

主题

2020—2025 的第四题。

T4 的特点

- 压轴题，落在省选/NOI 难度：题目模型更深，先抽象、再套一个数据结构或数学工具
- 六年 T4 的形态：二分答案 + 树上贪心、计数 DP、树上 DP、平面图最短路、博弈归纳、数据结构优化
- 分差在「能不能把题意翻译成已知模型」，翻译对了代码量反而可控

T4 (2020—2025)

题目	难度	通过率	知识点
2023 T4 种树	提高+/省选-	18.54%	二分答案、树上贪心
2025 T4 员工招聘	省选/NOI-	25.38%	计数 DP
2022 T4 数据传输	省选/NOI-	20.64%	树上 DP、倍增
2021 T4 交通规划	省选/NOI-	19.97%	平面图、最短路
2020 T4 贪吃蛇	省选/NOI-	19.95%	博弈、归纳
2024 T4 擂台游戏	NOI/NOI+	14.57%	线段树、优化

2023 T4 种树

知识点：二分答案、树上贪心

难度：提高+/省选-

题目

一片森林有 n 片地块，由 $n - 1$ 条道路连成树，1 号地块是入口。每天可选一个**未种树且与已种树地块直接邻接**的地块种一棵高 0 米的树（第 1 天只能在 1 号种）。

第 i 号地块的树在第 x 天（ x 从第 1 天算起）长高 $\max(b_i + x \cdot c_i, 1)$ 米。求使每片地块的树都长到不低于 a_i 米的最少天数。

限制

- $1 \leq n \leq 10^5$, $1 \leq a_i \leq 10^{18}$, $1 \leq b_i \leq 10^9$, $0 \leq |c_i| \leq 10^9$
- 保证存在不超过 10^9 天的方案

二分答案 + 最晚种植时间

思路

二分答案天数 D 。对每个地块 i ，种得越早长得越高，所以存在一个**最晚种植日** ddl_i ：在第 ddl_i 天及之前种，到第 D 天才能长到 a_i 。

- $\text{calc}(i, s, t)$ ：第 i 地块从第 s 天到第 t 天的总生长量 $\sum_{x=s}^t \max(b_i + xc_i, 1)$ 。
- $c_i \geq 0$ 或 $b_i + c_it > 0$ 时是等差数列，直接用梯形公式；否则在 t_0 处分段： $[s, t_0]$ 是等差， $[t_0 + 1, t]$ 每天长 1。
- 对每个地块二分出 ddl_i ($\text{calc}(i, \text{mid}, D) \geq a[i]$ 时往右找更晚的)。

树上调度

贪心

种植顺序必须父先子后（以 1 为根）。从**叶子**开始，用大根堆按 ddl 从大到小取：给 ddl 最大的叶子安排 $\min(\text{当前最大可用日}, ddl)$ ，可用日再减 1；父节点等所有子节点安排完才入堆。

- 若某步可用日耗尽或某地块 $ddl < 1$ ，说明 D 天不够。
- 二分下界取 n （至少 n 天种完 n 棵树），上界 10^9 。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 1e5 + 5;
4 typedef long long ll;
5 typedef __int128 i128;
6
7 vector<int> g[maxn];
8 int n, parent[maxn];
9 ll a[maxn], b[maxn], c[maxn];
10 int ddl[maxn];
11
12 i128 calc(int i, int s, int t) {
13     if (c[i] >= 0 || b[i] + c[i] * t > 0) // 全程等差数列
14         return (i128)(b[i] + (ll)s * c[i] + b[i] + (ll)t * c[i]) * (t - s + 1) / 2;
15     int t0 = (b[i] - 1) / -c[i]; // 最后一次长 >=1 的天
16     if (s > t0) return t - s + 1;
17     return (i128)(b[i] + (ll)s * c[i] + b[i] + (ll)t0 * c[i]) * (t0 - s + 1) / 2 + (t - t0);
18 }
19
20 bool check(int D) {
21     for (int i = 1; i <= n; i++) {
22         int l = 0, r = D + 1; // 二分最晚种植日
23         while (l + 1 < r) {
24             int mid = (l + r) / 2;
25             if (calc(i, mid, D) >= a[i]) l = mid; else r = mid;
26         }
27         if (l == 0) return false;
28         ddl[i] = l;
29     }
30     priority_queue<pair<int, int>> q; // 大根堆按 ddl
31     vector<int> deg(n + 1);
32     for (int i = 2; i <= n; i++) deg[parent[i]]++;
33     for (int i = 1; i <= n; i++) if (deg[i] == 0) q.push({ddl[i], i});
34     int t = 0;
35     while (!q.empty()) {
36         auto p = q.top(); q.pop();
37         if (t == 0) return false;
38         t = min(t, p.first) - 1;
39         int pa = parent[p.second];
40         if (--deg[pa] == 0) q.push({ddl[pa], pa});
41     }
42     return true;
43 }
44
45 void dfs(int u, int p) {
46     parent[u] = p;
47     for (int v : g[u]) if (v != p) dfs(v, u);
48 }
49
50 int main() {
51     ios::sync_with_stdio(false); cin.tie(0);
52     cin >> n;
53     for (int i = 1; i <= n; i++) cin >> a[i] >> b[i] >> c[i];
54     for (int i = 0; i < n - 1; i++) { int u, v; cin >> u >> v; g[u].push_back(v); g[v].push_back(u); }
55     dfs(1, 0);
56     int l = n - 1, r = 1e9;
57     while (l + 1 < r) { int x = (l + r) / 2; if (check(x)) r = x; else l = x; }
58     cout << r << '\n';
59 }
```

实现要点

- `calc` 的 `t0` 是 $b_i + t_0 c_i \geq 1$ 的最后一天, $c_i < 0$ 时生长先降后托底为 1。
- `ddl_i` 二分: `calc(i, mid, D) >= a[i]` 时 `l = mid` (尽量晚种), `l == 0` 表示第 1 天种都不够。
- 调度用「反拓扑 + 大根堆」: 叶子先安排、`ddl` 大的先安排, `t` 从 D 往前减, 模拟「从后往前排种植时间」。
- 生长量用 `__int128` 防溢出 (a_i 到 10^{18})。
- 答案在 $[n, 10^9]$ 之间二分。

部分分

测试点	$n \leq$	特殊性质	拿法
1	20	A	暴搜
2—4	20	无	暴搜
5—6	500	A	枚举天数
7—8	10^5	A	二分 + 贪心 ($c_i = 0$)
9—10	10^5	B	链上贪心
11—13	10^5	C	链上贪心
14—16	10^5	D	菊花图贪心
17—20	10^5	无	进阶做法

特殊性质：A $c_i = 0$ ；B $u_i = i, v_i = i + 1$ ；C 任意地块度数不超过 2；D $u_i = 1$ 。

这道题的难点是「最晚种植时间 + 树上父先子后的调度」两层结合，二分答案把它们串起来。

2025 T4 员工招聘

知识点：计数 DP

难度：省选/NOI-

题目

n 人应聘，第 i 天面试题的难度为 $s_i \in \{0, 1\}$ (0 表示难，无人能做出；1 表示易，所有人都能做出)。面试官按排列 p 的顺序每天面试一人。编号 j 的人耐心上限为 c_j ：若他面试前已被**拒绝或放弃**的人数不少于 c_j ，则他也放弃。求能录用至少 m 人的排列 p 的数量，对 998244353 取模。

限制

- $1 \leq m \leq n \leq 500$, $s_i \in \{0, 1\}$, $0 \leq c_j \leq n$

未录用与耐心

设难题 ($s_i = 0$) 共有 H 套。排在难题位置的人若没放弃就被拒绝，所以录用人数至多 $n - H$ ；要录用 $\geq m$ 必须先有 $H \leq n - m$ 。

观察

设 $B(x)$ 为位置 $1 \dots x$ 中「拒绝或放弃」的人数，它单调不减。位置 x 上耐心为 c 的人放弃当且仅当 $B(x - 1) \geq c$ 。于是耐心 c 的人放弃，当且仅当他排在第 $c + 1$ 个及以后的「拒绝或放弃」位置。

因此「拒绝或放弃」的总数就是解题的全局量：它不能超过 $n - m$ 。

按耐心分层 DP

按耐心 c 从小到大分层 ($i = 0, 1, 2, \dots$), $\text{cnt}[i]$ 是耐心恰好为 i 的人数, $\text{sum}[i]$ 是耐心 $< i$ 的人数。

状态

$f[i][j][k]$: 已经确定了 i 个「放弃」位置, 第 i 个在位置 j ; 耐心 $< i$ 的人中有 k 个排在位置 j 之后 (具体位置未定)。

从 $f[i][j][k]$ 推下一步: 耐心 $= i$ 的 $\text{cnt}[i]$ 人里, 有 x 个排在 j 之后、其余排在 j 之前。则耐心 $\leq i$ 的人中排在 j 之后共有 $k + x$ 个, 第 $i + 1$ 个放弃位置 p 只能落在 j 之后、下一个难题位置 q 之前。

转移与差分优化

- 选 x 个人排到 j 之后、其余排到前面并占位，系数为 $\binom{cnt_i}{x} \cdot (cnt_i - x)! \cdot \binom{j - (sum_i - k)}{cnt_i - x}$ 。
- 若 $k + x > 0$ ：从 $k + x$ 人中选一个放到第 $i + 1$ 个放弃位置 $p \in (j, q]$ ，系数 $k + x$ ，对 $f[i + 1][p][k + x - 1]$ 做区间加。
- 否则第 $i + 1$ 个放弃位置直接落到 q 之后，系数 1。
- 区间加用差分数组 `g`，每层结束后前缀和 + 滚动到 `f`。

最终 `ans` 累加所有「不再出现新的放弃、且难题都在最前面」的合法终态。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int mod = 998244353;
4 typedef long long ll;
5
6 int n, m, cnt[505], sum[505];
7 ll f[505][505], g[505][505], fact[505], choose[505][505];
8
9 void range_add(int j1, int j2, int k, ll x) { g[j1][k] += x; g[j2][k] -= x; }
10
11 int main() {
12     ios::sync_with_stdio(false); cin.tie(0);
13     string s; cin >> n >> m >> s;
14     int H = 0; for (char c : s) H += (c == '0');
15     vector<int> next_hard(n + 1); next_hard[n] = n;
16     for (int i = n - 1; i >= 0; i--) next_hard[i] = (s[i] == '0') ? i : next_hard[i + 1];
17     for (int i = 0; i < n; i++) { int c; cin >> c; cnt[c]++; }
18     for (int i = 0; i <= n; i++) sum[i + 1] = sum[i] + cnt[i];
19     if (H > n - m) { cout << 0 << '\n'; return 0; }
20     fact[0] = 1; for (int i = 1; i <= n; i++) fact[i] = fact[i - 1] * i % mod;
21     for (int i = 0; i <= n; i++) { choose[i][0] = 1; for (int j = 1; j <= i; j++) choose[i][j] = (choose[i - 1][j] + choose[i - 1][j - 1]) % mod; }
22     ll ans = 0;
23     f[0][0] = 1;
24     for (int i = 0; i <= n - m; i++) {
25         for (int j = 0; j <= n; j++)
26             for (int k = 0; k <= sum[i]; k++) {
27                 if (f[j][k] == 0) continue;
28                 for (int x = 0; x <= cnt[i]; x++) {
29                     ll coeff = choose[cnt[i]][x] * fact[cnt[i] - x] % mod * choose[j - (sum[i] - k)][cnt[i] - x] % mod;
30                     int q = next_hard[j];
31                     if (k + x > 0) {
32                         ll add = coeff * (k + x) % mod * f[j][k] % mod;
33                         range_add(j + 1, q + 2, k + x - 1, add); // p ∈ (j, q]
34                     }
35                     range_add(q + 1, q + 2, k + x, coeff * f[j][k] % mod); // 全放到 q 之后
36                     if (k + x == 0 && q == n) {
37                         ans = (ans + coeff * f[j][k] % mod * fact[n - sum[i + 1]]) % mod;
38                     }
39                 }
40             }
41         for (int k = 0; k <= n; k++) for (int j = 1; j <= n; j++) g[j][k] = (g[j][k] + g[j - 1][k]) % mod;
42         memset(f, 0, sizeof f); swap(g, f);
43     }
44     cout << (ans % mod + mod) % mod << '\n';
45 }
```

实现要点

- 耐心 c 的人放弃 $\iff$ 排在第 $c + 1$ 个「拒绝或放弃」之后，这是整道题的入口。
- `next_hard[j]` 是位置 j 之后第一个难题位置；第 $i + 1$ 个放弃位置只能在它之前，难题位置本身被「拒绝」占用。
- `coeff` 的三项：选 x 个耐心 $= i$ 的人去后面（组合）、前面的人全排列（阶乘）、把前面的人插进空闲位置（组合）。
- 区间加用差分 `g`，每层结束求前缀和并 `swap(g, f)` 滚动。
- `sum[i]`、`fact`、`choose` 预处理； $H > n - m$ 直接输出 0。

部分分

测试点	$n \leq$	特殊性质	拿法
1—2	10	无	暴搜全排列
3—5	18	无	状压/暴搜
6—8	10^2	A	DP
9—11	10^2	无	DP
12—14	500	$m = 1$	特化
15	500	$m = n$	特化
16—17	500	A	s_i 全 1
18—21	500	B	状压 $\bigvee s_i < 18$

特殊性质：A s_i 全为 1；B $s_1, \dots, s_n$ 中最多 18 个为 1。

这道题的难点在「耐心 $\iff$ 第 $c + 1$ 个未录用」的等价转化，转化完才落得到一个逐层 DP。

2022 T4 数据传输

知识点：树上 DP、倍增、 $(\min,+)$ 矩阵

难度：省选/NOI-

题目

树上有 n 台主机，第 i 台处理信息耗时 v_i 。主机 a 能直接把信息传给 b 当且仅当两机在树上距离不超过 k ($k \leq 3$)。第 i 次请求把数据从主机 s_i 传到 t_i ，要选主机序列 $c_1 = s_i, c_2, \dots, c_m = t_i$ (相邻主机距离 $\leq k$)，总耗时 $\sum v_{c_j}$ 。 Q 次询问，求每次的最小总耗时。

限制

- $n, Q \leq 2 \times 10^5, 1 \leq k \leq 3, 1 \leq v_i \leq 10^9, s_i \neq t_i$

k = 1: 树上路径

$k = 1$ 时相邻主机必须直接相连，路径只能是 $s \rightarrow t$ 的树上简单路径。预处理前缀和 sum ，答案是 $sum_s + sum_t - 2 sum_{lca} + v_{lca}$ 。

k = 2, 3: 偏离层数 + 倍增

观察

最优路径的节点都在 $s \rightarrow t$ 树上路径的「 $k - 1$ 步邻域」内。设**状态** = 当前节点偏离路径的层数：0 在路径上、1 偏离一层（邻居）、2 偏离两层（邻居的邻居）。

给每个节点一个 $k \times k$ 的 $(\min, +)$ 转移矩阵，表示「经过该节点」时各状态之间的最小代价：

- $k = 2$: $a[0][0]=a[0][1]=v_i$, $a[1][0]=0$, $a[1][1]=\text{INF}$ 。
- $k = 3$: $a[0][0]=a[0][1]=a[0][2]=v_i$, $a[1][0]=0$ 、 $a[1][1]=w_i$ （邻居最小权）、 $a[1][2]=\text{INF}$, $a[2][1]=0$ 、其余 INF 。

矩阵乘法是 $c[i][j] = \min_l (a[i][l] + b[l][j])$ （同 Floyd），倍增把路径上的矩阵按顺序乘起来。

查询

把 $s \rightarrow t$ 拆成 $s \rightarrow lca$ (向上) 和 $lca \rightarrow t$ (向下) 两段:

- `prod_1`: 向上, 矩阵从下往上乘 ($p1[u][j] = p1[anc][j-1] * p1[u][j-1]$)。
- `prod_2`: 向下, 矩阵从上往下乘 ($p2[u][j] = p2[u][j-1] * p2[anc][j-1]$)。

答案 = $(prod_2(t) \cdot prod_1(s))[0][k-1]$, 即从「在路径上的 s 」到「在路径上的 t 」的最小代价。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const ll INF = 4e18;
5 const int maxn = 2e5 + 5;
6
7 int n, Q, k, v[maxn];
8 vector<int> g[maxn];
9 int anc[maxn][18], depth[maxn];
10 using mat = array<array<ll,3>,3>;
11
12 mat get_I() { mat a; for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) a[i][j] = (i == j ? 0 : INF); return a; }
13 mat mul(const mat& a, const mat& b) { mat c;
14     for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) { c[i][j] = INF;
15         for (int l = 0; l < k; l++) c[i][j] = min(c[i][j], a[i][l] + b[l][j]); }
16     return c; }
17 mat p1[maxn][18], p2[maxn][18];
18
19 mat prod_1(int u, int d) { mat a = get_I();
20     for (int i = 0; i < 18; i++) if (d >> i & 1) { a = mul(p1[u][i], a); u = anc[u][i]; } return a; }
21 mat prod_2(int u, int d) { mat a = get_I();
22     for (int i = 0; i < 18; i++) if (d >> i & 1) { a = mul(a, p2[u][i]); u = anc[u][i]; } return a; }
23
24 void dfs(int x, int p) { anc[x][0] = p; depth[x] = depth[p] + 1; for (int y : g[x]) if (y != p) dfs(y, x); }
25 int lca(int x, int y) {
26     if (depth[x] < depth[y]) swap(x, y);
27     for (int i = 0; i < 18; i++) if ((depth[x] - depth[y]) >> i & 1) x = anc[x][i];
28     if (x == y) return y;
29     for (int i = 17; i >= 0; i--) if (anc[x][i] != anc[y][i]) x = anc[x][i], y = anc[y][i];
30     return anc[x][0];
31 }
32
33 mat make_mat2(int i) { mat a; a[0][0] = a[0][1] = v[i]; a[1][0] = 0; a[1][1] = INF; return a; }
34 mat make_mat3(int i, vector<int>& w) { mat a;
35     a[0][0] = a[0][1] = a[0][2] = v[i];
36     a[1][0] = 0; a[1][1] = w[i]; a[1][2] = INF;
37     a[2][0] = INF; a[2][1] = 0; a[2][2] = INF; return a; }
38
39 void solve_general(int last) {
40     for (int j = 1; j < 18; j++) for (int i = 1; i <= n; i++) {
41         int x = anc[i][j-1];
42         p1[i][j] = mul(p1[x][j-1], p1[i][j-1]);
43         p2[i][j] = mul(p2[x][j-1], p2[i][j-1]);
44     }
45     while (Q--) { int s, t; cin >> s >> t; int x = lca(s, t);
46         mat a = mul(prod_2(t, depth[t]-depth[x]), prod_1(s, depth[s]-depth[x]+1));
47         cout << a[0][last] << "\n"; }
48 }
49
50 int main() {
51     ios::sync_with_stdio(false); cin.tie(0);
52     cin >> n >> Q >> k;
53     for (int i = 1; i <= n; i++) cin >> v[i];
54     for (int i = 0; i < n-1; i++) { int a, b; cin >> a >> b; g[a].push_back(b); g[b].push_back(a); }
55     dfs(1, 0);
56     for (int j = 1; j < 18; j++) for (int i = 1; i <= n; i++) anc[i][j] = anc[anc[i][j-1]][j-1];
57     if (k == 1) {
58         vector<ll> sum(n + 1);
59         function<void(int,int)> f = [&](int x, int p) { sum[x] = sum[p] + v[x]; for (int y : g[x]) if (y != p) f(y, x); };
60         f(1, 0);
61         while (Q--) { int s, t; cin >> s >> t; int x = lca(s, t); cout << sum[s]+sum[t]-2*sum[x]+v[x] << "\n"; }
62         return 0;
63     }
64     if (k == 2) { for (int i = 1; i <= n; i++) p1[i][0] = p2[i][0] = make_mat2(i); solve_general(1); }
65     else { vector<int> w(n + 1, INT_MAX);
66         for (int i = 1; i <= n; i++) for (int j : g[i]) w[i] = min(w[i], v[j]);
67         for (int i = 1; i <= n; i++) p1[i][0] = p2[i][0] = make_mat3(i, w); solve_general(2); }
68     return 0;
69 }
```

实现要点

- $(min, +)$ 矩阵乘法等价于「拼接两条路径取最小」，矩阵元素是状态之间的最短代价。
- 状态 $0/1/2$ 是「偏离路径的层数」； $k = 2$ 用 2×2 ， $k = 3$ 用 3×3 ， $k = 1$ 直接前缀和。
- 向上与向下的矩阵乘法**顺序相反**：p1 从下往上、p2 从上往下，倍增时不能写反。
- w_i 是「 i 的邻居中最小权值」，只有 $k = 3$ 需要（偏离两层时用）。
- 查询答案取 `a[0][k-1]`：从「在路径上的 s 」到「在路径上的 t 」；矩阵用 `array`，18 层倍增。

部分分

测试点	$n, Q \leq$	k	特殊性质	拿法
1	10	2	是	暴搜
2	10	3	是	暴搜
3	200	2	是	暴力建图 + 最短路
4—5	200	3	是	同上
6—7	2000	1	否	前缀和
8—9	2000	2	否	树上 DP
10—11	2000	3	否	树上 DP

特殊性质：保证 $a_i = i + 1$ ， b_i 从 $1, \dots, i$ 中等概率选取。

这道题把「路径上的最小花费」抽象成 $(\min, +)$ 矩阵链， $k \leq 3$ 让状态只到 3 层，倍增一乘就出来。

2021 T4 交通规划

知识点：平面图、对偶图最短路

难度：省选/NOI-

题目

平面上 n 条水平直线与 m 条垂直直线交成 $n \times m$ 网格，格点之间的边有非负边权。网格边缘有 $2n + 2m$ 条向外的射线，从左上角起顺时针编号 $1 \dots 2n + 2m$ 。

进行 T 次询问，第 i 次询问给 k_i 个附加点，每个附加点位于一条射线 p 上、颜色为 t (0 白、1 黑)，与最近格点之间有一条边权为 x 的边。把每个格点染成黑白，使**所有两端颜色不同的边**的边权和最小，输出这个最小值。

限制

- $2 \leq n, m \leq 500, 1 \leq T \leq 50, 1 \leq k_i \leq \min\{2(n + m), 50\}, 1 \leq \sum k_i \leq 50, 1 \leq p \leq 2(n + m)$
- $0 \leq x \leq 10^6, t \in \{0, 1\}$

最小割 = 对偶图最短路

转化

「异色边权和最小」是平面图的最小割；平面图的最小割等于**对偶图的最短路**。

对偶图：每个**小方格**是一个节点，边界外用 $2n + 2m$ 个节点排成环；每条网格边对应一条对偶边（权相同），把相邻的小方格（或小方格与边界环）连起来。

附加点落在环上：把环上对应的一条边权设为附加点与格点的边权 x ，即「从这条边界割开要花 x 」。

配对 + 环形 DP

观察

按射线编号顺时针排附加点，相邻异色 ($0 \rightarrow 1$ 或 $1 \rightarrow 0$) 的位置就是「割」的端点；把 $0 \rightarrow 1$ 记为 S 集、 $1 \rightarrow 0$ 记为 T 集，二者数量相等。

- 对每个 S 点跑一次 Dijkstra，得到到每个 T 点的最短路 g 。
- 环形区间 DP 求「 S 与 T 两两配对、路径互不相交」的最小总距离： $dp[i][j] = \min(dp[i+1][j-1] + g(i, j), \min_l dp[i][l] + dp[l+1][j])$ 。
- 答案 = $\min_i dp[i][i + cnt - 1]$ 。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const ll INF = 1e16;
5 const int MAXV = 260000, MAXE = 260000 * 8;
6
7 int n, m, T, idx;
8 int head[MAXV], tot = 1;
9 struct Edge { int v, nxt; ll w; } e[MAXE];
10 void add(int u, int v, ll w) { e[++tot] = {v, head[u], w}; head[u] = tot; e[++tot] = {u, head[v], w}; head[v] = tot; }
11 int find(int l, int j) { return (l - 1) * m + j; }
12
13 struct Add { ll x; int p, t; } a[60];
14 int s[60], t[60], c[60], id[MAXV], cnts, cntt, cnt;
15 ll dis[MAXV], g[60][60], dp[120][120];
16 bool vis[MAXV];
17
18 void dijkstra(int st) {
19     fill(dis, dis + MAXV, INF);
20     priority_queue<pair<ll, int>, vector<pair<ll, int>, greater<>> pq;
21     dis[st] = 0; pq.push({0, st}); memset(vis, 0, sizeof(vis));
22     while (!pq.empty()) { int u = pq.top().second; pq.pop();
23         if (vis[u]) continue; vis[u] = true;
24         for (int i = head[u]; i; i = e[i].nxt) { int v = e[i].v;
25             if (dis[v] > dis[u] + e[i].w) { dis[v] = dis[u] + e[i].w; pq.push({dis[v], v}); } } }
26 }
27 int cir(int x) { return (x - 1) % cnt + 1; }
28
29 void sol() {
30     int k; cin >> k;
31     for (int i = 1; i <= idx; i++) e[i << 1].w = e[i << 1 | 1].w = 0; // 重置环边权
32     for (int i = 1; i <= k; i++) { cin >> a[i].x >> a[i].p >> a[i].t; e[a[i].p << 1].w = e[a[i].p << 1 | 1].w = a[i].x; }
33     sort(a + 1, a + k + 1, [](Add& A, Add& B) { return A.p < B.p; });
34     cnts = cntt = cnt = 0;
35     if (a[1].t == 1 && a[k].t == 0) { s[++cnts] = a[1].p; c[++cnt] = a[1].p; }
36     if (a[1].t == 0 && a[k].t == 1) { t[++cntt] = a[1].p; c[++cnt] = a[1].p; }
37     for (int i = 2; i <= k; i++) {
38         if (a[i].t == 1 && a[i-1].t == 0) { s[++cnts] = a[i].p; c[++cnt] = a[i].p; }
39         if (a[i].t == 0 && a[i-1].t == 1) { t[++cntt] = a[i].p; c[++cnt] = a[i].p; }
40     }
41     if (cnts == 0) { cout << 0 << '\n'; return; }
42     sort(c + 1, c + cnt + 1);
43     for (int i = 1; i <= cnt; i++) id[c[i]] = i;
44     for (int i = 1; i <= cnt; i++) for (int j = 1; j <= cnt; j++) g[i][j] = INF;
45     for (int i = 1; i <= cnts; i++) { dijkstra(s[i]);
46         for (int j = 1; j <= cntt; j++) g[id[s[i]]][id[t[j]]] = g[id[t[j]]][id[s[i]]] = dis[t[j]]; }
47     for (int i = 1; i <= cnt * 2; i++) for (int j = 1; j <= cnt * 2; j++) dp[i][j] = INF;
48     for (int i = 1; i <= cnt * 2; i++) dp[i][i+1] = g[cir(i)][cir(i+1)];
49     for (int len = 4; len <= cnt; len += 2)
50         for (int i = 1; i <= cnt * 2 - len + 1; i++) { int j = i + len - 1;
51             dp[i][j] = dp[i+1][j-1] + g[cir(i)][cir(j)];
52             for (int l = i + 1; l < j; l += 2) dp[i][j] = min(dp[i][j], dp[i][l] + dp[l+1][j]); }
53     ll ans = INF;
54     for (int i = 1; i <= cnt; i++) ans = min(ans, dp[i][i + cnt - 1]);
55     cout << ans << '\n';
56 }
57
58 int main() {
59     ios::sync_with_stdio(false); cin.tie(0);
60     cin >> n >> m >> T; idx = (n + m) * 2;
61     for (int i = 1; i <= idx; i++) add(i, i + 1, 0);
62     add(idx, 1, 0);
63     for (int i = 1; i <= n; i++) for (int j = 1; j <= m; j++) { ll x; cin >> x;
64         if (j == 1) add(idx + find(i, j), idx - i + 1, x);
65         else if (j == m) add(idx + find(i, j - 1), m + i + 1, x);
66         else add(idx + find(i, j - 1), idx + find(i, j), x); }
67     for (int i = 1; i <= n; i++) for (int j = 1; j <= m; j++) { ll x; cin >> x;
68         if (i == 1) add(idx + find(i, j), j + 1, x);
69         else if (i == n) add(idx + find(i - 1, j), 2 * m + n - j + 1, x);
70         else add(idx + find(i - 1, j), idx + find(i, j), x); }
71     while (T--) sol();
72 }
```

实现要点

- 对偶图：小方格编号 $\text{find}(i, j) = (i-1) * m + j$ ，边界环节点 $1 \dots 2(n + m)$ ；环边初始权 0，附加点把对应环边权设为 x 。
- 附加点按射线编号排序，相邻异色处产生 S/T 端点； S 、 T 数量相等，全同色直接输出 0。
- 每个 S 点一次 Dijkstra， g 记两两最短路； $\text{cnt} \leq k \leq 50$ ，Dijkstra 次数 ≤ 25 。
- 环形 DP 在 2cnt 长度上做区间配对，答案取长度 cnt 的环区间最小值。
- 题目与「交通」无关，本质是平面图最小割；对拍用暴力枚举格点染色验证。

部分分

测试点	$n, m \leq$	$k_i \leq$	拿法
1—2	5	50	暴搜染色
3—5	18	2	最大流/最短路
6—8	18	50	最大流
9—10	100	2	最短路
11—12	100	50	最大流
13—16	500	2	对偶图最短路
17—20	500	50	满分做法

这道题难在「最小割 = 对偶图最短路」的建模和边界环的处理，建好图后就是最短路 + 环形配对 DP。

2020 T4 贪吃蛇

知识点：博弈、归纳

难度：省选/NOI-

题目

草原上有 n 条蛇，第 i 条体力 a_i （不降序）。实力比较：体力大的强，相等时编号大的强。每轮**最强**的蛇可选吃或不吃**最弱**的蛇：吃则最强蛇体力减去最弱蛇体力、最弱蛇被吃；不吃则决斗结束。每条蛇希望在自己不被吃的前提下尽量多吃。求最终存活蛇数。共 T 组数据，第一组给出全部体力，之后每组把 k 条蛇的体力 a_x 改为 y 。

限制

- $3 \leq n \leq 10^6, 1 \leq T \leq 10, 0 \leq k \leq 10^5, 0 \leq a_i, y \leq 10^9$

安全就吃，危险就犹豫

结论

最强的蛇吃了最弱的之后，若新体力**仍不是最弱**（安全），它一定吃；若新体力**变成最弱**（可能被下一条吃），则进入「犹豫」——它吃不吃要看后续。

犹豫链：从「吃了变最弱」开始，后续每轮最强蛇都面临同样的问题，直到某轮「吃了不变最弱」或「只剩 2 条」。这段的最终结果由**奇偶性**决定：链长偶数则最初那条吃、奇数则不吃。

双端队列 $O(n)$ 模拟

- 原数组不降序，读成递减的 `w1` (`pair<体力, 编号>`)。
- 每条被吃后新产生的体力 `强-弱` 一定小于原来最强，所以 `w2` 也递减；每轮从两个队列头取最强、队尾取最弱。
- `unsafe`：吃了之后（还剩 > 1 条时）新体力小于剩余最弱。
- `parity` 记录犹豫链奇偶；一旦回到「安全」，结算 `parity` 结束。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef pair<int,int> pii;
4 #define x first
5 #define y second
6
7 int n, a[1000005];
8 namespace run {
9   pii w1[1000005], w2[1000005];
10   pii *l1, *r1, *l2, *r2;
11   const pii inf(0x3f3f3f3f, 0x3f3f3f3f), ind(-0x3f3f3f3f, -0x3f3f3f3f);
12   pii mx1() { return l1 <= r1 ? *l1 : ind; }
13   pii mn1() { return l1 <= r1 ? *r1 : inf; }
14   pii mx2() { return l2 <= r2 ? *l2 : ind; }
15   pii mn2() { return l2 <= r2 ? *r2 : inf; }
16
17   int n_eaten, parity;
18   void mark_unsafe() { if (parity != -1) parity ^= 1; else parity = 0; }
19
20   void solve() {
21     l2 = w2 + 1; r2 = w2;
22     l1 = w1 + 1; r1 = w1 + n;
23     for (int i = 1; i <= n; ++i) { w1[n + 1 - i].x = a[i]; w1[n + 1 - i].y = i; }
24     n_eaten = 0; parity = -1;
25     int s = n;
26     auto unsafe = [&](pii x) { return s > 1 && x < min(mn1(), mn2()); };
27     while (s > 1) {
28       pii mx, mn;
29       if (mx1() > mx2()) { mx = mx1(); ++l1; } else { mx = mx2(); ++l2; }
30       if (mn1() < mn2()) { mn = mn1(); --r1; } else { mn = mn2(); --r2; }
31       mx.x -= mn.x; --s;
32       if (unsafe(mx)) { mark_unsafe(); }
33       else { if (parity != -1) { n_eaten += parity; return; } ++n_eaten; }
34       *(++r2) = mx;
35     }
36   }
37   void main() { solve(); printf("%d\n", n - n_eaten); }
38 }
39
40 int main() {
41   int t; scanf("%d", &t);
42   for (int id = 1; id <= t; ++id) {
43     if (id == 1) { scanf("%d", &n); for (int i = 1; i <= n; ++i) scanf("%d", &a + i); }
44     else { int k, x, y; scanf("%d", &k); for (; k; --k) { scanf("%d%d", &x, &y); a[x] = y; } }
45     run::main();
46   }
47   return 0;
48 }
```

实现要点

- `pair<体力, 编号>` 比较天然满足「体力大优先、相等编号大优先」，`w1` 用倒序填成递减。
- 新体力 `mx-mn` 单调不增，`w2` 始终递减，所以两队列头/尾即全局最值，全程 $O(n)$ 。
- `unsafe` 的判定用 `x < min(mn1(), mn2())`，注意只剩 1 条时不再判断。
- 组间修改只改 `a[x]`，数据保证 `a` 修改后仍不降序。

部分分

测试点	n	拿法
20%	$n = 3$	手推
40%	$n \leq 10$	递归模拟
55%	$n \leq 2000$	暴力模拟
70%	$n \leq 5 \times 10^4$	$O(n \log n)$ 堆模拟
100%	$n \leq 10^6$	双端队列 $O(n)$

这道题的难点全在「危险就犹豫、奇偶定胜负」这条结论，推导清楚后模拟反而简单。

2024 T4 擂台游戏

知识点：线段树、递归、阈值

难度：NOI/NOI+

题目

n 位选手按报名顺序编号，能力值 a_i 。补充最少选手使人数为 2^k 。擂台游戏分 k 轮，第 R 轮第 G 场抽签 $d_{R,G} \in \{0, 1\}$ ：0 表示编号小的选手为擂主、1 表示编号大的选手为擂主；**擂主获胜当且仅当其能力值 $\geq R$** （与另一位无关）。求所有可能成为冠军的选手的**编号之和**。

m 次询问 c_i ：只保留前 c_i 位选手时的答案。共 T 组测试数据，每组的能力值 $a_i = a'_i \oplus X_{i \bmod 4}$ 。

限制

- $2 \leq n, m \leq 10^5$, $0 \leq a_i, X_j < 2^{31}$, $1 \leq c_i \leq n$, $1 \leq T \leq 256$

线段树建模

把淘汰赛建成完全二叉树：叶节点是选手，内部节点 i 是第 r_i 轮的一场对局，区间 $[lp_i, rp_i]$ 。

- `winner[i]` 自底向上： `d[i]=0` 时左孩子是擂主， `a[左胜者] >= r[i]` 则左胜者晋级、否则右胜者晋级； `d[i]=1` 对称。
- 「可能成为冠军」 = 存在一种补充选手能力值的设定使其夺冠。

递归维护两类候选

状态

递归到当前区间时，把区间外「可能成为冠军」的选手分成两类：`champions` 不论当前区间胜者能力值如何都可能夺冠；`dependent_champions` 只有当当前区间胜者的能力值不够大（赢不出去）时才可能夺冠。

- `f[x]`：若当前区间胜者能力值为 x ，区间外 `dependent champions` 的编号之和。
- 递归按 `d[i]=0/1` 分两路，用 `at_least`（要赢出去的最低能力值）与 `lim` 做阈值判断。
- 对「前缀」 c_i 的答案落到叶子上：`ans[叶子] = champions + dependent_champions + 自己编号`。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 char d[1 << 18];
5 int winner[1 << 18]; // winner[i] : 对局 i 的胜者
6 int r[1 << 18];      // r[i] : 对局 i 所在轮数
7 int a[1 << 17];
8 int lp[1 << 18], rp[1 << 18]; // 区间左右端点
9 long long sum[1 << 18];
10 int N;
11 long long ans[1 << 17];
12 long long f[18]; // f[x] : 若当前区间胜者能力值为 x, dependent champions 编号之和
13
14 void dfs(int i, long long champions, long long dependent_champions, int at_least, int lim) {
15     if (i >= N) { ans[i - N] = champions + dependent_champions + (i - N + 1); return; }
16     if (d[i] == '0') { // 左孩子是擂主
17         if ((i & (i - 1)) == 0) dfs(i * 2, 0, 0, 0, 20);
18         else {
19             f[r[i] - 1] = dependent_champions + sum[2 * i + 1];
20             dfs(i * 2, champions, dependent_champions + sum[2 * i + 1], max(at_least, r[i]), max(r[i], lim));
21         }
22     } if (a[winner[i * 2]] >= r[i]) { // 左孩子是胜者
23         long long s;
24         if (a[winner[i * 2]] >= at_least) s = winner[i * 2];
25         else if (a[winner[i * 2]] < lim) s = f[a[winner[i * 2]]];
26         else s = 0;
27         fill(ans + lp[2 * i + 1] - 1, ans + rp[2 * i + 1], champions + s);
28     } else { // 左孩子不是胜者
29         f[r[i] - 1] = f[r[i]];
30         dfs(2 * i + 1, champions, dependent_champions, at_least, lim);
31     }
32 } else { // 右孩子是擂主
33     if ((i & (i - 1)) == 0) dfs(i * 2, 0, 0, 0, 20);
34     else dfs(i * 2, champions + sum[i * 2 + 1] + dependent_champions, 0, at_least, 0);
35     if (a[winner[i * 2]] >= at_least) f[r[i] - 1] = winner[i * 2];
36     else if (a[winner[i * 2]] < lim) f[r[i] - 1] = f[max(a[winner[i * 2]], r[i])];
37     else f[r[i] - 1] = 0;
38     int new_dependent_champion = 0;
39     if (a[winner[i * 2]] >= at_least) new_dependent_champion = winner[i * 2];
40     dfs(i * 2 + 1, champions, dependent_champions + new_dependent_champion, max(at_least, r[i]), max(r[i], lim));
41 }
42 }
43
44 int main() {
45     int n, m; cin >> n >> m;
46     vector<int> A(n + 1), c(m + 1), round(m + 1);
47     for (int i = 1; i <= n; i++) cin >> A[i];
48     for (int i = 1; i <= m; i++) { cin >> c[i]; while (1 << round[i] < c[i]) round[i]++; }
49     int k = 0; while ((1 << k) < n) k++;
50     N = 1 << k;
51     for (int i = k - 1; i >= 0; i--) for (int j = 1 << i; j < 1 << (i + 1); j++) cin >> d[j];
52     for (int i = 0; i < N; i++) { r[N + i] = 0; sum[N + i] = lp[N + i] = rp[N + i] = winner[N + i] = i + 1; }
53     for (int i = N + 1; i >= 1; i--) {
54         r[i] = r[i * 2] + 1; lp[i] = lp[i * 2]; rp[i] = rp[i * 2 + 1]; sum[i] = sum[i * 2] + sum[i * 2 + 1];
55     }
56     int T; cin >> T;
57     while (T--) {
58         vector<int> x(4); for (int i = 0; i < 4; i++) cin >> x[i];
59         for (int i = 1; i <= n; i++) a[i] = A[i] xor x[i & 3];
60         for (int i = N - 1; i >= 1; i--) {
61             int L = winner[2 * i], R = winner[2 * i + 1];
62             if (d[i] == '0') winner[i] = a[L] >= r[i] ? L : R;
63             else winner[i] = a[R] >= r[i] ? R : L;
64         }
65         dfs(1, 0, 0, 0, 20);
66         long long sum = 0;
67         for (int i = 1; i <= m; i++) {
68             long long v = (1 << round[i]) == c[i] ? winner[N >> round[i]] : ans[c[i]];
69             sum ^= i * v;
70         }
71         cout << sum << '\n';
72     }
73 }
```

实现要点

- `winner[i]` 依赖「擂主能力值 $\geq$ 轮数」这一条规则，与挑战者能力值无关，所以树可以一次建好。
- 递归只访问「胜者可能夺冠」的节点；`champions` 与 `dependent_champions` 分别是不依赖/依赖当前区间胜者能力值的候选。
- `at_least` 是「赢出去的最低能力值」、`lim` 是阈值上界；`f[x]` 用能力值当下标存 dependent 和，避免枚举。
- 叶子上直接得到前缀答案，非叶子用 `fill` 把同一段答案区间赋值。
- 输出是 $i \times A_i$ 的异或和，注意 `i * v` 可能爆 `int`。

部分分

测试点	T	$n, m \leq$	A	B	拿法
1—3	1	8	否	否	暴搜
4—5	1	500	是	否	分治
6—8	1	500	否	是	分治
9—10	1	5000	否	否	分治
11—12	1	10^5	是	否	特化
13—15	1	10^5	否	是	特化
16—17	4	10^5	否	否	满分
18—19	16	10^5	否	否	满分

特殊性质 A：所有 c_i 都是 2 的幂；B：所有 $d_{R,G} = 0$ 。

这道题是「能力值阈值」驱动的树上递推，分差全在能不能把「两类候选」的状态想清楚。

take away

- **先翻译模型**：T4 的分差在「把题意翻译成已知模型」，翻译对了代码量可控

六道 T4 各归一类

- **二分答案 + 树上贪心**：最晚种植时间 + 父先子后调度（种树）
- **计数 DP**：耐心 $\iff$ 第 $c + 1$ 个未录用，逐层 DP（员工招聘）
- **树上 (min,+) 矩阵 + 倍增**：偏离路径的层数当状态（数据传输）
- **平面图最小割 = 对偶图最短路**：边界环 + 环形配对 DP（交通规划）
- **博弈归纳**：安全就吃、危险就犹豫、奇偶定胜负（贪吃蛇）
- **阈值驱动的树上递推**：两类候选 + 能力值当下标（擂台游戏）
- **一套查错习惯**：二分边界、矩阵乘法方向、环形 DP 的环长、多组数据的清空