解题报告

宁波市镇海中学 罗煜翔

CF627 F Island Puzzle

题目大意

给定一棵 n 个点的树，每个点上有一个初始权值 a_i 和目标权值 b_i，保证 $\{a_n\},\{b_n\}$是 $0 \sim n-1$ 的排列。

需要把点权从初始权值重新排列成目标权值，可以进行如下操作:

- 选择一条边，满足这条边的一个端点的权值是 0。交换这条边的两个端点的权值。

现在可以添加至多一条边，要求在添加边的数量最小的情况下最小化操作次数，并给出添加边的方案。

如果无解，输出 -1。

数据范围

$n \le 200000$。

解题过程

首先考察是否需要添加边以及添加哪些边是可行的。

注意到，连续对一条边操作两次相当于不进行任何操作，从而操作具有**可逆性**，且操作次数最少时不会进行这样的操作。同时，操作也可以看做是权值为 0 的点的**移动**。

设初始时 0 在顶点 s，最终需要让 0 到达顶点 t。

由于操作具有可逆性，不妨先通过操作将 0 从 s 移动到 t。于是接下来的操作需要让 0 走一个回路。

首先考虑不添加边的情况：在一棵树上，将 0 经过一些操作移动从 t 移动回 t 一定是反复地利用重复走过一条边的操作。从而这样的操作相当于不做任何操作。所以可以不添加边当且仅当所有点的权值都与目标权值相等。

- 这也说明了，在树上把 0 从一个点移动到另一个点，不论如何进行移动结果都是**相同**的，即沿着这两点间的路径进行移动。

对于需要添加边的情况，设添加的边为 (u,v)。于是，整个图变成了一个基环树，基环树中的环为原树中 u,v 间的路径和边 (u,v) 组成。

考虑 0 的移动路线：从 t 出发，然后经过若干次 (u,v) 这条边，最后回到点 t。

如果同时存在 $u \to v$ 和 $v \to u$ 两种经过边 (u,v) 的方式，那么必然存在相邻两次经过边 (u,v) 分别是这两种方式。但这样就要在树上从 v 移动回 v 或从 u 移动回 u，相当于没有移动。而连续两次经过 (u,v) 这条边，也相当于没有移动。这表明这样的移动方式不是最短的，从而最短的移动方式中只存在 $u \to v$ 或 $v \to u$ 中的一种经过边 (u,v) 的方式，不妨设为 $u \to v$。

于是 0 的移动可以看做：

- 先沿着树边从 t 移动到 u，并沿 (u,v) 边移动到 v；

- 然后进行若干（可以为 0）次绕圈：
    - 从 v 沿着树边移动回 u，再沿 (u,v) 边移动到 v;
- 最后沿树边从 v 移动回 t。

这也可以描述为：先从 t 移动到环上，然后绕着环移动若干圈，最后移动回 t。

设 t 从 p 点进入环。特别地，如果 t 本身在环上那么 $t=p$。于是，p 为 u,v 间路径上离 t 最近的节点，或者说 u,v,t 这三点的最近公共祖先（即以 t 为根，u 和 v 的最近公共祖先）。

首先考察 t 到 p 这段路径上点的权值变化。由于 0 在环上绕圈对这段路径的点权没有影响，而除去绕圈的部分相当于从 t 移动到 p 再从 p 移动回 t，也不会对这段路径的点权造成改变。所以这段路径上的点的权值不变。

再考察环上权值变化情况：

- 从 t 移动到 p 的过程中，除了 p 外环上的点都没有变化。而 p 的点权变为了 0。
- 绕一圈后，环上除 p 外的点的变化形成一个**循环**，而绕若干圈则为一个**轮换**（即**循环**的幂次）。
    - 从而除了 p 以外的点的权值或者全部不变，或者全部都改变。
- 最后从 p 移动回 t 时，只会将 p 的权值复原。
- 由于需要添加边，说明存在权值需要变化的节点。从而环上除了 p 以外的所有点权值都应该改变，即：
    - 所有权值需要改变的点就是 u,v 在树上的路径上除了 p 以外的所有点。

现在考虑确定 u,v。首先以 t 为根对原树进行 DFS。此时取所有权值需要改变的点中，深度最小的点的父亲节点就是 p。这个点和所有权值需要改变的点的并就是 u,v 路径上的点。从而就确定了 u,v。

- 如果深度最小的点不唯一，或者这些点不是树的一条路径，或者权值需要改变的点按照环上顺序排列后权值的改变情况不是一个**轮换**，则说明无解。

最后，来计算最少的操作次数。

对于不需要添加边的情况就是 $\mathrm{dist}(s,t)$，其中 $\mathrm{dist}(x,y)$ 表示 x,y 在树上的距离。

对于需要添加边的情况，分按照 $u\to v$ 和 $v\to u$ 经过 (u,v) 边进行讨论。以第一种为例，由这个轮换可以计算出 $u\to v$ 边需要经过的最少次数 c。而 0 实际的移动过程应当是：

- 先沿着树边从 s 移动到 u，并沿着 (u,v) 边移动到 v。
- 绕这个环 $c-1$ 次，即重复从 v 沿着树边移动到 u 再沿着 (u,v) 边移动到 v 的过程 $c-1$ 次。
- 从 v 沿着树边移动到 t。
- 于是操作次数为 $\mathrm{dist}(s,u)+1+(c-1)(\mathrm{dist}(v,u)+1)+\mathrm{dist}(v,t)$。

最后对两种情况取较小值。

综上所述，本题以 $O(n)$ 的时间复杂度解决。

AGC035 D Add and Remove

题目大意

给定一个长度为 N 的数列 A，每次操作可以选择其中连续的三项，权值依次为 a,b,c，将其替换为 $a+b,c+b$ 这两项。

求经过操作最后剩下的两个数的和的最小值。

数据范围

$2 \le N \le 18, 0 \le A_i \le 10^9$。

解题过程

首先，操作可以看做是选择不是 A_1, A_N 的一个数，将它删去并把它加到左右两项上。

所以 A_1, A_N 实质上并没有影响这个过程，于是问题可以重新叙述为：

- 给定 N 个数 $A_1, A_2, \cdots, A_N$，每次可以删除一项，把它的值加到左右两边。删除最左侧的数会把它的值加到变量 L 中，删除最右侧的数会把它的值会加到变量 R 中。删除最后的一个数会把它的值同时加到 L, R 中。
- 最小化删除完所有数后 $L+R$ 的值。
- 这里的 N 实际上是原题中的 $N-2$。
- 接下来我们都考虑这个问题。

首先考虑下面这个类似区间 DP 的方法：

- 考虑计算 $S(l,r)$ 表示删除 $A_l, A_{l+1}, \cdots, A_r$ 这些数之后得到的 (L,R) 二元组形成的集合。
- 每次转移时枚举最后一个删除的数为 A_i，于是对于 $(L_1,R_1) \in S(l,i-1)$ 和 $(L_2,R_2) \in S(i+1,r)$,可以得到 $(L_1+(R_1+L_2+A_i), R_2+(R_1+L_2+A_i)) \in S(l,r)$。
- 最后对于所有的 $(L,R) \in S(1,N)$，计算 $L+R$ 的最小值。
- 可以通过一些偏序关系或保留凸包等手段删去无用状态进行加速。
- 足够优秀的实现可以保证 $|S(l,r)| \le 2^{r-l+1}$，而且这也是比较松的，于是可以通过本题。

从上面的做法可以看出，逆向考虑这个问题的操作会有不错的结果：

- 设最后一次删除的数为 A_i：
  - A_i 会同时加到 L,R 中，在最终答案中的系数为 2。
  - $1 \sim i-1$ 这一部分的 R 加到 A_i 中，而 L 加到最终的 L 中，需要计算 $L+2R$ 的最小值。
  - $i+1 \sim N$ 这一部分的 L 加到 A_i 中，而 R 加到最终的 R 中，需要计算 $2L+R$ 的最小值。
- 一般地，设删除了区间 $l \sim r$ 的所有 A_i 后 $uL+vR$ 的最小值为 $f(l,r,u,v)$，则：
  - 设最后一次操作的是 A_i，于是 $l \sim i-1$ 需要计算 $uL+(u+v)R$ 的最小值，$i+1 \sim r$ 需要计算 $(u+v)L+vR$ 的最小值。
  - 于是即为 $(u+v)A_i+f(l,i-1,u,u+v)+f(i+1,r,u+v,v)$。
  - 对所有的情况取最小值即可。

    - 最终答案为 $f(1,N,1,1)$。
- 直接利用上式进行搜索，考虑分析其时间复杂度：
    - 时间复杂度显然只与 $r-l+1$ 有关，设为 $T(r-l+1)$。
    - 若只考虑状态数，则 $T(0)=1$，则 $T(n)=\sum_{i=0}^{n-1}(T(i)+T(n-1-i))+1$，可以得到 $T(n)=3^n$。
    - 若考虑转移数，依然有 $T(n)=O(3^n)$。
- 于是我们以 $O(3^N)$ 的时间复杂度解决了问题。

考虑对上述搜索继续优化，一个直接的想法是进行记忆化搜索：

- 直接使用哈希表存储 l,r,u,v 的状态，即可实现记忆化搜索。
- 实际上，注意到 u,v 的状态形成了一个二叉树结构，即由 $(u,v)\to(u,u+v),(u+v,v)$，于是可以用 $x\to 2x,2x+1$ 的方式进行存储，可以直接用数组实现，常数比哈希更好。
- 对于记忆化搜索的时间复杂度：
    - 粗略分析：l,r 的状态是 $O(N^2)$ 的，u,v 的状态是 $O(2^N)$ 的，转移是 $O(N)$ 的，所以总的复杂度是 $O(2^NN^3)$ 的。
    - 更精确的分析：
        - 设进行了 i 次向右 DFS（即 $(u,v)\to(u+v,v)$）和 j 次向左 DFS，则 $l\ge i+1$，$r\le N-j$，(u,v) 有 $\binom{i+j}{i}$ 种可能，而考虑转移的复杂度则相当于选择三个整数 l,m,r 满足 $i+1\le l\le m\le r\le N-j$，从而时间复杂度为 $O\left(\binom{i+j}{i}\binom{N-i-j+2}{3}\right)$。
            - 实际上，上式在 $i,j\ne 0$ 时是精确的。由于在 $i=0$ 时必有 $l=1$，在 $j=0$ 时必有 $r=N$，所以会比实际值略大。
        - 合并 $i+j$ 相同的项，总时间复杂度为 $O\left(\sum_{i=0}^{N}2^{N-i}\binom{i+2}{3}\right)=O(2^N\cdot 8)=O(2^N)$。
            - 转移的精确数量是 $2^{N+3}-\dfrac{N^4+2N^3+23N^2+58N+96}{12}$，状态的精确数量是 $2^{N+2}-\dfrac{N^3+11N+9}{3}-[N=0]$。这里的 N 是原题中的 $N-2$，结果是利用上述的 i,j 中有 0 的特殊处理与一些和式变化得到的。
        - 使用直接数组的存储方式，空间上需要 $O(2^NN^2)$，需要进一步处理以达到 $O(2^N)$，但由于这个空间可以承受，所以这里就不再优化了。

综上所述，本题以 $O(2^N)$ 的时间复杂度解决。

AGC033 F Adding Edges

题目大意

给定一棵树 T 和一张简单无向图 G，T 和 G 的顶点个数为 N，编号均为 $1 \sim N$，G 的边有 M 条。

每次操作可以选择在树的一条链上（在链上排列的顺序不限）的三个点 a, b, c，如果边 (a, b) 和边 (b, c) 均在 G 中，而边 (a, c) 不在 G 中，则可以添加边 (a, c) 到 G 中。

当操作到无法操作时，求图 G 的总边数。

数据范围

$2 \leq N \leq 2000, 1 \leq M \leq 2000$。

解题过程

我们可以给出如下的朴素算法：

- 按照被添加的先后顺序处理所有边。
  - 对于初始时在 G 中的边，可以按照输入的顺序规定被添加的顺序。
- 首先将初始在 G 中的所有边加入待处理队列。
- 设当前处理边 (a, b)。对于所有 c，满足 (b, c) 在 G 中，(a, c) 不在 G 中且 a, b, c 在 T 的一条链上，则将 (a, c) 添加进 G 和待处理边的队列中。
- 当待处理队列为空时，G 的边数就是答案。

由于至多处理 $O(N^2)$ 条边，处理一条边需要 $O(N)$ 的时间，所以该算法时间复杂度为 $O(N^3)$。

重新叙述上述算法，首先对于所有的 a, b，计算出 $S(a, b)$ 表示与 a, b 在 T 的同一条链上的点 c 形成的集合：

- $S(a, b)$ 是如下三个部分的并：当 T 以 a 为根时，子树 b 中所有的点；当 T 以 b 为根时，子树 a 中所有的点；T 中a 到 b 的路径上的点。
- 以 $1 \sim N$ 中的每个点为根，分别进行一次 DFS ，求出这三部分。
- 只以 1 为根 DFS 一次，进行一些在 $O(N^2)$ 时间内完成的预处理，可以直接计算出 $S(a, b)$：
  - 设 $P(a)$ 表示 1 到 a 路径上的点，$Q(a)$ 表示以 1 为根，子树 a 中的点。
  - 可得 $(P(a) \bigtriangleup P(b)) \cup \{\text{lca}(a, b)\}$ 为 a 到 b 路径上的点，其中 $A \bigtriangleup B = (A \setminus B) \cup (B \setminus A)$ 表示集合的对称差，$\text{lca}(a, b)$ 表示以 1 为根，点 a 和点 b 的最近公共祖先。
  - 如果 $\text{lca}(a, b) \neq b$，那么以 a 为根，子树 b 中的点就是 $Q(b)$；
  - 如果 $\text{lca}(a, b) = b$，设 b 到 a 的路径上的下一个点是 c，则以 a 为根，子树 b 中的点就是 $\complement_V Q(c)$ 。

接下来考虑处理边的过程，当处理 (a, b) 这条边时：

- 令 $N(v)$ 表示当前和 v 在 G 中有边相连的点集， $C = (N(a) \bigtriangleup N(b)) \cap S(a, b)$，则 C 是恰好和 a, b 中的一个点在 G 中有边相连，且在 T 中与 a, b 在一条链上的点形

成的集合。
- 于是对于所有 $c \in C$ 且 $c \neq a, b$，添加 (a, c) 和 (b, c) 中不在 G 中的那一条边即可。

使用 bitset 实现上述的集合以及相关操作，时间复杂度为 $O\left(\frac{N^3}{\omega}\right)$，空间复杂度为 $O(N^2)$，可以通过此题。

上述算法并没有充分利用 a, b, c 在树 T 同一条链上这一条件，利用这一条件可以得出如下算法：

我们将题目描述中的操作称为**添加边**操作，定义下面的操作为**缩短边**操作：

- 对于**依次排列**在 T 的一条链上的三个点 a, b, c，如果图 G 中存在 $(a, b), (a, c)$ 这两条边：
  - 删去 (a, c) 这条边。
  - 如果 (b, c) 边不存在，添加 (b, c) 这条边。
- 事实上，这个操作或者将边 (a, c) 缩短成了 (b, c)，或者缩短成了被删除。这里的缩短指的是边的两个端点在树 T 上距离的缩短。

对于**缩短边**操作，我们可以得到：

- 由于每条边至多被缩短 $N-1$ 次，所以这样的操作至多进行 $(N-1)M$ 次。
- 由于不论是否进行缩短边操作，在 a, b, c 三点间进行**添加边**操作都可以达到 a, b, c 两两连边的状态，所以进行**缩短边**操作不影响答案。

对 G 进行**缩短边**操作，直到无法操作为止。

对于现在的图 G，判定两点间最终是否连边是很容易的：

- 对任意两点 a, b，a, b 间可以利用**添加边**操作连接边当且仅当存在一条 G 中从 a 到 b 的路径 $a = x_1 - x_2 - \cdots - x_k = b (k \geq 2)$，且 $x_1, x_2, \cdots, x_k$ **依次排列**在 T 的一条链上。

证明上述的命题：

- 充分性是显然的，因为一旦存在这样的路径，那么 (x_1, x_2) 已经在 G 中，而 $(x_1, x_3), (x_1, x_4), \cdots, (x_1, x_k)$ 可以依次通过**添加边**操作得到。
- 对于必要性，用反证法。
  - 初始存在于 G 的边显然满足条件，设最早通过操作得到的不满足条件的边为 (a, c)，它是由边 $(a, b), (b, c)$ 进行**添加边**操作得到的。
  - 由于 $(a, b), (b, c)$ 是初始在 G 中或更早通过**添加边**操作得到的边，所以它们都满足条件，即存在链 $a = x_1 - x_2 - \cdots - x_l = b = y_1 - y_2 \cdots - y_r = c$。
  - 若 a, b, c **依次排列**在某条链上，那么 $x_1 - x_2 - \cdots - x_l - y_2 - \cdots - y_r$ 就是满足条件的链，与反证假设矛盾。
  - 若 a, b, c 排列在某条链上，但不**依次排列**。
    - 则 a, b, c 三点在链上的排列中，b 点位于端点处。于是 x_l, x_{l-1}, y_2 或 x_l, y_2, x_{l-1} 中存在一个是**依次排列**的。
    - 但不论哪一种，说明可以进行**缩短边操作**，与 G 无法操作矛盾。
  - 综上所述，必要性成立。

事实上，上述命题也表明，对于此时的 G，在**添加边**操作时保证有一条是初始 G 中的边不会影响最终答案。

现在我们来实现上述的内容。对 G 充分的进行**缩短边**操作，可以通过如下方式完成：

- 考虑依次加入边，设当前加入边 (a,b)：
  - 如果 (a,b) 已经存在，那么结束加边的过程。
  - 如果存在 (a,c) 这条边，且 a,c,b **依次排列**在一条链上，那么将 (a,b) 改为 (c,b)，继续考虑加入 (c,b)。对于 b 处缩短 (a,b) 情况类似处理。
  - 如果不存在上述边，则保留 (a,b) 这条边。
    - 此时，如果存在 (a,c) 这条边，且 a,b,c **依次排列**在一条链上，那么将 (a,c) 改为 (b,c)，继续添加 (b,c)。对于 b 处缩短其他边的情况类似处理。
- 朴素实现上述过程，寻找这样的 c 需要 $O(N)$ 的时间，**缩短边操作**导致的加入边有 $O(NM)$ 次，总的时间复杂度为 $O(N^2M)$。
- 利用数据结构等优化上述过程：
  - 具体来说，对于当前存在的所有边 (a,b)，考察顶点 a 的情况，把以 a 为根，子树 b 中的所有点均标记上 b，这样的标记一共有 N^2 个。
  - 进行添加边时，如果当前已经有了标记，说明需要缩短当前这条边。
  - 否则依次考虑子树中情况，找到子树中的所有当前存在的边，缩短这些边，并给整个子树打上标记。
  - 使用线段树，平衡树等数据结构，并采用 DFS 序将子树转换为区间，可以做到 $O(NM\log N)$ 的时间复杂度。
- 事实上，上述过程中，对于已经有标记的点，可以保持之前的标记，而只对当前没有标记的点添加标记。
  - 这是因为，考虑一条边 (a,b)，如果之前存在过标记 c，这说明 (a,c) 边曾经存在过，且满足 a,c,b **依次排列**在一条链上。
  - 所以，我们可以在缩短边 (a,c) 之前把边 (a,b) 缩短为 (b,c)。
  - 于是，我们可以直接对整个子树做一次 DFS：
    - 如果当前点没有标记，那么就添加标记，并继续搜索。
    - 如果当前点有标记，那么就找到了一条需要缩短的边。此时子树中已经有标记，不需要继续搜索。
  - 这样，每次标记一个新的点或者缩短一条边，所以时间复杂度是 $O(N^2+NM)$。

在对 G 充分的进行了缩短边操作之后，借助 DFS 就可以完成计算：

- 考虑枚举边的一个端点 a，枚举可以作为边的另一个端点的点。
- 设当前找到了 b 使得最终 a,b 有边相连，考察所有在 G 中和 b 相连的点 c，如果 a,b,c 依次排列在一条链上，则说明 a,c 最终有边相连，然后继续搜索。
- 对于一个点进行 DFS 的时间复杂度是 $O(M)$ 的，于是这部分的时间复杂度是 $O(NM)$ 的。
- 类似地，利用之前的标记可以得到一个 $O(N^2)$ 的方法。

综上所述，本题以 $O(N^2+NM)$ 的时间复杂度解决。