NOIP 2026 模拟赛　Day 1 题解（详细版）

目录

T1 自胡蹿（string） **2**

T2 跳跃（jump） **4**

T3 套娃（dolls） **7**

T4 导览图（map） **10**

T1 自胡蹿（string）

考察知识点：区间 DP。

题意

给一个长度为 n 的字符串 s，一次操作可以把某个字符复制一份、插在它自己后面（比如 abc 在位置 1 操作，变成 $aabc$）。问最少操作几次，才能让 s 里出现一个长度 $\geq k$ 的回文子串。$n \leq 3000$，$k \leq 10^9$。

先想清楚：我们其实在干什么

操作只能把一个字符「变多」，不能新造字符、也不能改顺序。所以最终那个回文串，一定是从原串里**取出一段连续的字符**，再把其中一些字符多复制几份得到的。

比如 $s = aaabaa$，取出整段，把最右边的 a 复制一次，就变成 $aaabaaa$，这是个长度 7 的回文串。

所以做法的框架是：**枚举取原串的哪一段 $[i, j]$，算出把这一段变成回文最少要几次操作。**

区间 DP

设 $dp_{i,j}$ 表示「把 $s_i s_{i+1} \cdots s_j$ 这一段变成回文串，最少复制几次」。如果 $s_i \neq s_j$，那这段是没救的（首尾字符不同，复制也改不了首尾是谁），记为 $+\infty$。如果 $s_i = s_j$，有三种选择：

- $dp_{i,j} = dp_{i+1,j-1}$：首尾这两个字符正好配成一对，剩下中间那段自己去变回文；
- $dp_{i,j} = dp_{i+1,j} + 1$：左边的 s_i 没人配，那就花一次操作在右端复制一个字符出来陪它；
- $dp_{i,j} = dp_{i,j-1} + 1$：对称，右边多一个，在左端复制一个。

三者取最小值。这就是最普通的区间 DP，按区间长度从小到大枚举即可，$O(n^2)$。

举个例子体会一下：$s = aaabaa$，看整段 $[1, 6]$。首尾都是 a，用第三种转移走到 $dp_{1,5}$（也就是 $aaabaa$ 去掉最后一个 a）$= 1$？不如看第一种：$dp_{2,5}$ 对应 $aabaa$，它本来就是回文，$dp_{2,5} = 0$，所以 $dp_{1,6} \leq dp_{2,6} + 1$。总之最优是 1 次：把右边补成 $aaabaaa$。

长度不够怎么办

$dp_{i,j}$ 只保证「变成了回文」，长度可能还不到 k。这时候继续在**正中间**那个字符上复制就行——每复制一次，回文串长度就加 1，而且还是回文。比如 $aaabaaa$ 想更长，就在中间的 b 上复制：$aaabbbaaa$ 依然是回文。

设这段原长 $len = j - i + 1$，花了 $dp_{i,j}$ 次操作后长度是 $len + dp_{i,j}$，还差 $k - len - dp_{i,j}$ 就补这么多次，于是这一段的总代价是

$$dp_{i,j} + \max\bigl(0,\ k - len - dp_{i,j}\bigr) \ = \ \max\bigl(dp_{i,j},\ k - len\bigr).$$

另外别忘了「只取一个字符、一路复制到长度 k」这种方案，代价 $k - 1$，它也是答案的一个候选（$k = 1$ 时答案就是 0）。所有候选取最小值就是答案。

核心代码

```
res = K - 1;                                    // 只取一个字符，一路复制
for (int len = 2; len <= n; len++)
  for (int i = 1, j; (j = i+len-1) <= n; i++) {
    dp[i][j] = INF;
```

```
    if (s[i-1] == s[j-1]) {                    // 首尾相同才有救
      dp[i][j] = min(dp[i][j], dp[i+1][j-1]);   // 首尾配对
      dp[i][j] = min(dp[i][j], dp[i+1][j] + 1); // 右端补一个
      dp[i][j] = min(dp[i][j], dp[i][j-1] + 1); // 左端补一个
    }
    res = min(res, dp[i][j] + max(0, K - len - dp[i][j]));
  }
```

部分分怎么拿

- **测试点** $1 \sim 4$（$n, k \le 10$，20 **分**）：暴力 BFS。从 s 出发，每一步枚举复制哪个位置，得到一堆新字符串，用 `set<string>` 判重，一层一层往外扩展，哪一层里出现了长度 $\ge k$ 的回文子串就输出层数。串很短，规模完全够。
- **测试点** $5 \sim 12$（$n, k \le 3000$，40 **分**）：写出上面的区间 DP 就够了。这一档 k 很小，所以「补长度」这一步你哪怕写成一格一格加的循环也不会超时。
- **测试点** $13 \sim 20$（$k \le 10^9$，40 **分**）：唯一的区别是 k 变得非常大。凡是「按 k 循环」「开一个长度 k 的数组」的写法都会 TLE 或 MLE，正好卡在 60 分。必须用上面那个 $\max(0, k - len - dp)$ 的式子一步算出来。

容易写错的地方

1. 忘记「只取一个字符」这种方案（res 的初值 $k-1$）。k 很大时答案往往就是它。
2. $dp_{i,j}$ 的初值要写 $+\infty$ 不能写 0，否则首尾不同的区间会被当成能变回文。
3. 只会想「以某个位置为中心向两边扩展」，结果漏掉了跨越好几个字符的情况——用区间 DP 就自动全考虑到了。
4. 这一档答案能到 10^9 左右，虽然还在 `int` 范围内，但保险起见用 `long long`。

T2 跳跃（jump）

考察知识点：代价拆分、连通段 DP（JOI「摩天大楼」套路）、贪心。

题意

数轴上 n 个点，坐标从小到大是 $x_1 < x_2 < \cdots < x_n$，每个点还有四个权值 a_i, b_i, c_i, d_i。要求一个排列 p，从 $p_1 = s$ 出发、到 $p_n = e$ 结束，每个点恰好经过一次，从 i 跳到 j 的代价是

$$\text{向左跳（}j<i\text{）：}(x_i - x_j) + c_i + b_j, \qquad \text{向右跳（}j>i\text{）：}(x_j - x_i) + d_i + a_j.$$

最小化总代价。$n \le 10^5$。

关键的第一步：把每一跳的代价「拆」给两个端点

代价里既有距离又有权值，看起来很乱。但把式子重新括一下就清楚了：

$$\underbrace{(x_i - x_j) + c_i + b_j}_{\text{向左跳}} = \underbrace{(x_i + c_i)}_{i\text{ 往左出发}} + \underbrace{(-x_j + b_j)}_{j\text{ 被右边的点跳进来}}$$

$$\underbrace{(x_j - x_i) + d_i + a_j}_{\text{向右跳}} = \underbrace{(-x_i + d_i)}_{i\text{ 往右出发}} + \underbrace{(x_j + a_j)}_{j\text{ 被左边的点跳进来}}$$

所以给每个点定义四个数：

$$A_v = x_v + a_v, \quad B_v = -x_v + b_v, \quad C_v = x_v + c_v, \quad D_v = -x_v + d_v,$$

它们的含义分别是「入边来自左边 / 入边来自右边 / 出边走向左边 / 出边走向右边」的花费。

在一条路径里，除了 s，每个点都恰有一条入边（贡献 A 或 B）；除了 e，每个点都恰有一条出边（贡献 C 或 D）。所以**总代价 = 每个点各自选一个入边花费 + 一个出边花费，加起来**。跳跃的具体顺序不重要了，只要知道「每个点的入边、出边分别朝左还是朝右」。

连通段 DP（95 分做法）

现在按坐标从小到大（也就是下标 $1, 2, \ldots, n$）一个一个把点「放进」我们要造的路径里。放到一半时，手上是若干条**零散的链**（路径的碎片）。比如已经放了 $1, 2, 3$，手上可能是 $3 \to 1$ 和 2 这两条链，将来再插入更大的点把它们串起来。

设 $f_{i,j}$ 表示「前 i 个点都放好了，手上有 j 条链」时的最小代价。现在插入第 $i+1$ 个点，它比之前所有点都靠右，所以它的边只要「朝左」就是连到已有的链上，「朝右」就是留个口子等以后的点来接。四种组合分别对应四种操作：

选择	含义	链数变化	代价
$B + D$	两条边都朝右，自己单独成为一条新链	$j \to j+1$	$-2x + b + d$
$A + C$	两条边都朝左，把两条已有的链接起来	$j \to j-1$	$+2x + a + c$
$A + D$	入边朝左、出边朝右，接在某条链的尾巴后面	j 不变	$a + d$
$B + C$	入边朝右、出边朝左，接在某条链的头部前面	j 不变	$b + c$

s 和 e 要特殊处理，因为它们各自只有一条边：s 只有出边，选 D 就是自己新建一条链、选 C 就是接到某条链的头部前面；e 只有入边，选 B 就是新建链、选 A 就是接到某条链的尾巴后面。另外要记住：s 永远是它所在链的**头**，e 永远是它所在链的**尾**。

最容易写错的：几个「不许这么干」的判断

链的头和尾就像插头，s 把它那条链的头堵住了，e 把它那条链的尾堵住了。所以：

- 想**接在尾巴后面**，得真的有一条尾巴是空着的。e 那条链的尾是堵住的，所以要求「e 还没放进来」或者「链数 $j>1$（那就换一条链去接）」。
- 想**接在头部前面**，同理，要求「s 还没放进来」或者 $j>1$。
- 想**合并两条链**，必须真有两条，所以要求 $j>1$。
- 如果某个状态是「只剩 1 条链，而且 s 和 e 都已经放进来了」，那这条链两头都堵死了，后面的点无处可放——这个状态直接判成非法（$+\infty$），除非所有点都放完了（那就是答案）。

漏掉最后一条，会算出比正确答案更小的值，样例还照样能过。

最终答案是 $f_{n,1}$。复杂度 $O(n^2)$，能拿 95 分。

```
// 插入普通点 i+1，x = x[i+1]，a/b/c/d 同理
if (j < n) f[i+1][j+1] = min(f[i+1][j+1], f[i][j] - 2*x + b + d);  // 新建链
if (j > 1) f[i+1][j-1] = min(f[i+1][j-1], f[i][j] + 2*x + a + c);  // 合并两条链
if (j) {
  if (e > i+1 || j > 1) f[i+1][j] = min(f[i+1][j], f[i][j] + d + a); // 接在尾巴后
  if (s > i+1 || j > 1) f[i+1][j] = min(f[i+1][j], f[i][j] + c + b); // 接在头部前
}
```

最后 5 分：贪心

把「合并」当成默认动作：每个普通点默认选 $A+C$。那另外三种选择就相当于在默认价格上打折，每个点手里有两张**折扣券**

$$\alpha_v = A_v - B_v, \qquad \gamma_v = C_v - D_v,$$

用掉 0 张就是合并（j 减 1），用 1 张是接在头/尾（j 不变），用 2 张是新建链（j 加 1）。

数一下就会发现：j 要从 0 变到 1，所以**总共用掉的折扣券张数是固定的**，恰好等于普通点的个数（最左边的点 1 是个例外，它的边只能朝右，两张券强制用掉）。既然总张数固定，那就是「从所有券里挑最大的若干张」——用一个**大根堆**，从左到右扫描，每遇到一个普通点就把当前可用的券丢进堆里，然后取出堆顶那张最大的用掉。注意用掉的可以是前面某个点的券，自己的券留给后面的点用，这样就不会漏掉更优的搭配。

唯一的技巧是**入堆的时机**：α_i 表示「i 的入边来自右边」，也就是 i 要当某条链的新头，这需要有一条链的头是空的；如果 s 已经放进来了（$i>s$），s 就占了一个头，所以 α_i 只能**等下一步**才可用，代码里表现为「先取数、再入堆」。γ_i 和 e 的关系同理。

```
if (i == s) q.push(C - D);          // s、e 只贡献券，不取券
else if (i == t) q.push(A - B);
else {
  ans += A + C;                     // 默认按“合并”计价
  if (i < s) q.push(A - B);         // 本步就能用的券，先入堆
  if (i < t) q.push(C - D);
  ans -= q.top(); q.pop();          // 取一张最大的折扣
  if (i > s) q.push(A - B);         // 只能下一步才用的券，后入堆
  if (i > t) q.push(C - D);
}
```

初始的 `ans` 是最左端那个点强制的部分：若 $1 \notin \{s, e\}$ 取 $w(s,1) + w(1,e)$，否则取 $w(s,e)$。复杂度 $O(n \log n)$。

部分分怎么拿

- **测试点 $1 \sim 4$（$n \le 100$，20 分）**：给各种「不够优」的做法留的口子。n 特别小的时候直接 $O(n!)$ 枚举所有排列就能拿；$n \le 100$ 则允许 $O(n^3)$ 甚至 $O(n^4)$ 的做法，常数特别大、写得比较糙的 $O(n^2)$ DP 也在这里保底。
- **测试点 $5 \sim 8$（$n \le 1000$，20 分）**：$O(n^2)$ DP 的舒适区，就算你的实现里套了 `map`、`vector` 之类常数很大的东西也没问题。
- **测试点 $9 \sim 12$（性质 A：$d_i \le c_i$ 且 $a_i \le b_i$，20 分）**：这条性质说的是「向右走的附加权值总是不劣于向左走」，也就是说**尽量往右走比较划算**。于是那种「先假设所有点都尽量向右走，再对不合法的地方做修补」的简化贪心，在这一档是对的，能拿分。注意性质 A 不改变复杂度，它只是让你可以写一个更简单、更好想的做法。
- **测试点 $13 \sim 16$（性质 B：$s = 1, e = n$，20 分）**：这就是 JOI「摩天大楼」的原始形态。s 是第一个被放进来的、e 是最后一个，于是上面那一堆「不许这么干」的判断全都消失了，DP 短很多。只会写这个简化版的选手正好拿这四个点。
- **测试点 $17 \sim 19$（$n \le 5000$，无性质，15 分）**：s, e 可以是任意位置，必须把端点在中间、而且 $s > e$ 的情况讨论干净。
- **测试点 20（$n \le 10^5$，5 分）**：只有 $O(n \log n)$ 的贪心能过。

容易写错的地方

1. **不开 `long long`**：x, a, b, c, d 都能到 10^9，$n = 10^5$，总代价能到 10^{14} 量级。
2. 忘记「只剩一条链且 s, e 都放好了」这个非法状态。
3. 贪心里两张券的入堆时机（取数之前还是之后）搞反了——样例能过，大数据会错。

T3 套娃（dolls）

考察知识点：单调栈判定、贪心、二分、倍增。

题意

n 个大小互不相同的套娃排成一行（a 是 $1\sim n$ 的排列）。每个位置记下它里面套娃的最小值 l_i 和最大值 r_i。相邻的两个位置能合并，当且仅当 $r_i < l_{i+1}$ 或 $r_{i+1} < l_i$，也就是两个位置的**大小区间不相交**（一个整体比另一个小）。求最多能合并几次。多测，$\sum n \le 5\times 10^5$。

第一步：答案 $= n-$ 最少块数

因为只能合并相邻位置，所以任何时刻每个位置装的都是原数组里**连续的一段**。合并到最后停下来时，整个数组被分成了若干连续段，每段内部自己合并完了。如果分成了 m 段，合并次数就是 $n-m$。所以：

$$\text{答案} = n - (\text{最少能分成几段}),$$

其中每一段都必须能**缩成一个位置**。我们把能缩成一个位置的区间叫**可缩区间**。

于是问题变成两个子问题：(1) 怎么判断一个区间可不可缩？(2) 怎么把数组分成最少的可缩区间？

第二步：用栈判定一段是否可缩

判定的关键结论是：**如果目标是把一整段缩成一个位置，那么每次合并的两个区间必须紧挨着，中间不能留空。**因为空缺里的那些套娃就在这一段的其他位置上，一旦被合并出来的大区间包住，就永久地被「套」在里面拿不出来了。

注意「紧挨着」要在**这一段内部离散化之后**再看：比如单独看 $[3,1]$ 这一段，中间缺的大小 2 并不在这一段里，不算空缺；把 $3,1$ 离散化成 $2,1$，它们就是紧挨着的，可以合并。所以判定前要先把这一段的值离散化。

知道「只能合并紧挨着的」之后，判定就变成了一个很机械的过程：从左到右扫描，维护一个栈，每来一个新的区间，就不停地看它和栈顶是否紧挨着，是就合并（合并后再和新的栈顶比），不是就压进栈。扫完之后，**如果栈里只剩一个元素，这一段就是可缩的。**（合并的顺序不影响结果，可以合并的机会不会因为你先合并了别处而消失，所以从左到右贪心地合并就是对的。）

```
bool chk(pii x, pii y){ return x.second+1 == y.first || y.second+1 == x.first; }
bool check(int l, int r) {                    // 判断 [l,r] 是否可缩
    cnt = 0;
    for (int i = l; i <= r; i++) lsh[++cnt] = a[i];
    sort(lsh+1, lsh+1+cnt);                   // 段内离散化
    int sz = 0;
    for (int i = l; i <= r; i++) {
        int x = get(a[i]); pii cur = {x, x};
        while (sz && chk(stk[sz], cur)) {      // 和栈顶紧挨着就合并
            cur = {min(stk[sz].first, cur.first), max(stk[sz].second, cur.second)};
            sz--;
        }
        stk[++sz] = cur;
    }
    return sz == 1;
}
```

一次判定是 $O(len \log len)$（排序），如果不排序、用桶计数可以做到 $O(len)$，但没必要。

第三步：贪心分段 + 二分

有了判定，怎么分段？答案是最朴素的贪心：**从左端点开始，每次取尽可能长的可缩区间**。这样做是对的，因为可缩区间有一条很好的性质：

> **可缩区间的任意子区间也可缩。**

（直观理解：一段能一层层套回去，那它里面的一小截当然也能。）有了这条性质，贪心就没有后顾之忧：第一段取得越长，剩下的部分只会更好分（因为剩下部分是原来那些段的子区间，照样可缩）。

这条性质还带来一个更重要的好处：固定左端点 l，可缩的右端点是**一段连续的前缀**——$[l, r]$ 可缩就意味着 $[l, r-1], [l, r-2], \ldots$ 都可缩。所以「以 l 开头最长的可缩区间」可以**二分！**

第四步：倍增——本题最重要的 trick

但直接在 $[l, n]$ 上二分是有问题的：二分的每一次判定，长度都可能是 $O(n)$ 级别的。如果块很多很短（比如几千个长度只有 10 的块），那每个块都要花 $O(n \log n)$ 去二分，总共就退化成 $O(n^2 \log n)$ 了。

解决办法是先用**倍增**把二分的范围「框」到块长的两倍以内：

1. 令 $k = 0$，不断检查 $[l, l + 2^k]$ 是否可缩，可缩就 k++，直到不可缩（或者超出 n）为止。这一步的所有判定长度是 $1 + 2 + 4 + \cdots + 2^k$，加起来和 2^k 同阶。
2. 此时已经知道答案在 $[l + 2^{k-1},\ l + 2^k)$ 里，也就是说 2^k 最多是块长的两倍。在这个范围里用倍增的方式二分出精确的右端点，判定次数 $O(\log)$，每次长度不超过 2^k。

这样每个块的工作量只和**它自己的长度**有关，而所有块的长度加起来是 n，总复杂度就是 $O(n \log^2 n)$（一个 log 来自二分次数，一个来自判定里的排序）。

```
int l = 1, res = 0;
while (l <= n) {
    int r = l, k = 0;
    while (l + (1<<k) <= n && check(l, l + (1<<k))) k++;   // 倍增找上界
    k--;
    while (k >= 0) {                                       // 在框好的范围里二分
        if (r + (1<<k) <= n && check(l, r + (1<<k))) r += 1<<k;
        k--;
    }
    res += r - l;                                          // 这一块贡献 len-1 次合并
    l = r + 1;
}
```

部分分怎么拿

- **测试点 $1 \sim 4$（$n, \sum n \le 2000$，20 分）**：想到「答案 $= n -$ 最少块数」+ 贪心 + 上面的栈判定就行，连二分都不用：对每个块的左端点 l，让 r 从 l 开始**一格一格往右试**，每次重新跑一遍判定，直到失败为止。$\sum n \le 2000$ 下随便过。

- **测试点** $5 \sim 8$（**性质 A：$|a_i - i| \le 300$，20 分**）：这条性质说每个套娃的大小和它的位置相差不超过 300，也就是数组「几乎是有序的」，局部性很强，可缩块通常都很短。所以上一档那个「一格一格往右试」的暴力，在这一档直接就能过 $n = 10^5$。
- **测试点** $9 \sim 12$（**性质 B：任意可缩区间长度 ≤ 1000，20 分**）：块长有了明确的上界 1000，于是**不需要倍增**了——直接在 $[l,\ \min(n, l + 1000)]$ 这个固定宽度的窗口里二分就行，每块 $O(\log)$ 次判定、每次长度不超过 1000。这一档如果还写「一格一格往右试」，块一长就会 TLE。
- **测试点** $13 \sim 15$（**$n \le 10^5$，无性质，15 分**）：块可以很长，「一格一格试」失效，必须二分。但因为 n 不大，**直接在 $[l, n]$ 上二分（不倍增）**也能过。
- **测试点** $16 \sim 20$（**$n \le 5 \times 10^5$，25 分，每点 5 分**）：必须倍增 + 二分才能拿满。只要有一档「块又多又短」的数据，不倍增的二分就会退化。

容易写错的地方

1. **多测不清空**。本题 T 最大 10^4。栈顶指针 `sz`、离散化的 `cnt`、答案累加器，忘了一个就 WA。
2. 把合并条件看成「两个区间相邻」而不是「不相交」。样例第三组 $a = [3, 1, 4, 2]$ 就是为此准备的。
3. 判定时直接用**原始的值**去比是否紧挨着，而没有先在段内离散化。$[3, 1]$ 用原始值看隔着一个 2，但它其实是可缩的。
4. 倍增时忘了 `l+(1<<k)<=n` 的边界判断，或者 k 减成 -1 之后还进二分的循环。

T4 导览图（map）

考察知识点：DFS 树、计数、树状数组数点、换根 DP。

题意

n 个城市两两之间都有路，其中 $n-2$ 条是「风景路」，它们构成一个森林，恰好有两个连通块。导览图就是任选一些路组成的集合。从城市 1 出发，每次走到**编号最小的没去过的邻居**，走不动了就沿原路退回。问有多少张导览图，能让这趟旅行走遍所有风景路，并且额外走的非风景路不超过一条。$n \le 10^6$，答案对 998244353 取模。

第一步：这个走法就是 DFS，走过的边就是 DFS 树

「走到没去过的邻居，走不动就回退」就是深搜。关键的一点：**被走过的边一定是 DFS 树边**。因为如果邻居已经访问过了，规则不允许你走过去——非树边永远不会被走。

所以题目的条件可以翻译成：

> 所有 $n-2$ 条风景路都得是 DFS 树边，而且 DFS 树里的非风景边不超过 1 条。

风景路构成的两个连通块，记含城市 1 的那块为 T_1，另一块为 T_2。DFS 树要包含它们的所有边，所以只有两种可能：

- **情形一**：DFS 树 $= T_1 + T_2 +$ 一条连接它们的桥 (p, q)（$p \in T_1$，$q \in T_2$）。这时 q 就是 DFS 进入 T_2 的入口。这样的 (p, q) 一共 $|T_1| \times |T_2|$ 种，都合法。
- **情形二**：DFS 树 $= T_1$，也就是压根没走到 T_2 去。这要求 T_2 里没有风景边，即 T_2 只有一个点 q；而且导览图里 q 必须是**孤立点**，否则 DFS 迟早会走到它。

第二步：固定一棵树，有多少张导览图？

现在假设 DFS 树 T 已经定下来了（以城市 1 为根）。导览图必须包含 T 的所有边，除此之外还可以再加一些「不影响遍历的废边」。那么答案就是 $2^{\text{废边条数}}$（每条废边独立地选或不选）。

哪些边是废边？考虑加一条边 $u \to w$。DFS 走到 u 时，会按**编号从小到大**依次去访问 u 的儿子。如果这条新边让 DFS 在某个时刻改了主意，就不是废边。分析一下：

- 如果 w 在 u 的子树里，设 u 通往 w 的那个儿子是 c。DFS 在 u 处依次挑儿子时，w 一直是没访问过的（它要等 c 的子树被走完才会被访问），所以只要 w 的编号**比 c 大**，DFS 每次都会优先选真正的儿子，w 就不会捣乱；等 c 的子树走完，w 已经访问过了，更不会捣乱。所以条件是 $\boxed{w > c}$。
- 如果 u, w 是「表兄弟」（谁都不是谁的祖先），那么一定会捣乱：两个子树中必有一个先被走完，走完的那一刻另一边还没访问，DFS 就会顺着这条边走过去，它就变成树边了。

所以**废边只能是「祖先—后代」之间的边，而且后代的编号要大于路径上那个儿子的编号**。统计的时候按树边来数最方便：

$$f(T) = \sum_{\text{每条树边}(u,c)} \#\{\, c \text{ 的子树里编号比 } c \text{ 大的点} \,\}.$$

第三步：把 $f(T)$ 拆成互不干扰的三块

现在要对所有的桥 (p, q) 求和：$\sum_{p,q} 2^{f(T)}$。如果每换一个 (p, q) 都重新算一遍 $f(T)$，那太慢了。好消息是 $f(T)$ 能拆成三块，而且每一块只跟 p 或只跟 q 有关：

- **两端都在 T_1 里的废边**：路径完全在 T_1 内部，跟桥**一点关系都没有**。记这个数为 S（在 T_1 上算一次就行）。
- **祖先在 T_1、后代在 T_2 里的废边**：因为 T_2 整个挂在 p 的下面，所以祖先只能取在 $1 \to p$ 这条路径上的点。这部分只跟 p 和 q 有关，而且能进一步拆成「跟 p 有关的 $d(p)$」加「跟 q 有关的 $d_2(q)$」两部分（见下）。
- **两端都在 T_2 里的废边**：就是「T_2 以 q 为根时的废边数」，只跟 q 有关，记为 $d_1(q)$。

（祖先在 T_2、后代在 T_1 是不可能的，T_2 底下没有 T_1 的点。）

于是 $f(T) = S + d(p) + d_2(q) + d_1(q)$，指数相加就是幂相乘，求和自动拆成三个式子的乘积：

$$\text{答案} = 2^S \times \Big(\sum_{p\in T_1} 2^{d(p)}\Big) \times \Big(\sum_{q\in T_2} 2^{d_1(q)+d_2(q)}\Big) + \big[\,|T_2| = 1\,\big] \times 2^S.$$

最后那一项就是情形二（T_2 是孤立点，且导览图里不连它）。它顺便也处理了 $n = 2$：$S = 0$，答案 $= 1 \times 1 \times 1 + 1 = 2$，和样例 2 一致。

其中两个小量的含义是：

- $d_2(q) = \#\{\,T_2$ 里编号比 q 大的点 $\}$：这是桥 (p, q) 这条树边自己贡献的废边数。
- $d(p) =$ 沿着 $1 \to p$ 的路径，每条边的下端点 c 都贡献 $\#\{T_2$ 里编号比 c 大的点 $\}$，累加起来。可以在 T_1 上一遍 DFS 递推：$d(1) = 0$，$d(c) = d(\text{父亲}) + \#\{T_2$ 里编号 $> c$ 的点 $\}$。

第四步：三个量各用什么工具算

每一块归根结底都是同一件事：**数「某棵子树里编号大于某个值的点有几个」**。这是经典的二维数点：一维是 DFS 序的区间（子树对应一段连续的 DFS 序），一维是编号的范围。做法是把每个询问按 DFS 序差分成两个前缀询问，然后按 DFS 序扫一遍，用**树状数组**维护「已经插入的点的编号」，询问时取 $\text{ask}(n) - \text{ask}(c)$。离线一遍 $O(n \log n)$。

S 和 $d(p)$、$d_2(q)$ 都可以这样直接算出来。麻烦的是 $d_1(q)$：它要求「T_2 以 q 为根时的废边数」，而 q 要取遍 T_2 的每一个点——这就是**换根 DP**。

换根的思路是逐条边地看。在 T_2 上先随便定一个根 rt，考虑一条边 (u, v)（v 是 u 的儿子）：

- 如果新的根 q 在 v 的子树**外面**，那 u 还是 v 的父亲，v 的子树也没变，这条边贡献「v 的子树里编号 $> v$ 的点数」；
- 如果 q 在 v 的子树**里面**，那父子关系**翻转**了，变成 v 是 u 的父亲，而「u 的子树」变成了原来 v 子树的补集，这条边贡献「v 的子树外面编号 $> u$ 的点数」。

两种情况都是「在一段（或两段）DFS 序区间里数编号大于某个值的点数」，而且「哪些 q 适用」也是一段连续的 DFS 序。所以每条边挂 $O(1)$ 个询问，仍然是离线树状数组扫一遍，把结果用差分累加到对应的 q 上，最后求前缀和还原出 d_1。整个算法 $O(n \log n)$。

```
void dfs3(int u) {                      // 在 T2 上挂换根用的询问
    for (int v : G[u]) if (v != fa1[u]) {
        int L = dfn1[v], R = dfn1[v] + sz1[v] - 1;
        Add(L, R, v, 1,   L-1 );        // q 在 v 子树外：数 v 子树内编号 > v
        Add(L, R, v, R+1, num1);
        Add(1,   L-1, u, L, R);         // q 在 v 子树内：数 v 子树外编号 > u
        Add(R+1, num1, u, L, R);
```

```
        dfs3(v);
    }
}
```

拿样例 1 手算一遍

$n=4$，风景路是 $(1,4)$ 和 $(2,3)$，所以 $T_1=\{1,4\}$、$T_2=\{2,3\}$。

- S：T_1 只有一条边 $(1,4)$，4 的子树只有它自己，没有编号比 4 大的点，所以 $S=0$。
- $\sum_p 2^{d(p)}$：$d(1)=0$；$d(4)=\#\{T_2\text{ 中编号 }>4\}=0$。和为 $1+1=2$。
- $\sum_q 2^{d_1(q)+d_2(q)}$：$q=2$ 时 $d_2=\#\{T_2\text{ 中编号 }>2\}=1$、$d_1=0$，贡献 $2^1=2$；$q=3$ 时 $d_2=0$、$d_1=0$，贡献 1。和为 3。
- $|T_2|=2\neq 1$，没有额外项。

答案 $=1\times 2\times 3=6$，和样例一致。其中 $p=1,q=2$、再加上废边 $(1,3)$ 的那张图，正是题面解释里给出的那个方案。

部分分怎么拿

- **测试点 $1\sim5$（$n\le 8$，20 分）**：直接枚举所有边集（$2^{n(n-1)/2}$ 种），每种都模拟一遍旅行，检查「所有风景路都走过」且「走过的非风景路 ≤ 1 条」。n 这么小的时候边数最多只有 $\binom{n}{2}$，枚举量完全可以承受。这份暴力只有十几行，而且和正解的思路完全独立，是拿来对拍的最佳选择。
- **测试点 $6\sim10$（$n\le 3000$，20 分）**：思路和正解完全一样，只是把数点换成暴力。最省事的写法是把树状数组的两个函数改掉：`add` 变成 $O(1)$ 的单点加，`ask` 变成 $O(n)$ 的循环求和，其他代码一个字都不用动，复杂度就变成 $O(n^2)$。
- **测试点 $11\sim15$（性质 A：城市 1 那块有 $n-1$ 个点，20 分）**：这意味着 T_2 只有一个孤立点 q。于是 $d_1(q)=d_2(q)=0$，第三个括号恒等于 1 ——**整个换根 DP 都不用写了！**而且因为只有一个 q，$d(p)$ 的递推退化成「沿路径累加 $[q>c]$」，一个计数器就够。剩下要做的只有算 S（一次子树数点）和别忘了加上情形二的 2^S。这一档直接砍掉了本题一半的难度，是最值得抢的部分分。
- **测试点 $16\sim20$（性质 B：两个连通块都是链，20 分）**：链上的「子树」就是一段后缀，于是所有的二维数点都退化成一维：S 从后往前扫一遍树状数组就能算（本质上是在数顺序对）；换根时以 q 为根的链就是左右两条臂，用两个方向的前缀和就能求出 $d_1(q)$，不用写 DFS 序 + 区间挂询问那一整套。
- **测试点 $21\sim25$（$n\le 10^6$，20 分）**：需要完整的 $O(n\log n)$ 做法。

容易写错的地方

1. **千万不要漏掉情形二**（$|T_2|=1$ 时答案要额外 $+2^S$）。$n=2$ 就是这种情形最小的例子。
2. **枚举儿子时必须按编号排序**。DFS 的规则是「编号最小优先」，如果你邻接表的顺序不对，算出来的 DFS 序和子树区间就全错了。记得 `sort(G[i])`。

3. **递归深度**。$n = 10^6$ 时如果风景边构成一条长链，递归会非常深。本地用 MinGW 编译要加 `-Wl,--stack,1073741824`；评测环境的栈空间等于内存限制（512 MiB），也可以改写成手工栈。

4. **读入速度**。$n = 10^6$ 时有一百万行输入，一定要 `ios::sync_with_stdio(false)` 或者手写快读。

5. **树状数组的清空和顺序**。S、d/d_2、d_1 这三个阶段共用一个树状数组，阶段之间要清零；而且第二阶段里树状数组装的是 T_2 的点（d 和 d_2 都靠它），顺序不能乱。

6. 常数：$n = 10^6$ 时大约有一千多万次树状数组操作和几百万个挂在 `vector` 上的询问，很不 cache 友好。真要卡常，可以把 `vector` 换成「计数 + 前缀和 + 一维数组」的静态邻接表。