明经过一次缩 2 度点以及一次叠合重边操作（如果缩 2 度点之后出现重边）之后，G' 仍是点双连通图。

设操作之后的图为 G'' 。

判断一个图是点双连通图的依据是图中不存在任何一个点 u 使得删去 u 后图不连通。

由于重边不会改变点的连通性，所以叠合重边操作不会对图的点双连通性产生影响。

现在考虑缩 2 度点操作。设 G' 中操作的 2 度点为 x ，与 x 相邻的两个节点为 y,z 。

考虑在 G'' 的边 (y,z) 中“插入”一个节点 x ，即加入节点 x 后把边 (y,z) 替换为边 (y,x) 和边 (x,z) ，这样 G'' 就变成了 G' 。

所以证明删去 G'' 中任何一个节点，G'' 均连通，相当于证明删去 G' 中任何一个不为 x 的节点，G' 均连通。

因为 G' 是点双连通图，所以删去 G' 中任意一个节点 G' 均连通，所以 G'' 是点双连通图。

所以在操作过程中的任一时刻，G' 是点双连通图，且 G' 中不存在度数不超过 1 的点，且 G' 中 2 度点的数量不超过 2 。

每一次缩 2 度点操作会使 G' 的点数减少 1 。假设某一时刻 G' 的点数为 3 ，又因为 G' 是点双连通图，所以此时 G' 一定是 K_3 。K_3 有 3 个 2 度点，而任一时刻 G' 中的 2 度点数量不超过 2 ，所以 G' 不能够转化为一个不超过 3 个节点的图。又因为任意时刻 G' 中均没有 1 度点，所以在经过足够但有限多次（可能为 0 次）缩 2 度点操作和叠合重边操作后，G' 的每个节点的度数都大于等于 3 。

由于最初的 G' 中只有 2 个节点在 C 上，而 C 至少有 3 个节点，所以 G' 的点数一定比 G 小。把操作后每个节点的度数都大于等于 3 的 G' 作为新的 G 迭代下去，G' 仍然能被归为情况 1,2,3 中的一种情况，其中 G' 满足情况 1 或情况 2 时 G' 不是广义串并联图，而满足情况 3 时 G' 的点数一定会减小，但 G' 的点数一定大于等于 4 ，故经过有限次迭代之后，一定能够找到 6 条两两没有公共边的路径连接 4 个顶点中的每一对顶点。根据定理 5.4 ，得出 G 不是广义串并联图。

因此，一张所有点的度数都大于等于 3 的简单无向连通图，一定不是广义串并联图。

□

显然，一个连通图进行缩 2 度点操作、叠合重边操作、删 1 度点操作均不能使其变为一个不连通的图。

根据定理 5.4 和定理 5.5 ，可以得出，任意一个不能用缩 2 度点操作、叠合重边操作、删 1 度点操作使其变为一个只包含一个节点的图的无向连通图不是广义串并联图。

由于删 1 度点操作与缩 2 度点操作会使图的节点数和边数均减少 1 ，并且叠合重边操作只可能在缩 2 度点操作之后执行并使边数减少 1 ，所以可以证明，任意广义串并联图可以在 $O(n)$ 次缩 2 度点、叠合重边、删 1 度点的操作后变为一个只包含一个节点的图；此外还可以推出，简单广义串并联图的边数是 $O(n)$ 级别的，更具体地，有 $m \le 2n$ 。

5.4 算法七 广义串并联图，没有修改

有了上面的结论，就可以直接使用 5.2 中的算法通过子任务 $1,3,4$ 。每次修改后暴力重新计算，就可以通过子任务 $2,5$ 。

时间复杂度 $O(nQ)$ 。

期望得分 30 分。

6 考虑修改

现在有了一种能够在线性时间内解决一组询问的算法，那么可以考虑在此之上进行一些优化，使其能更高效地支持修改。

先来考虑一种较为简单的情况：子任务 7 中所有点的点权均为 0 ，此时无需考虑点权对答案造成的影响。

6.1 算法八 广义串并联图，点权均为 0

考虑需要使用的所有操作。

叠合重边操作是把两条重边的边权对应相加，并换成一条新的边。

缩 2 度点操作是把一个 2 度点的相邻两条边，经计算后换成一条新的边。由于点权均为 0 ，所以 2 度点的点权对新边的边权没有影响。

注意到当点权为 0 时 $w_{00} = w_{11}, w_{01} = w_{10}$ ，所以可以如题目描述中的那样，只用 s_e 和 d_e 来表示一条边 e 的边权。

对于删 1 度点操作，由于可以通过确定 1 度点的颜色来最大化其相邻边的贡献，所以可以把 1 度点相邻的边的 $\max\{s_i, d_i\}$ 直接计入答案。

由于叠合重边操作与缩 2 度点操作都是把两条边的权值通过计算变为一条新的边的权值，所以可以用运算的形式来表示它们。

6.1.1 定义运算

设一条边 e 的边权为 $v_e = \begin{bmatrix} s_e & d_e \end{bmatrix}^T$ 。

对于两条边的边权，定义 $+$ 运算的结果为叠合重边操作后的边权：

$$\begin{bmatrix} s_1 \\ d_1 \end{bmatrix} + \begin{bmatrix} s_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} s_1 + s_2 \\ d_1 + d_2 \end{bmatrix}$$

定义 $*$ 运算的结果为缩 2 度点操作后的边权：

$$\begin{bmatrix} s_1 \\ d_1 \end{bmatrix} * \begin{bmatrix} s_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} \max\{s_1 + s_2, d_1 + d_2\} \\ \max\{s_1 + d_2, d_1 + s_2\} \end{bmatrix}$$

可以发现，$+$ 运算和 $*$ 运算均满足交换律。

对于删 1 度点操作，可以定义初始答案 $ans = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$，然后遇到 1 度点时令 ans 为 $ans * v_e$（其中 e 为 1 度点相邻的边）。

由于每条边的边权都可以由最初的边权用 $+$ 运算和 $*$ 运算表示，所以整个问题的答案也可以用一个表达式来表示。于是可以建立这个表达式的表达式树，使问题重新变为树形动态规划的形式，进而用链分治的方法优化修改。

6.1.2 建立表达式树

以下是建立表达式树的过程：

定义 op_v 为表达式树上非叶子节点 v 上的运算符。

首先对于每一条边 e 新建表达式树上的对应节点 $v_e = \begin{bmatrix} s_e & d_e \end{bmatrix}^T$ [53]，对初始答案新建节点 $ans = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$。

然后使用算法七来对图 G 进行操作，直到 G 只剩下一个节点。在操作的过程中：

1. 若需要进行缩 2 度点操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令原有的两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ ‘$*$’；

2. 若需要进行叠合重边操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令原有的两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ ‘$+$’；

3. 若需要进行删 1 度点操作，则在操作后建立表达式树上的节点 v，令 ans 与删去的边在表达式树上的对应节点的父亲节点为 v，并定义 $op_v =$ ‘$*$’，最后令 ans 为 v。

6.1.3 分治

考虑沿用算法三的分治。

在表达式树中，每个非叶子节点都有两个子节点，所以设 $c_{v,0}, c_{v,1}$ 为 v 的两个子节点，设 sn_v 为 v 的重子节点的参数编号，即 $c_{v,sn_v} = son_v$ 为 v 的重子节点。

[53] 为叙述方便，这里不区分一个节点和其对应的权值。

为了快速维护重链上的信息，仍然沿用算法三中 g_v 的定义，即

$$f_v = g_v f_{son_v}$$

对于非叶子节点 v ，设 $son_v = \begin{bmatrix} x & y \end{bmatrix}^T, c_{v,1-sn_v} = \begin{bmatrix} z & w \end{bmatrix}^T$ 。

1. 若 $op_v = '+'$ ，则有

$$v = \begin{bmatrix} x+z \\ y+w \end{bmatrix}$$

所以有

$$g_v = \begin{bmatrix} z & -\infty \\ -\infty & w \end{bmatrix}$$

2. 若 $op_v = '*'$ ，则有

$$v = \begin{bmatrix} \max\{x+z, y+w\} \\ \max\{x+w, y+z\} \end{bmatrix}$$

所以有

$$g_v = \begin{bmatrix} z & w \\ w & z \end{bmatrix}$$

然后就可以套用算法三来解决问题了。唯一不同的是，算法三中的 g 不能暴力计算，必须通过维护增量的方式来维护 g 的值，而本算法中 g 的值可以暴力计算。

该算法时间复杂度为 $O(n + Q\log^2 n)$ ，可以通过子任务 $6, 7$ 。

期望得分 36 分。

最后可以开始考虑点权对问题的影响。

6.2 算法九 广义串并联图，带修改

考虑在算法八的基础上进行改进。

此时缩 2 度点操作和删 1 度点操作均需要考虑到点权，并且边权中需要记录 $w_{00}, w_{01}, w_{10}, w_{11}$ 四个值，即记

$$v_e = \begin{bmatrix} w_{00} \\ w_{01} \\ w_{10} \\ w_{11} \end{bmatrix}$$

此外，还需要记录点权，对于节点 u 记

$$v_u = \begin{bmatrix} b_u \\ w_u \end{bmatrix}$$

算法八中的 $+$ 运算和 $*$ 运算已经不适用，所以需要重新定义一下运算。

6.2.1 扩展运算

同样地，记 $+$ 运算的结果为叠合重边操作产生的新边的边权，即

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} a+x \\ b+y \\ c+z \\ d+w \end{bmatrix} \tag{33}$$

记 $*$ 运算的结果为缩 2 度点操作产生的新边的边权，而此时该边权与操作删去的 1 个节点和 2 条边的权值均有关。设 $e_0 = (u_0, u_1, (a,b,c,d)), e_1 = (u_1, u_2, (x,y,z,w)), u_1 = (\alpha, \beta)$ ，则有

$$*\left(\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}\right) = \begin{bmatrix} \max\{a+\alpha+x, b+\beta+z\} \\ \max\{a+\alpha+y, b+\beta+w\} \\ \max\{c+\alpha+x, d+\beta+z\} \\ \max\{c+\alpha+y, d+\beta+w\} \end{bmatrix} \tag{34}$$

对于删 1 度点操作，需要定义 $\oplus$ 运算的结果为操作修改的点（即被删去的点的相邻点）的点权，设 $e_0 = (u_0, u_1, (a,b,c,d)), u_0 = (\alpha, \beta), u_1 = (\gamma, \delta)$ 且 u_1 为操作选择的 1 度点，则有

$$\oplus\left(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \begin{bmatrix} \gamma \\ \delta \end{bmatrix}\right) = \begin{bmatrix} \max\{\alpha+a+\gamma, \alpha+b+\delta\} \\ \max\{\beta+c+\gamma, \beta+d+\delta\} \end{bmatrix} \tag{35}$$

此外，还需要考虑将一条边反向[54]，即把 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 变为 $(v, u, (w_{00,}, w_{10}, w_{01}, w_{11}))$ ，所以定义 R 运算为

$$R\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix} \tag{36}$$

6.2.2 建立表达式树

建立表达式树的方法与算法八类似，但不同的是，这里的 $*$ 运算和 $\oplus$ 运算是不满足交换律的，所以需要考虑子节点的顺序。此时定义 $c_{v,0}, c_{v,1}, c_{v,2}$ 依次为 $*$ 运算或 $\oplus$ 运算的三个参数，即 $v = *(c_{v,0}, c_{v,1}, c_{v,2})$ 或 $v = \oplus(c_{v,0}, c_{v,1}, c_{v,2})$ 。

使用算法七来对图 G 进行操作，直到 G 只剩下一个节点。在操作的过程中：

[54]在某些实现方法中需要进行该运算，也存在一些实现方法不需要进行该运算，而是使用一些别的方法来解决关于边的方向的问题。我的做法使用了该运算，所以我把它写在了这里。

1. 若需要进行缩 2 度点操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令操作中删去的一个节点和两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，确定 $c_{v_{e'},0}, c_{v_{e'},1}, c_{v_{e'},2}$ 的值，并令 $op_{e'} =$ '$*$'；

2. 若需要进行叠合重边操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令原有的两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ '$+$'；

3. 若需要进行删 1 度点操作，则在操作后对修改的点 u' 建立表达式树上的对应节点 $v_{u'}$，令与操作相关的两个节点和一条边在表达式树上的对应节点的父亲节点为 $v_{u'}$，确定 $c_{v_{u'},0}, c_{v_{u'},1}, c_{v_{u'},2}$ 的值，并令 $op_{u'} =$ '$\oplus$'；

4. 注意操作的时候需要考虑边的方向，若需要改变边的方向，则在操作后建立表达式树上的节点 $v_{e'}$，令需要改变方向的边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ 'R'。

6.2.3 分治

同样地，只要能够计算出 g 矩阵，就可以运用算法八的方法进行分治。

对于非叶子节点 v，

1. 若 $op_v =$ '$+$'，设

$$\begin{aligned} son_v &= \begin{bmatrix} x & y & z & w \end{bmatrix}^T \\ c_{v,1-sn_v} &= \begin{bmatrix} a & b & c & d \end{bmatrix}^T \end{aligned}$$

根据 (33)，

$$v = \begin{bmatrix} x+a \\ y+b \\ z+c \\ w+d \end{bmatrix}$$

所以

$$g_v = \begin{bmatrix} a & -\infty & -\infty & -\infty \\ -\infty & b & -\infty & -\infty \\ -\infty & -\infty & c & -\infty \\ -\infty & -\infty & -\infty & d \end{bmatrix}$$

2. 若 $op_v =$ 'R'，根据 (36) 则有

$$g_v = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

3. 若 $op_v =$ ‘$*$’，设

$$\begin{aligned} c_{v,0} &= \begin{bmatrix} a & b & c & d \end{bmatrix}^T \\ c_{v,1} &= \begin{bmatrix} \alpha & \beta \end{bmatrix}^T \\ c_{v,2} &= \begin{bmatrix} x & y & z & w \end{bmatrix}^T \end{aligned}$$

根据 (34)，

$$v = \begin{bmatrix} \max\{a+\alpha+x, b+\beta+z\} \\ \max\{a+\alpha+y, b+\beta+w\} \\ \max\{c+\alpha+x, d+\beta+z\} \\ \max\{c+\alpha+y, d+\beta+w\} \end{bmatrix}$$

3.1. 若 $sn_v = 0$，则有

$$g_v = \begin{bmatrix} \alpha+x & \beta+z & -\infty & -\infty \\ \alpha+y & \beta+w & -\infty & -\infty \\ -\infty & -\infty & \alpha+x & \beta+z \\ -\infty & -\infty & \alpha+y & \beta+w \end{bmatrix}$$

3.2. 若 $sn_v = 1$，则有

$$g_v = \begin{bmatrix} a+x & b+z \\ a+y & b+w \\ c+x & d+z \\ c+y & d+w \end{bmatrix}$$

3.3. 若 $sn_v = 2$，则有

$$g_v = \begin{bmatrix} a+\alpha & -\infty & b+\beta & -\infty \\ -\infty & a+\alpha & -\infty & b+\beta \\ c+\alpha & -\infty & d+\beta & -\infty \\ -\infty & c+\alpha & -\infty & d+\beta \end{bmatrix}$$

4. 若 $op_v =$ ‘$\oplus$’，设

$$\begin{aligned} c_{v,0} &= \begin{bmatrix} \alpha & \beta \end{bmatrix}^T \\ c_{v,1} &= \begin{bmatrix} a & b & c & d \end{bmatrix}^T \\ c_{v,2} &= \begin{bmatrix} \gamma & \delta \end{bmatrix}^T \end{aligned}$$

根据 (35)，

$$v = \begin{bmatrix} \max\{\alpha+a+\gamma, \alpha+b+\delta\} \\ \max\{\beta+c+\gamma, \beta+d+\delta\} \end{bmatrix}$$

4.1. 若 $sn_v = 0$ ，则有

$$g_v = \begin{bmatrix} \max\{a+\gamma, b+\delta\} & -\infty \\ -\infty & \max\{c+\gamma, d+\delta\} \end{bmatrix}$$

4.2. 若 $sn_v = 1$ ，则有

$$g_v = \begin{bmatrix} \alpha+\gamma & \alpha+\delta & -\infty & -\infty \\ -\infty & -\infty & \beta+\gamma & \beta+\delta \end{bmatrix}$$

4.3. 若 $sn_v = 2$ ，则有

$$g_v = \begin{bmatrix} \alpha+a & \alpha+b \\ \beta+c & \beta+d \end{bmatrix}$$

这里构造 g 矩阵的方式类似于构造一般的线性关系的转移矩阵。在代码实现的过程中，可以不需要对 sn_v 进行分类讨论来计算 g_v 的值，具体方式详见我的代码[55]的 tran 函数。

其余的步骤，均与算法八相同，这里不再重复描述。

该算法的时间复杂度为 $O(n + Q\log n)$ ，可以通过全部测试点，期望得分 100 分。

6.3 优缺点

6.3.1 优势

通过缩小问题规模的方式来解决问题，会使动态规划的过程更加直观，从而减少一些思维量。

相比于一般的链分治算法，使用表达式树上链分治的方法，不需要考虑一个节点有多个轻子节点的情况，从而降低代码实现的难度。

6.3.2 局限性

使用表达式树上链分治的方法通常只能支持全局或子树的查询，而难以支持链上问题的查询，且该方法通常需要在线段树上维护矩阵的乘积，这会带来不小的常数因子。

7 拓展

对于一些问题，需要统计最优方案的数量。此时可以把最优方案数同时记录在矩阵的元素中。令矩阵中的每个元素均为一个二元组 (a, cnt) ，其中 a 表示最优值，cnt 表示最优

[55] https://loj.ac/submission/433979

方案的数量，并定义

$$\begin{aligned}(a_1, cnt_1) + (a_2, cnt_2) &= (\max\{a_1, a_2\}, cnt_1[a_1 \geq a_2] + cnt_2[a_2 \geq a_1]) \\ (a_1, cnt_1) \times (a_2, cnt_2) &= (a_1 + a_2, cnt_1 \times cnt_2)\end{aligned}$$

不难发现，这里定义的加法和乘法满足交换律、结合律，并且乘法对加法满足分配律，所以能够使用矩阵乘法的形式来转移。

所以若《公园》或类似的题要求输出最优方案数（对大质数取模），那么仍然可以使用树链剖分线段树维护矩阵乘积的方法。

8 命题思路

这道题最初源于我对独立集问题的一些思考。在搜索最大独立集[56] 时有一个剪枝：若存在一个度数为 1 的节点，那么一定存在一个最优解包含这个度数为 1 的节点，于是可以删去这个度数为 1 的节点和其相邻节点，缩小问题规模。那么放在最大权独立集问题[57]上是否也有类似的方法呢?

可以发现，若 1 度点的点权大于等于其相邻节点的点权，那么前面所述的结论[58]仍然正确。否则我们可以先假定与那个 1 度点相邻的边不存在，并且选择该点。那么在之后的过程中若选择了与原来那个 1 度点相邻的节点，就意味着不能选择该 1 度点，于是可以把相邻节点的点权减去 1 度点的点权来处理这种情况。

钟知闲在 IOI2017 候选队的论文《浅谈信息学竞赛中的独立集问题》中提到了一种先搜索度数大于等于 3 的节点的算法，该算法的复杂度很大程度取决于枚举的度数大于等于 3 的节点个数。我发现通过删去 1 度点的方式可以在一定程度上减少需要枚举的度数大于等于 3 的节点的个数（如树上的独立集问题不需要枚举任何度数大于等于 3 的节点），于是想到，能否对度数为 2 的节点做类似的操作，来进一步减少需要枚举的度数大于等于 3 的节点个数，从而增加算法效率甚至优化复杂度呢？通过类似于删 1 度点方法的研究，我发现这是可以做到的，于是我设计了一个更加优秀一些的最大权独立集算法。

接着我想，这个算法在图满足什么性质的时候，不需要枚举任何度数大于等于 3 的节点呢？首先我就想到了仙人掌，显然仙人掌是满足条件的，但是我觉得，满足条件的图不仅仅有仙人掌，可以考虑更一般的情况。通过研究发现，这个算法不需要枚举任何度数大于等于 3 的节点，当且仅当图的每个连通块都是广义串并联图。这让我产生了《公园》命题的最初构想。

[56]最大独立集问题即给定图 $G = (V, E)$ ，求一个最大的 V 的子集 S 使得 E 中任意一条边的两个端点中均有至少一个不属于 S

[57]最大权独立集问题即给定图 $G = (V, E)$ 和权值函数 $w : V \to \mathbb{R}^+$ ，求一个 V 的子集 S 使得 E 中任意一条边的两个端点中均有至少一个不属于 S 且 $\sum_{u \in S} w(u)$ 最大

[58]即“一定存在一个最优解包含这个度数为 1 的节点”