四题完整题解

断边 / 删段 / 前缀 / 感染

从模型转换到正确性证明，再到可实现的算法。

题目	核心方法	时间复杂度（单组）
断边 cut	连续块代价 + 容量 DP	O(n)
删段 delete	前缀 Trie + 后缀 Trie 破坏式合并	O(26nm + m² + q)
前缀 lcp	重链下界 + 容斥多项式 + NTT	O(S + n log n)
感染 infect	带权直径 + 点分治球计数	O((n + q) log² n)

阅读说明

本文对应附件 otc_day8.pdf
中的四道题。复杂度均按单组数据计算；处理完一组后应释放或重置数据结构。前三题时限为 4 秒，感染为 5 秒，内存限制均为 1024 MiB。

每题包含模型、证明、算法和核心 C++14 代码。代码段用于说明关键转移与接口，并非四份可直接提交的完整程序；输入输出、多测清理、标准 NTT、LCA 和点分治框架的接口要求会明确说明。

前缀题所有计数均模
998244353。其余三题输出整数；断边代价、前缀最小代价、感染的带权距离均使用 64 位整数。

验证依据：附件样例，以及小规模随机数据上的暴力对照。验证范围和实现检查清单见末页。

断边：先决定连通块大小

题意转述

初始图是完全图。恰好删除 k 条边后，同一连通块中的点对 (i, j) 产生代价 $a_{|i-j|}$，不同块之间不产生代价。a 单调不降且为正数。要求输出每个 k 的最小代价，n ≤ 100。

令 M = n(n-1)/2，R = M-k。把“删除 k 条边”改写为“保留 R 条边”。只要连通块划分相同，总代价就相同，块内部具体保留哪些边不会改变代价。

一个大小为 s 的连通块，最低代价是多少？

将该块的点按编号排列为 $v_1 < v_2 < \cdots < v_s$。由于编号是整数，$v_j - v_i \geq j-i$。a 单调不降，故每个点对的代价至少为 a_{j-i}。

$w(s) = \sum_{d=1}^{s-1} (s-d)\, a_d$，其中 $w(1)=0$。

因此该块代价至少为 w(s)。当这 s 个点在数轴上连续时，每个不等式都取等号。对任意给定的块大小 $s_1, \cdots, s_c$，把它们依次放成连续区间，就能同时达到每个块的最低代价。

所以无需记录每个点属于哪个块，只需决定各块的大小。连续区间是最优解的一种实现形式，题目并没有要求连通块必须连续。

给定块大小，能保留多少条边？

大小为 s 的块要连通，至少需要 s-1 条边；至多可以保留 s(s-1)/2 条边。介于两者之间的每个整数都可实现：先放一棵生成树，再逐条加边。

$L = \sum (s_i - 1) = n - c$；$E = \sum s_i(s_i-1)/2$。

这份划分可实现 R 条保留边，当且仅当 $L \leq R \leq E$。

接下来的关键简化

DP 只记录最大容量 E，不记录最少边数 L。看起来这样会遗漏 $R \geq L$ 的限制，但可以证明：在 $E \geq R$ 的所有划分中，代价最小的划分一定满足 $L \leq R$。下一页给出证明。

断边：为什么容量 DP 就足够

证明最优划分不会缺少连通所需的边

假设某划分满足 E ≥ R，却有 L > R。因为都是整数，L-R ≥ 1。并且

$$E-R = \sum (s_i-1)(s_i-2)/2 + (L-R)。$$

取任意大小为 s ≥ 2 的块，把它拆成一个单点和一个大小为 s-1 的块。容量减少 s-1。由上式，原来的容量余量满足

$$E-R \geq (s-1)(s-2)/2 + 1 \geq s-1。$$

最后一个不等式对 s ≥ 2 成立，因为两边之差为 (s-2)(s-3)/2。因此拆分后仍有 E ≥ R。

但代价严格下降：$w(s)-w(s-1) = a_{\cdots}+\cdots+a_{s-1} > 0$。这与代价最小矛盾。故最优划分一定满足 L ≤ R，不用再记录 L。

动态规划

设 dp[v][e] 为：把 v 个点分成若干块，容量总和恰好为 e 时，最小的块代价之和。不可达状态为 INF，初始化 dp[0][0]=0。

$$dp[v+s][e+s(s-1)/2] = \min(dp[v+s][e+s(s-1)/2],\ dp[v][e]+w(s))。$$

枚举 1 ≤ s ≤ n-v。最终对 dp[n][e] 求后缀最小值，得到 $best[R] = \min_{e\geq R} dp[n][e]$。删除 k 条边的答案就是 best[M-k]。

复杂度与数值范围

状态数为 $O(n^3)$，每个状态枚举块大小，粗略上界 $O(n\ )$，空间 $O(n^3)$。实现时只枚举 e ≤ v(v-1)/2，实际转移量显著小于直接四重满循环。n=100 时适用。

总代价不超过 $M\cdot 10\ $，n=100 时不超过 4.95×10^{12}。使用 long long，INF 可取 $4\times 10^{1}\ $；跳过不可达状态，避免对 INF 加法。

样例核对

n=3，a=(1,4,4)。保留 3 或 2 条边时图仍连通，代价为 1+1+4=6；保留 1 条边时选相邻两点，代价为 1；保留 0 条边时为 0。按删除边数输出：6 6 1 0。

断边：核心实现

下面函数返回按删除边数 k=0,…,M 排列的全部答案。a 使用 1-based 下标。完整程序外层读取 c,T，再逐组调用即可。

```
vector<long long> solveCut(int n,
                           const vector<long long>& a) {
    int M = n * (n - 1) / 2;
    const long long INF = 4000000000000000000LL;
    vector<long long> w(n + 1), pref(n + 1);
    for (int s = 2; s <= n; ++s) {
        pref[s - 1] = pref[s - 2] + a[s - 1];
        w[s] = w[s - 1] + pref[s - 1];
    }
    vector<vector<long long>> dp(
        n + 1, vector<long long>(M + 1, INF));
    dp[0][0] = 0;
    for (int v = 0; v < n; ++v) {
        int bound = v * (v - 1) / 2;
        for (int e = 0; e <= bound; ++e) {
            if (dp[v][e] == INF) continue;
            for (int s = 1; v + s <= n; ++s) {
                int ne = e + s * (s - 1) / 2;
                dp[v + s][ne] = min(dp[v + s][ne],
                                    dp[v][e] + w[s]);
            }
        }
    }
    vector<long long> best = dp[n];
    for (int e = M - 1; e >= 0; --e)
        best[e] = min(best[e], best[e + 1]);
    vector<long long> ans(M + 1);
    for (int k = 0; k <= M; ++k)
        ans[k] = best[M - k];
    return ans;
}
```

常见误区

不能直接选择 R 个最便宜的点对。被选择的边可能连接出一个大连通块，而块内没有直接边的点对也会产生代价。计费对象是“连通点对”，不是“保留的边”。

不能把 dp[n][R] 直接作为答案。容量可以大于 R，块内删掉一部分非树边后仍能保持连通。因此最终要取后缀最小值。

此处对“最少边数限制”的省略依赖 a_i 为正数。证明中的拆块操作严格降低代价，正好来自这一条件。

删段：答案是前缀与后缀的组合数

题意转述

给出 n 个长度均为 m 的字符串。每次删除相同区间 [l,r]，问剩余字符串有多少种。n,m ≤ 1000，q ≤ 10 ，各次询问独立。

令 p=l-1，d=m-r。每个字符串保留长度 p 的前缀和长度 d 的后缀。因为 p 固定、d 固定，拼接后相等，当且仅当前缀相等且后缀相等。

> ans[p][d] = 不同二元组（长度 p 的前缀，长度 d 的后缀）的数量。
>
> 询问 [l,r] 的答案为 ans[l-1][m-r]。

虽然只会查询 p+d < m 的位置，但预处理整个 (m+1)×(m+1) 表更方便。空前缀、空后缀分别对应 p=0、d=0。

按前缀分组

建一棵前缀 Trie。深度 p 的每个节点对应一类“长度 p 的前缀相同”的字符串。对这一组字符串，统计不同长度 d 后缀的数量，最后对所有组求和，就得到 ans[p][d]。

将字符串反转后建 Trie。反串的长度 d 前缀，就是原串的长度 d 后缀。所以某组字符串对应的后缀 Trie，在深度 d 有多少个节点，该组就有多少种长度 d 的后缀。

为什么不能给每个前缀节点都做一遍统计？

前缀 Trie 最多有 O(nm) 个节点。若每个节点都遍历 m 个后缀长度，可能达到 O(nm²)。n=m=1000 时规模过大。

但从前缀长度 p-1 变成 p 时，只有真正分叉的组会发生变化。单儿子节点对应的字符串集合不变，不会改变答案。我们只记录“分组细化带来的增量”，再按 p 累加。

重复输入字符串

相同原串在任何删除操作后都相同，因此可以在前缀 Trie 叶子处自然去重。每个不同原串只建一份独立的后缀路径；重复到达同一叶子的输入不再建新路径。

删段：合并时顺便统计答案增量

从叶子向上合并

一个前缀 Trie 叶子对应一个不同原串，给它建立一条完整的反串 Trie 路径。内部节点的后缀 Trie 是所有儿子后缀 Trie 的集合并。

合并两棵后缀 Trie 时，按相同字符递归。如果某位置只有一边有节点，直接移接子树；如果两边都有节点，它们代表同一个后缀，合并后只能保留一个节点。

重合节点正好就是分组细化的增量

设前缀节点 u 的深度为 p-1。它的儿子是长度 p 的前缀组。对后缀长度 d，设这些儿子各自的不同后缀数为 $c_1,\cdots,c_t$，父组的不同后缀数为 c。

分组细化的增量 $= c_1+\cdots+c_t-c$。

从儿子集合依次合并成父集合时，每消掉一个深度 d 的重合节点，就使节点总数减少 1。被消掉节点的数量，恰好等于上述差值。

因此：在 u 处合并儿子的后缀 Trie，每遇到一个深度 d 的重合节点，就执行 delta[p][d]++。所有节点处的增量都加进同一个表。后缀根深度为 0，也必须计数。

得到答案表

全部合并结束后，前缀根对应所有字符串的后缀集合。遍历它的后缀 Trie，深度 d 的节点数就是 ans[0][d]。然后累加：

ans[p][d] = ans[p-1][d] + delta[p][d]，$1 \leqslant p \leqslant m$。

证明可以用归纳：ans[p-1][d] 是所有父组的贡献；加上每个父组拆成儿子组时的变化，就得到深度 p 所有组的贡献。

为什么合并总量是线性的？

开始时，每个不同原串独立建立 m+1 个后缀节点，总计 O(nm)。每次“两边都有节点”的递归合并，会永久弃用其中一个节点。一个被弃用的节点以后不能再次参与合并，因此这种递归调用总计 O(nm)。

单侧非空时直接移接，不遍历整棵子树。所以无需小并大；关键是破坏式合并，父节点接管儿子的数据，合并后不再使用儿子的旧结构。每次重合最多检查 26 个儿子。

时间 $O(26nm+m^2+q)$，空间 $O(26nm+m^2)$。采用整型节点编号及固定儿子数组，符合 1024 MiB 内存限制。

删段：核心实现

P 是前缀 Trie，含 ch[26]、depth、root；S 是后缀 Trie，含 ch[26]。节点 0 表示空指针。插入时为每个不同原串对应的前缀叶子建立独立后缀路径。下面展示合并和增量累加。

```
int mergeSuffix(int a, int b, int d, int p) {
    if (!a || !b) return a ? a : b;
    ++delta[p][d];
    for (int c = 0; c < 26; ++c)
        S[a].ch[c] = mergeSuffix(S[a].ch[c],
                                 S[b].ch[c], d + 1, p);
    return a;  // b is discarded permanently
}

int solvePrefix(int u) {
    if (P[u].depth == m) return P[u].root;
    int rt = 0;
    for (int c = 0; c < 26; ++c) {
        int v = P[u].ch[c];
        if (!v) continue;
        int child = solvePrefix(v);
        rt = mergeSuffix(rt, child, 0, P[u].depth + 1);
    }
    return rt;
}

void countBase(int u, int d) {
    if (!u) return;
    ++ans[0][d];
    for (int c = 0; c < 26; ++c)
        if (S[u].ch[c]) countBase(S[u].ch[c], d + 1);
}

void finish(int prefixRoot) {
    int suffixRoot = solvePrefix(prefixRoot);
    countBase(suffixRoot, 0);
    for (int p = 1; p <= m; ++p)
        for (int d = 0; d <= m; ++d)
            ans[p][d] = ans[p - 1][d] + delta[p][d];
    // query(l, r) returns ans[l - 1][m - r]
}
```

实现检查

后缀路径必须包含空后缀根，这样 d=0 才正确。不同原串的初始后缀路径不能共享节点，否则“弃用一个节点”的复杂度证明和增量含义都会失效。

先完成节点构建再合并，或确保 vector 扩容时没有悬空引用。每组清空两个节点池、delta、ans。递归深度至多 m+1，可用显式栈改写。

附件样例的四个答案依次是 3、3、2、1；删除整个串时应返回 ans[0][0]=1。

前缀：用重子树同时达到所有下界

把 LCP 拆成经过的 Trie 节点

建二进制 Trie。每个输入串在结束节点打终止标记；不能只把叶子视为输入串，因为某个串可能是另一个串的前缀。记 s_v 为 v 子树内的输入串数量。

两个字符串的 LCP 等于它们共同经过的非根 Trie 节点数。对一个排列，设 e_v 为相邻两串都在 v 子树内的次数，那么总代价为所有非根节点的 e_v 之和。

一个大小为 s 的集合至少产生多少次内部相邻？

集合外有 n-s 个串，最多把集合内元素分成 n-s+1 段。每段长度为 z 时贡献 z-1 次内部相邻。因此

$e_v \geqslant \max(0,\ 2s_v-n-1)$。

把 $s_v > (n+1)/2$ 的节点称为重节点。两个不相交子树不可能都包含这么多串，所以所有重节点必定位于一条从根向下的链上。根只用于定位，不计入代价。

找到最深重节点，划分四类字符串

令 v 为最深重节点；如果没有非根重节点，就令 v 为根。设它的 0 儿子、1 儿子子树大小分别为 a,b；v 自己是否终止记为 $t\in\{0,1\}$；v 子树之外的串数为 h。于是 n=a+b+t+h。

构造排列，要求：A 类内部不相邻，B 类内部不相邻，H 类内部不相邻，首尾都不能是 H 类。终止串 T 至多一个，不会内部相邻。

这样，v 内的各个轻子树都没有内部相邻，故所有轻节点都达到下界 0。对任何重节点 u，u 外部元素是 H 的子集；它们互不相邻且不在首尾，所以恰好把 u 内元素分成 $n-s_u+1$ 段，达到其下界。

这些要求确实能实现

给 H 类额外添加一个虚拟元素，并考虑圆环排列。各类大小均不超过圆环总大小 (n+1) 的一半：a,b 由最深重节点的定义保证，h+1 由 v 的重性保证，$t\leqslant 1$。

使用经典圆环排列结论：若每类数量不超过总数的一半，就能把同类元素在圆环上隔开。删除 H 类的虚拟元素，得到首尾非 H 且同类不相邻的线性排列。这个结论也可由完全多部图的最小度至少为顶点数一半、存在 Hamilton 圈得到。

最小代价 $=\ \Sigma_{v\neq root}\ \max(0,2s_v-n-1)$。

前缀：用容斥统计最优排列

哪些排列恰好是最优排列？

各节点下界都能同时达到，所以最优排列必须逐个达到下界。对最深重节点 v，这等价于 H 类互不相邻且不在首尾；对 v 的两个儿子，这等价于 A 类、B 类各自内部不相邻。

因此上页的四类排列要求不仅充分，而且必要。各类中的具体字符串仍然不同；先数类别序列，再乘 a!b!h!，T 类至多一个。

对同类相邻进行容斥

一类有 s 个元素。把它们按出现顺序编号，第 i 与第 i+1 个之间有 s-1 个潜在粘合位置。容斥选择其中 s-k 个位置粘合，就把这一类缩成 k 个块，系数为 $(-1)^{s-k}\cdot C(s-1,k-1)$。

排列各类块时要除以各类块数的阶乘。因此定义

$P_0(z)=1$；

$P_s(z)=\sum_{k=1}^{s}(-1)^{s-k}C(s-1,k-1)\,z^k/k!$，$s\geq 1$。

T 类只有 t 个元素，t=0 或 1。令 $Q(z)=P_a(z)P_b(z)z^t$。设其中一个项表示非 H 块总数 L；H 类缩成 j 个块，总块数 K=L+j。

首尾非 H 如何计入？

不限制端点时，类别块的排列数为 K! 除以各类块数阶乘。任意两个端点都来自非 H 块的比例是 L(L-1)/(K(K-1))。因此分子变成

$L(L-1)(K-2)!$。

这里把块暂时看作不同对象选首尾，再除回同类块数的阶乘。粘合同类块不会改变“首尾是否属于 H”，所以该端点约束可以直接放入容斥。

用两次卷积算完

若 $Q(z)=\sum q_L z^L$，令 $Q_0(z)=\sum L(L-1)q_L z^L$，然后计算 $R(z)=Q_0(z)P_h(z)$。

最优排列数 = $a!b!h!\cdot\sum_{K=2}^{n}[z^K]R(z)\cdot(K-2)!$，模 998244353。

所有多项式次数总和不超过 n，使用 NTT 求两个卷积即可。预处理阶乘和逆阶乘后，构造每个 P 只需线性时间。

前缀：核心实现与边界

MOD=998244353。fac、ifac 为阶乘与逆阶乘；C(N,K) 用它们计算；mul 是标准 NTT 卷积，空系数按 0 处理。数组 ch、sz、terminal 属于二进制 Trie。

```
vector<int> makeP(int s) {
    if (s == 0) return {1};
    vector<int> p(s + 1);
    for (int k = 1; k <= s; ++k) {
        int x = 1LL * C(s - 1, k - 1) * ifac[k] % MOD;
        if ((s - k) & 1) x = x ? MOD - x : 0;
        p[k] = x;
    }
    return p;
}

pair<long long, int> solveLCP(int n) {
    int v = 0;
    long long cost = 0;
    while (true) {
        int next = -1;
        for (int c = 0; c < 2; ++c) {
            int u = ch[v][c];
            if (u && 2LL * sz[u] > n + 1) next = u;
        }
        if (next == -1) break;
        v = next;
        cost += 2LL * sz[v] - n - 1;
    }
    int a = ch[v][0] ? sz[ch[v][0]] : 0;
    int b = ch[v][1] ? sz[ch[v][1]] : 0;
    int t = terminal[v], h = n - a - b - t;
    auto q = mul(makeP(a), makeP(b));
    if (t) q.insert(q.begin(), 0);
    for (int L = 0; L < (int)q.size(); ++L)
        q[L] = 1LL * q[L] * L % MOD * (L - 1 + MOD) % MOD;
    auto r = mul(q, makeP(h));
    long long ways = 0;
    for (int K = 2; K < (int)r.size(); ++K)
        ways = (ways + 1LL * r[K] * fac[K - 2]) % MOD;
    ways = ways * fac[a] % MOD * fac[b] % MOD;
    ways = ways * fac[h] % MOD;
    return {cost, (int)ways};
}
```

建 Trie 时节点按父先子后的顺序创建，可倒序汇总 sz，避免长度达到 10 的递归。NTT 使用原根 3；卷积长度取不小于结果长度的最小 2 的幂。时间 O(S+n log n)，空间 O(S+n)。

附件三个样例结果分别为 (0,2)、(1,8)、(1,2)。要保留终止标记 t，否则输入串为另一个串前缀时会漏掉一类元素。

感染：整个区间只需保留两个病毒

把感染条件写成最大值

病毒 i 在点 v_i、时刻 t_i 出现。点 u 在时刻 T 被该病毒感染，当且仅当 $t_i+\mathrm{dist}(v_i,u) \le T$。因此区间 [l,r] 的询问是统计

$$\max_{l\le i\le r} \{t_i+\mathrm{dist}(v_i,u)\} \le T \text{ 的点 } u。$$

即使询问时刻早于某病毒出现时刻，这个不等式也会自然失败，无需依赖时刻单调性。

定义带权直径

对一个病毒集合，选一对病毒 a,b，使下式最大。允许 a=b。

$$D = t_a+\mathrm{dist}(v_a,v_b)+t_b。$$

允许重复端点非常重要：只有一个病毒时，$D=2t_a$；如果一个病毒的出现时刻非常大，它也可能完全支配其他病毒。

直径端点为什么代表整个集合？

想象在原树的每个病毒出现点挂两片新叶子，每条挂边长度为该病毒的出现时刻 t_i。到这些叶子的距离恰好是 $\mathrm{dist}(u,v_i)+t_i$。挂两片是为了把 a=b 的情况实现成两片不同叶子。

在这棵非负边权树中，病毒集合的带权直径就是挂叶子的普通直径。树的直径有一个标准性质：对任意点 u，最远点一定可以取两个直径端点之一。

$$\max_i\{t_i+\mathrm{dist}(v_i,u)\}$$
$$= \max(t_a+\mathrm{dist}(v_a,u),\ t_b+\mathrm{dist}(v_b,u))。$$

这个性质可从直径中点理解：树上每个分支的延伸长度不超过它通向直径端点的剩余长度，否则会产生更长直径。故任意点到某分支末端的距离，不会同时大于到两个直径端点的距离。

线段树维护区间直径

每个节点只存两个病毒编号。合并两段集合时，新直径端点只需从两边各自的两个端点中选，检查这四个候选点的所有配对，包括重复配对即可。每次合并只需常数次距离计算。

可以先读完整组操作，按出现顺序收集全部病毒，再建立静态线段树。题目保证查询 [l,r] 中的病毒已经出现，所以后来的病毒不会落入该查询区间，离线建树不会改变答案。

感染：两个球的交集是什么形状

确定中心位置

设端点位置为 A=v_a、B=v_b，d=dist(A,B)，权重为 α=t_a、β=t_b。D=α+β+d。由于 D 至少为 2α 和 2β，有 |α−β| ≤ d。

因此存在位于路径 A→B 上的中心 C，使 α+dist(A,C)=β+dist(B,C)。从 A 到 C 的距离为

x = (d+β−α)/2；

max(α+dist(A,u), β+dist(B,u)) = D/2 + dist(C,u)。

证明：将 u 投影到路径 A→B，设投影距 A 为 y，u 到投影为 z。左边为 z+max(α+y, β+d−y)，正好等于 z+D/2+|y−x|。

所以所有病毒的公共感染区域，就是以 C 为中心、半径 T−D/2 的树上球。C 可能在一个顶点，也可能在一条边的中点。

D 为偶数：中心是顶点

因为 d+β−α 与 D 同奇偶，D 为偶数时 x 是整数。用树上第 k 个路径点查询找到 C，令 R=T−D/2。

答案 = Ball(C,R)。R < 0 时答案为 0。

这里 Ball(v,R) 表示原树中距离 v 不超过 R 的顶点数，新挂的虚拟叶子从不参与计数。

D 为奇数：中心是边中点

此时 C 位于相邻顶点 C_u、C_v 的中点。令 R=T−(D+1)/2，则原树顶点满足感染条件，当且仅当到 C_u、C_v 中较近一个的距离不超过 R。

区域 = Ball(C_u,R) ∪ Ball(C_v,R)。

为避免重复计数，将这条边按固定根方向记为 parent P、child Q。Ball(P,R) 已经覆盖 Q 子树里距 Q 不超过 R−1 的点；另一边的球只会额外覆盖 Q 子树里距 Q 恰好为 R 的点。

答案 = Ball(P,R)

+ #{u 在 Q 子树内：depth(u)=depth(Q)+R}。

R=0 时，这个公式得到两个端点。R<0 时直接返回 0。不能把两个球的大小直接相加，否则交集会重复。

感染：如何快速数球内顶点

点分治维护距离表

对原树做点分治。对于每个重心 c，保存当前分治连通块中所有点到 c 的距离，排序为 all[c]。同时对去掉 c 后的每个分支，保存该分支所有点到 c 的距离表 sub[c][branch]。

对每个原树点 u，记录它在各层分治中的重心祖先 c、距离 d=dist(u,c)，以及 u 位于 c 的哪个分支。若 u=c，则分支记为空。

$$\mathrm{Ball}(u,R) = \sum_{\text{重心祖先 } c} \big(\mathrm{count}(\mathrm{all}[c], R-\mathrm{dist}(u,c)) - \mathrm{count}(\mathrm{sub}[c][\mathrm{branch}(u)], R-\mathrm{dist}(u,c))\big)。$$

count(list,K) 用 upper_bound 统计距离 ≤K 的项数；K<0 时为 0。如果 u=c，不做减法。

为什么正确？在一层重心处，u 与不在同一分支的点之间路径经过 c，可以用两段距离之和计算。与 u 同分支的点先减掉，留到下一层处理。每个点恰好在它与 u 第一次被分开的层贡献一次。

重心祖先只有 O(log n) 个，每个表二分 O(log n)，故一次 Ball 查询 $O(\log^2 n)$。距离表总长度 O(n log n)，空间也是 O(n log n)。

子树内固定深度的点数

将原树根定为 1，DFS 得到 depth、tin、tout，约定子树 Euler 区间为 [tin,tout)。对每个深度 d，存储深度为 d 的点的 tin，按 Euler 访问顺序排列，天然递增。

$$\mathrm{Level}(Q,d) = \mathrm{lower_bound}(\mathrm{pos}[d], \mathrm{tout}[Q]) - \mathrm{lower_bound}(\mathrm{pos}[d], \mathrm{tin}[Q])。$$

若目标深度不存在，则返回 0。这样，奇数 D 时的额外一层只需 O(log n) 时间。

LCA 与路径第 k 个点

倍增预处理 LCA，距离为 depth(u)+depth(v)-2depth(LCA)。求路径 A→B 上距 A 为 k 的顶点：若 k 不超过 A 到 LCA 的长度，就向上跳 A；否则从 B 向上跳 d-k。

总复杂度与工程注意事项

线段树查询做 O(log q) 次合并，每次距离查询 O(log n)。总时间 $O((n+q)\log^2(n+q))$，按 n,q 同阶记为 $O((n+q)\log^2 n)$；空间 O(n log n+q)。距离表排序预处理包含在该上界中。

原树可能是一条长链，普通递归 DFS 深度可能达到 10^5。LCA、Euler 序以及点分治中的连通块遍历建议采用显式栈；点分治本身的重心递归深度只有 O(log n)。

感染：查询核心实现

接口：virus[i]={v,t}；dist(u,v) 为树距离；kth(A,B,k) 返回路径上距 A 为 k 的点；ball(v,R) 为点分治球计数；level(Q,d) 为 Q 子树内深度等于 d 的点数。所有时间相关量用 long long。

```
struct Dia { int a, b; }; // {0,0} denotes empty

long long weight(int a, int b) {
    return virus[a].t + virus[b].t
         + dist(virus[a].v, virus[b].v);
}

Dia mergeDia(Dia x, Dia y) {
    if (!x.a) return y;
    if (!y.a) return x;
    int c[4] = {x.a, x.b, y.a, y.b};
    Dia best = x;
    long long value = weight(best.a, best.b);
    for (int i = 0; i < 4; ++i)
        for (int j = i; j < 4; ++j) {
            long long now = weight(c[i], c[j]);
            if (now > value) {
                value = now; best = {c[i], c[j]};
            }
        }
    return best;
}

long long answer(Dia z, long long T) {
    int A = virus[z.a].v, B = virus[z.b].v;
    long long alpha = virus[z.a].t;
    long long beta = virus[z.b].t;
    long long d = dist(A, B), D = alpha + beta + d;
    long long R = T - (D + 1) / 2;
    if (R < 0) return 0;
    int k = (int)((d + beta - alpha) / 2);
    int c1 = kth(A, B, k);
    if ((D & 1) == 0) return ball(c1, R);
    int c2 = kth(A, B, k + 1);
    int P = c1, Q = c2;
    if (depth[P] > depth[Q]) swap(P, Q);
    return ball(P, R) + level(Q, depth[Q] + R);
}
```

线段树与边界

第 i 个叶子存 {i,i}，空区间使用 {0,0}。每次询问先得到 [l,r] 的 Dia，再调用 answer。叶子允许重复端点，因此只有一个病毒、所有病毒在同一点出现等情况都不需要特殊分支。

附件样例输出 4、1、0。查询时间早于出现时刻、球半径超过树直径、中心位于边中点，都是必要的测试情况。

验证与实现清单

小规模验证结果

题目	对照方法	本次结果
断边	25 组随机 n≤6；枚举全部保留边集合，与容量 DP 对照全部 k	全部一致
删段	100 组随机 n≤10,m≤8；枚举全部删除区间，与前缀分组增量公式对照	全部一致
前缀	100 组随机 n≤7；枚举全部排列，与最小值及容斥计数公式对照	全部一致
感染	100 组随机树 n≤12；逐点比较所有病毒最大到达时刻与直径两端，并核对整数/半整数中心的区域	全部一致

这些检查验证了算法的核心结论与公式。本文展示的是核心代码，未将四份完整提交程序编译运行，也未进行最大数据的时限实测；实际提交前仍需补全所列基础模块并运行样例和压力测试。

建议的提交前检查

断边：确认计费按连通块，末尾求后缀最小值，输出顺序为 k 从小到大。n=1 时只输出一个 0。

删段：确认重复原串去重、后缀根深度为 0、合并后旧儿子结构不再使用；核对 p=l-1、d=m-r。

前缀：确认输入串的终止标记、子树计数包含终止串、所有非根重节点代价都累加；多项式负系数归一化，模乘使用 64 位中间值。

感染：确认病毒编号按出现顺序、直径允许重复端点、查询时刻不要求单调、奇数直径的两个球不重复计数；depth 与 Euler 序均来自同一固定根。

完整程序的输入输出约定

每题第一行都是 c,T，c 为测试点编号，不是数据组数。每组按各题格式读取。若评测沿用附件的文件输入输出，需要分别使用
cut.in/out、delete.in/out、lcp.in/out、infect.in/out；若平台改为标准输入输出，移除 freopen。

所需标准模块

前缀需要模幂、阶乘/逆阶乘和 NTT 卷积；感染需要邻接表、倍增 LCA、Euler 序、点分治距离表以及区间直径线段树。两题的核心接口及其含义已在正文明确列出。

题面来源：用户提供的 otc_day8.pdf（12 页）。本题解仅覆盖附件四题，不包含此前讨论的矩阵计数题。