

主题

2020—2025 的第三题。

T3 的特点

- 比 T1、T2 再上一档：题目不再直接给出模型，要先做一步转化或抽象
- 六年的 T3 落在三类形态里：贪心与双指针、DP 与贡献计算、模拟与哈希
- 分差在「能不能想到那一步转化」，想出来代码往往不长

T3 (2020—2025)

题目	难度	通过率	知识点
2021 T3 回文	普及+/提高-	22.04%	贪心、双指针
2024 T3 染色	提高	28.57%	DP、前缀和
2023 T3 结构体	提高+/省选-	25.55%	模拟、内存对齐
2020 T3 函数调用	提高+/省选-	23.21%	拓扑排序、贡献计算
2025 T3 谐音替换	提高+/省选-	19.98%	哈希、字符串
2022 T3 星战	省选/NOI-	25.01%	哈希、度数统计

2021 T3 回文

知识点：贪心、双指针

难度：普及+/提高-

题目

给定 n 和长为 $2n$ 的序列 a ，其中 $1 \sim n$ 各恰好出现两次。每次把 a 的开头或末尾元素追加到 b 的末尾，共操作 $2n$ 次，使 b 成为回文。输出字典序最小的操作串（**L** 表示取开头，**R** 表示取末尾），无解输出 **-1**。

限制

- $1 \leq T \leq 100, 1 \leq n, \sum n \leq 5 \times 10^5$

第一个操作定格局

b 是回文，所以 $b_1 = b_{2n}$ ：第一个取出的元素必须等于最后一个取出的元素。

观察

若第一个取 a_1 (L)，则最后一个必须取「另一个 a_1 」。设它的位置为 p ，于是 a_p 必须是全程最后一个取出的元素。

于是 a_p 把剩下的序列分成两半：

- 左臂： $a_2, a_3, \dots, a_{p-1}$ ，只能从左端取 (L)；
- 右臂： $a_{2n}, a_{2n-1}, \dots, a_{p+1}$ ，只能从右端取 (R)。

中间 $b_2 \dots b_{2n-1}$ 必须由这两臂合并成一个回文。

两臂合并成回文

判定

设左臂区间 $[ll, lr]$ 、右臂区间 $[rl, rr]$ （右臂从 rl 向 rr 递减取）。最外层的一对是「某一臂的头」配「某一臂的尾」，只有四种可能，按字典序优先试 **L**：

- 左头配左尾： $a_{ll} = a_{lr}$ ；
- 左头配右尾： $a_{ll} = a_{rr}$ ；
- 右头配左尾： $a_{rl} = a_{lr}$ ；
- 右头配右尾： $a_{rl} = a_{rr}$ 。

每种配对里，头是「先取」（记入前半），尾是「后取」（记入后半）。任何一步四种都不成立就无解。

两个注意点：单臂只剩一个元素时不能自己配自己（需 $ll < lr$ 、 $rl > rr$ 的守卫）；因为每个值恰好出现两次，四选一最多只有一种成立，贪心不会漏解。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn = 5e5 + 5;
5 int a[2 * maxn]; char ans[2 * maxn]; int n;
6 int check(int l, int m, int r) {
7     int i = 1, l1 = l, r1 = m - 1, l2 = m + 1, r2 = r;
8     while (l1 <= r1 && l2 <= r2) {
9         if (a[l1] == a[r1] && l1 < r1) {
10             ans[i] = 'L'; ans[2 * n - 1 - i] = 'L'; l1++; r1--;
11         } else if (a[l1] == a[l2]) {
12             ans[i] = 'L'; ans[2 * n - 1 - i] = 'R'; l1++; l2++;
13         } else if (a[r2] == a[l2] && l2 < r2) {
14             ans[i] = 'R'; ans[2 * n - 1 - i] = 'R'; r2--; l2++;
15         } else if (a[r2] == a[r1]) {
16             ans[i] = 'R'; ans[2 * n - 1 - i] = 'L'; r2--; r1--;
17         } else return 0;
18         i++;
19     }
20     while (l1 < r1 && a[l1] == a[r1]) {
21         ans[i] = 'L'; ans[2 * n - 1 - i] = 'L';
22         i++; l1++; r1--;
23     }
24     while (l2 < r2 && a[l2] == a[r2]) {
25         ans[i] = 'R'; ans[2 * n - 1 - i] = 'R';
26         i++; l2++; r2--;
27     }
28     return l1 > r1 && l2 > r2;
29 }
30
31 int solve() {
32     cin >> n;
33     for (int i = 0; i < 2 * n; i++) cin >> a[i];
34
35     for (int i = 1; i < 2 * n; i++)
36         if (a[i] == a[0] && check(1, i, 2 * n - 1)) {
37             ans[0] = 'L', ans[2 * n - 1] = 'L';
38             return 1;
39         }
40     for (int i = 0; i < 2 * n - 1; i++)
41         if (a[i] == a[2 * n - 1] && check(0, i, 2 * n - 2)) {
42             ans[0] = 'R', ans[2 * n - 1] = 'L';
43             return 1;
44         }
45     return 0;
46 }
47
48 int main() {
49     int T;
50     cin >> T;
51     while (T--) {
52         if (solve()) {
53             ans[2 * n] = 0;
54             cout << ans << "\n";
55         } else cout << -1 << "\n";
56     }
```

实现要点

- `check(l, m, r)` : a_m 是 a_l 的配对者, m 把区间隔成左臂 $[l + 1, m - 1]$ 与右臂 $[m + 1, r]$, 四选一配对往中间收。
- `ans[i]` 与 `ans[2n-1-i]` 成对填 (一前一后), 避免中间反复移位。
- 先试 `a[0]` 的配对者 (L 起手, 字典序更小), 失败再试 `a[2n-1]` (R 起手), 都失败输出 `-1`。
- 单臂只剩一个元素时不能自配 (`l1 < r1` 、 `l2 < r2` 的守卫)。
- 每个元素至多被取一次, 全程 $O(n)$ 。

部分分

测试点	$n \leq$	$\sum n \leq$	特殊性质	拿法
1—7	10	50	无	爆搜所有 L/R 序列
8—10	20	1000	无	同上
11—12	100	1000	无	同上（剪枝）
13—15	1000	25000	无	同上或朴素贪心
16—17	5×10^5	5×10^5	无	两臂合并贪心
18—20	5×10^5	5×10^5	有	贪心
21—25	5×10^5	5×10^5	无	贪心

特殊性质：每次删除 a 中两个相邻且相等的数，存在一种方式把序列删空。

这道题的分差全在「把取数过程看成两臂合并」这一步，代码本身很短。

2024 T3 染色

知识点：DP、两条贪心性质

难度：提高

题目

给定数组 $A_1 \dots A_n$ ，把每个数染成红或蓝。若 A_i 与它左侧最近的同色数相等，则贡献 A_i ，否则贡献 0。求最大总分。

限制

- $1 \leq T \leq 10, 2 \leq n \leq 2 \times 10^5, 1 \leq A_i \leq 10^6$

相邻相等直接吃

观察

相邻两个相等的数 ($A_{i-1} = A_i$) 一定染同色：把它们拆开只会少拿 A_i 。这类贡献直接加进答案，不进 DP。

记 pre_i 为前一个值等于 A_i 的位置。若 i 与 pre_i 同色，则中间 $pre_i + 1 \dots i - 1$ 整体取另一种颜色；其中 $pre_i + 2 \dots i - 1$ 的相邻相等贡献已经算过，只剩 $pre_i + 1$ 的归属。

链式配对与单数组 DP

观察

合并后序列无相邻相等。位置 i (值 B_i) 要得分, 只能与**上一个同值位置** pre_i 同色、中间整段异色。跳到更早的同值点一定不优: 中间的 pre_i 会因异色而浪费。

$$f_i = \max(f_{i-1}, f_{pre_i+1} + B_i)$$

- f_{i-1} : i 不得分;
- $f_{pre_i+1} + B_i$: i 与 pre_i 配对, 中间整段异色, 其相邻相等贡献已计入 `sum`。

答案 = $f_m + sum$, 其中 `sum` 是观察 1 拿走的相邻相等贡献。

参考实现（单数组 DP）

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int MAXA = 1e6 + 5;
5
6 int main() {
7     ios::sync_with_stdio(false); cin.tie(0);
8     int T; cin >> T;
9     while (T--) {
10         int n; cin >> n;
11         vector<ll> a(n + 1), f(n + 1);
12         for (int i = 1; i <= n; i++) cin >> a[i];
13         vector<int> last(MAXA, 0);
14         ll sum = 0; int m = 0;
15         for (int i = 1; i <= n; i++) {
16             if (i > 1 && a[i] == a[i-1]) { sum += a[i]; continue; } // 观察 1
17             m++;
18             int pr = last[a[i]];
19             last[a[i]] = m;
20             f[m] = f[m-1];
21             if (pr >= 1) f[m] = max(f[m], f[pr+1] + a[i]); // 观察 2
22         }
23         cout << f[m] + sum << "\n";
24     }
25     return 0;
26 }
```

参考实现（滚动 DP）

这是「笨做法」：不显式记 `pre`，用 `g[x]` 存「上一个值为 x 的位置作为配对起点」的最优，靠 `add` 抵消相邻相等的固定收益来滚动。

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6  const int MAXN = 2e5 + 5;
7  int n;
8  int a[MAXN];
9  int T;
10 int main() {
11     ios :: sync_with_stdio(false);
12     cin.tie(0);
13     cin >> T;
14     while (T--) {
15         cin >> n;
16         for (int i = 1; i <= n; i++) {
17             cin >> a[i];
18         }
19         vector<long long> g(1e6 + 5, -1e15);
20         long long add = 0;
21         long long f = 0;
22         for (int i = 2; i <= n; i++) {
23
24             long long t = max(f, g[a[i]] + add + a[i]);
25             f = max(a[i] == a[i - 1] ? f + a[i] : f, t);
26             if (a[i] == a[i - 1])
27                 add += a[i];
28             g[a[i - 1]] = max(g[a[i - 1]], t - add);
29         }
30         cout << f << '\n';
31     }
32     return 0;
33 }
```

实现要点

- **两条性质共用**：相邻相等直接吃（合并）；要得分只能配上一个同值点（链式，不跳过）。
- **单数组 DP**：`last[x]` 记值 x 上次出现的位置（合并后下标），`f[m] = max(f[m-1], f[pre+1] + a[i])`，无偏移、语义直白。
- **滚动 DP**：`g[x]` 直接存「配对起点最优值」，`add` 抵消相邻相等，`t - add` 是倒扣回起点，要小心 `add` 更新时机。
- **多组数据**：单数组 DP 每组重建 `last` / `f`；滚动 DP 每组重建 `g`。

部分分

测试点	$n \leq$	$A_i \leq$	拿法
1—4	15	15	爆搜 2^n 种染色
5—7	100	100	爆搜或朴素 DP
8—10	2000	2000	$O(n^2)$ DP
11—12	2×10^4	10^6	相邻预处理 + DP
13—15	2×10^5	10	值域小, DP 常数小
16—20	2×10^5	10^6	满分做法

这道题先想清「相邻相等直接吃」和「只配上一个同值点」两条性质，再写 DP 就顺了；滚动 DP 是等价写法，单数组 DP 更直观。

2023 T3 结构体

知识点：模拟、内存对齐

难度：提高+/省选-

题目

基本类型四种：byte、short、int、long，大小分别为 1, 2, 4, 8 字节，对齐要求等于大小。结构体类型的对齐要求 = 成员对齐要求的最大值。

处理 n 次操作：

1. 定义结构体类型：给定类型名与成员（类型+名称），输出该类型的大小与对齐要求；
2. 定义元素：给定类型与名称，从地址 0 起按序分配（满足对齐），输出起始地址；
3. 访问元素：给定形如 a.b.c 的路径，输出最内层元素的起始地址；
4. 查询地址：给定 $addr$ ，若某基本类型元素占据该地址则输出其路径，否则输出 ERR。

限制

- $1 \leq n \leq 100$, $1 \leq k \leq 100$, $0 \leq addr \leq 10^{18}$

对齐与排布

规则

- 基本类型：对齐 = 大小。
- 结构体类型：对齐 = 成员对齐的最大值；成员按定义顺序排布，每个成员的偏移对齐到其类型对齐；结构体大小对齐到自身对齐。

定义元素时，所有元素从地址 0 起依次排开，每个元素对齐到其类型对齐；元素之间可能留出「洞」。

参考实现

```
1 //
2 // Created by xjs on 18/26/23.
3 //
4 #include <bits/stdc++.h>
5 using namespace std;
6 struct member {
7     struct Type *type;
8     string name;
9     long long offset;
10 };
11
12 long long ceil(long long x, int align) {
13     return (x + align - 1) / align * align;
14 }
15
16 struct Type {
17     vector<member> mem;
18     long long size = 0;
19     int align = 0;
20     void add_member(Type *type, string name) {
21         size = ceil(size, type->align);
22         cout << type << " " << name << " " << size << '\n';
23         mem.push_back({type, name, size});
24         align = max(align, type->align);
25         size += type->size;
26     }
27 };
28
29 long long access(Type *t, long long offset, string path) {
30     // cout << "access " << offset << " " << path << '\n';
31     if (path.empty())
32         return offset;
33     int len = 0;
34     while (len < path.size() && path[len] != '.') len++;
35     string name = path.substr(0, len);
36     if (len == path.size()) len++;
37     for (auto &i: t->mem) {
38         if (i.name == name) {
39             return access(i.type, offset + i.offset, path.substr(len));
40         }
41     }
42     return -1;
43 }
44
45 string locate(Type *t, string path, long long offset) {
46     // cout << "path: " << path << " " << offset << '\n';
47     if (t->mem.empty()) {
48         if (offset < t->size)
49             return path;
50         return "ERR";
51     }
52     for (int i = (int) t->mem.size() - 1; i >= 0; i--)
53         if (t->mem[i].offset <= offset) {
54             cout << t->mem[i].name << '\n';
55             string new_path = path.empty() ? path + t->mem[i].name : path + "." + t->mem[i].name;
56             return locate(t->mem[i].type, new_path, offset - t->mem[i].offset);
57         }
58 }
59
60 int main() {
61     // freopen("struct2.in", "r", stdin);
62
63     map<string, Type> typeInfo;
64     typeInfo["byte"] = {{}, 1, 1};
65     typeInfo["short"] = {{}, 2, 2};
66     typeInfo["int"] = {{}, 4, 4};
67     typeInfo["long"] = {{}, 8, 8};
68
69     int n;
70     cin >> n;
71     Type global;
72     while (n--) {
73         int op;
74         cin >> op;
75         if (op == 1) {
76             string name;
77             int k;
78             cin >> name >> k;
79             Type cur;
80             while (k--) {
81                 string N, T;
82                 cin >> T >> N;
83                 cur.add_member(&typeInfo[T], N);
84             }
85             cur.size = ceil(cur.size, cur.align);
86             typeInfo[name] = cur;
87             cout << cur.size << " " << cur.align << '\n';
88         } else if (op == 2) {
89             string type, name;
90             cin >> type >> name;
91             global.add_member(&typeInfo[type], name);
92             cout << global.mem.back().offset << '\n';
93         } else if (op == 3) {
94             string s;
95             cin >> s;
96             cout << access(&global, 0, s) << '\n';
97         } else {
98             long long addr;
99             cin >> addr;
100             cout << locate(&global, "", addr) << '\n';
101         }
102     }
103 }
```

实现要点

- `Type::add_member` 做「对齐到成员对齐 → 记偏移 → 累加大小」，对齐取成员最大值；类型定义完把大小再对齐到自身对齐。
- `map<string, Type>` 存类型，`byte/short/int/long` 先手写进去；`global` 也当成一个 `Type`，元素就是它的成员，天然按序排布。
- `access` 沿 `.` 逐段找成员名累加偏移；`locate` 从大到小找「偏移 $\leq$ `addr`」的成员向下钻，钻到基本类型判断是否越界。
- 题目保证操作合法， $n \leq 100$ ，`map` + 线性遍历即可。

部分分

测试点	特殊性质	拿法
1	A、D	只有元素， <code>long</code> 一种基本类型
2—3	A	没有操作 1，只处理元素与查询
4—5	B、D	只有一个结构体类型
6—8	B	只有一个结构体类型（含嵌套）
9—10	C、D	成员全基本类型
11—13	C	成员全基本类型

特殊性质：A 没有操作 1；B 只有一个操作 1；C 操作 1 的成员类型全为基本类型；D 基本类型只有 `long`。

这道题是纯模拟，分差在「对齐、偏移、地址查询」三处细节，想清楚再写就能一遍过。

2020 T3 函数调用

知识点：拓扑排序、贡献计算

难度：提高+/省选-

题目

有 n 个数据 a_i ，和 m 个函数，每个函数为三种之一：

1. 给下标 P_j 加 V_j ；
2. 把所有元素乘 V_j ；
3. 依次调用 C_j 个函数（保证无递归，即调用关系是 DAG）。

给定一个执行序列 Q （依次执行 Q 个函数），求每个数据最终值，对 998244353 取模。

限制

- $n, m, Q \leq 10^5$, $\sum C_j \leq 10^6$, $a_i, V_j \leq 10^4$

乘数与加值的系数

两个量

- `mul[i]`：函数 i 执行一次，把整体乘了多少（type 1 为 1，type 2 为 V ，type 3 为子函数 `mul` 之积）。
- `c[i]`：函数 i 的「加」操作后面还跟着多大的乘数——即它每个加值要乘的系数。

`mul` 在逆拓扑序（被调用者先）里算。`c` 从执行序列倒着来：维护后缀乘数 `cheng`，遇到 f_k 若不是乘操作就把 `cheng` 累加进 `c[f_k]`，再 `cheng *= mul[f_k]`；最后 `cheng` 就是总乘数。

顺拓扑序传播

c 顺着拓扑序（调用者先）传给被调用者，同样从后往前、维护后缀乘数。

答案

- 初始 a_i 先整体乘总乘数 `cheng` ；
- 每个 type 1 的加值贡献 = $V \times c[i]$ ，累加到对应下标；
- type 2 的贡献已经折进 `mul` 和 `cheng`，不必单独数执行次数。

参考实现

```
1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4
5 using namespace std;
6
7 const int maxn = 1e5 + 5;
8 vector<int> g[maxn];
9 vector<int> fn; // 按拓扑序列的m个数。
10 int vis[maxn];
11
12 void dfs(int u) {
13     vis[u] = 1;
14     for (int v : g[u])
15         if (!vis[v])
16             dfs(v);
17     fn.push_back(u);
18 }
19
20
21 const int mod = 998244353;
22
23 int main() {
24     int n;
25     cin >> n;
26     vector<int> a(n + 1);
27     for (int i = 1; i <= n; i++)
28         cin >> a[i];
29     int m;
30     cin >> m;
31     vector<long long> mul(m + 1, 1);
32     vector<pair<int, long long>> add(m + 1);
33     vector<int> type(m + 1);
34
35     for (int i = 1; i <= m; i++) {
36         cin >> type[i];
37         if (type[i] == 1) {
38             cin >> add[i].first >> add[i].second;
39         } else if (type[i] == 2) {
40             cin >> mul[i];
41         } else {
42             int c;
43             cin >> c;
44             while (c--) {
45                 int x;
46                 cin >> x;
47                 g[i].push_back(x);
48             }
49         }
50     }
51
52     // 拓扑排序
53     for (int i = 1; i <= m; i++)
54         if (!vis[i])
55             dfs(i);
56
57     for (int i : fn)
58         for (int j : g[i])
59             mul[i] = mul[i] * mul[j] % mod;
60
61     int Q;
62     cin >> Q;
63     vector<int> f(Q);
64     for (int i = 0; i < Q; i++)
65         cin >> f[i];
66
67     vector<long long> c(m + 1); // c[i] : 函数1里的加操作结果的倍数。
68     long long cheng = 1;
69     for (int j = Q - 1; j >= 0; j--) {
70         int i = f[j];
71         if (type[i] != 2)
72             c[i] = (c[i] + cheng) % mod;
73         cheng = cheng * mul[i] % mod;
74     }
75
76     for (int i = 1; i <= n; i++)
77         a[i] = a[i] * cheng % mod;
78
79     // 顺着拓扑序从前往后“传播”。
80     for (int i = m - 1; i >= 0; i--) {
81         int j = fn[i];
82         cheng = 1;
83         for (int k = (int) g[j].size() - 1; k >= 0; k--) {
84             int l = g[j][k];
85             if (type[l] != 2) {
86                 c[l] = (c[l] + c[j] * cheng % mod) % mod;
87             }
88             cheng = cheng * mul[l] % mod;
89         }
90         if (type[j] == 1) {
91             a[add[j].first] = (a[add[j].first] + add[j].second * c[j] % mod) % mod;
92         }
93     }
94     for (int i = 1; i <= n; i++)
95         cout << a[i] << ' ';
96     cout << "\n";
97     return 0;
98 }
```

实现要点

- 拓扑序用 DFS 后序 (fn 数组); mul 在拓扑序上乘子函数的 mul。
- c[i] : 主序列倒着维护 cheng, 非乘函数 $c[i] += cheng$, 再 $cheng *= mul[i]$; cheng 收尾即总乘数。
- 顺拓扑序把 c 传给子函数 (从后往前、维护后缀乘数); type 1 直接 $a[P] += V * c[i]$ 。
- 初始 a 先整体乘 cheng; 全程取模, cheng 用 long long 防中间溢出。

部分分

测试点	$n, m, Q \leq$	特殊限制	拿法
1—2	1000	调用关系是树	直接递归模拟
3—4	1000	$\sum C_j \leq 100$	递归模拟
5—6	2×10^4	只有加或只有乘	单向贡献
7	2×10^4	$\sum C_j = 0$	无 type 3, 直接算
8—9	2×10^4	调用关系是树	树上递归 + 后缀乘数
10—11	2×10^4	$\sum C_j \leq 2 \times 10^5$	拓扑排序
12—13	10^5	只有加或只有乘	单向贡献
14	10^5	$\sum C_j = 0$	直接算
15—16	10^5	调用关系是树	树上后缀乘数

这道题的难点全在「加值后面跟了多少乘数」这个后缀贡献，想清楚 `mul` 与 `coef` 两个量就通了。

2025 T3 谐音替换

知识点：哈希、字符串、离线统计

难度：提高+/省选-

题目

给定 n 个字符串二元组 $(s_{i,1}, s_{i,2})$ ，满足 $|s_{i,1}| = |s_{i,2}|$ 。对字符串 s 的一次「替换」：选一个子串 y （位置 x ）和二元组 i 使 $y = s_{i,1}$ ，把 y 换成 $s_{i,2}$ 。

q 次询问，每次给定两个不同字符串 $t_{j,1}, t_{j,2}$ ，求有多少种替换能把 $t_{j,1}$ 变成 $t_{j,2}$ （位置或二元组不同算不同）。

限制

- $n, q \leq 2 \times 10^5$, $L_1, L_2 \leq 5 \times 10^6$ （字符串总长）

差异段唯一

替换不改变长度，且只改一个连续区间。所以 t_1 与 t_2 只能在一段连续区间上不同。

观察

记 t_1, t_2 的最长公共前缀为 P 、最长公共后缀为 Q ，中间的「差异段」为 M_1 (t_1 里) 与 M_2 (t_2 里)。一个二元组 (s_1, s_2) 能完成替换，当且仅当：

- s_1, s_2 去掉各自公共前后缀后的「差异段」分别等于 M_1, M_2 ；
- s_1, s_2 的公共前缀是 P 的**后缀**；
- s_1, s_2 的公共后缀是 Q 的**前缀**。

第一处不同 P 之前、 Q 之后都相等，所以替换区间的公共前缀只能对齐到 P 的末尾、公共后缀对齐到 Q 的开头。

签名 + Trie + 后缀链接

- 一次替换只改一个连续区间， t_1, t_2 只有一段不同。设首个不同位置 i 、末个不同位置 j ，把二元组编码成签名：公共前缀 + { + (s_1 差异段 + s_2 差异段) + { + 公共后缀 ({ 取 z 后一位当分隔符)。
- 把所有二元组的签名插进一棵 27 字符的 Trie，节点权值 = 以该节点结尾的二元组数。
- 建 AC 自动机的 fail 链（后缀链接），把权值沿 fail 链向上累加。
- 询问的签名同样在 Trie 上走，沿途累加权值即为答案。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #ifdef LOCAL
5 #include "debug.h"
6 #else
7 #define debug(...) 42
8 #endif
9
10 template<int sigma_size, char alpha>
11 struct Trie {
12     vector<array<int, sigma_size>> go;
13     int new_node() {
14         go.push_back({});
15         return (int) go.size() - 1;
16     }
17     Trie() {
18         new_node();
19     }
20     int add(string s) {
21         int p = 0;
22         for (char c : s) {
23             int i = c - alpha;
24             if (go[p][i] == 0) {
25                 go[p][i] = new_node();
26             }
27             p = go[p][i];
28         }
29         return p;
30     }
31     vector<pair<int, int>> get_suffix_link() {
32         vector<pair<int, int>> q;
33         for (int i = 0; i < sigma_size; i++)
34             if (go[0][i])
35                 q.push_back((go[0][i], 0));
36
37         for (int j = 0; j < (int) q.size(); j++) {
38             int u = q[j].first, v = q[j].second;
39             for (int i = 0; i < sigma_size; i++)
40                 if (go[u][i])
41                     q.push_back((go[u][i], go[v][i]));
42             else
43                 go[u][i] = go[v][i];
44         }
45         return q;
46     }
47 };
48
49 string work(string a, string b) {
50     int n = (int) a.size();
51     for (int i = 0; i < n; i++)
52         if (a[i] != b[i])
53             for (int j = n - 1; j >= 0; j--)
54                 if (a[j] != b[j]) {
55                     string t = a.substr(i, j - i + 1) + b.substr(i, j - i + 1);
56                     return a.substr(0, i) + '{' + k + '{' + b.substr(j + 1); // '{' 是 'z' 的下一个字符
57                 }
58     return "";
59 }
60
61 int main() {
62     ios::sync_with_stdio(0);
63     cin.tie(0);
64     int n, q;
65     cin >> n >> q;
66
67     Trie<27, 'a'> trie;
68     vector<int> id;
69     for (int i = 0; i < n; i++) {
70         string s1, s2;
71         cin >> s1 >> s2;
72         string s = work(s1, s2);
73         if (!s.empty())
74             id.push_back(trie.add(s));
75     }
76     vector<int> weight(trie.go.size()); // 节点的权值
77     for (int i : id)
78         weight[i]++;
79     // 正向传播
80     for (auto [u, v] : trie.get_suffix_link())
81         weight[u] += weight[v];
82     // 回溯询问
83     for (int i = 0; i < q; i++) {
84         string t1, t2;
85         cin >> t1 >> t2;
86         int ans = 0;
87         if (t1.size() == t2.size()) {
88             string t = work(t1, t2);
89             int p = 0;
90             for (char c : t) {
91                 p = trie.go[p][c - 'a'];
92                 ans += weight[p];
93             }
94             cout << ans << '\n';
95         }
96     }
97 }
```

实现要点

- `work(a,b)` 返回签名：公共前缀 + `{` + a、b 差异段拼接 + `{` + 公共后缀；两串相同返回空串（题面保证询问不会发生）。
- Trie 用 `array<int,27>` 存子节点，`get_suffix_link` 是标准 AC 自动机 fail 链 BFS。
- 权值先落在「以该节点结尾」上，再沿 fail 链向上累加，使节点权值 = 所有「签名是当前串后缀」的二元组数。
- 询问签名在 Trie 上走，每走一个字符累加当前节点权值；`t1.size()!=t2.size()` 直接 0。

部分分

测试点	$n, q \leq$	$L \leq$	拿法
1—2	10^2	200	枚举位置 $\times$ 二元组
3—5	10^3	2000	同上（哈希）
6—8	10^4	10^6	哈希 + 按差异段分组
9—10	2×10^5	10^6	分组 + Trie
11—14	2×10^5	5×10^6	满分做法

这道题要同时处理「公共前后缀」两个维度，分差全在能不能把它们统一成「差异段 + 两棵 Trie」这一步。

2022 T3 星战

知识点：哈希、度数统计

难度：省选/NOI-

题目

n 个据点、 m 条单向虫洞。据点 u 能「连续穿梭」当且仅当从它出发的可用虫洞恰好一条；能「反击」当且仅当能无限次穿梭。维护 q 次操作：

1. 摧毁虫洞 $u \rightarrow v$ ；2. 摧毁据点 u （毁掉它所有入边）；3. 修复虫洞 $u \rightarrow v$ ；4. 修复据点 u （修好它所有入边）。

每次操作后输出：是否所有据点都能反击且能连续穿梭（即「反攻时刻」）。

限制

- $n, m, q \leq 5 \times 10^5$

出度全 1 $\iff$ 反攻

观察

每个据点出度都为 1 时，图是一堆环，每个点都能无限穿梭。所以「反攻时刻」等价于**每个据点出度恰好为 1**。

直接维护每个点的出度，批量操作（毁/修一个点的所有入边）会波及很多点的出度，难以 $O(1)$ 更新。改用随机哈希判「出度全 1」。

随机哈希判出度

给每个据点 u 一个随机权值 w_u ，令 $S = \sum w_u$ 。维护

$$tot = \sum_{\text{存活的边 } u \rightarrow v} w_u$$

判定

$tot = S$ 当且仅当每个据点都恰好作为**起点**出现一次，即每个据点出度都为 1（随机哈希，错误概率可忽略）。

批量更新只需维护「每个据点当前存活入边的起点权值和」 $s[u]$ 与它的全量 $sum[u]$ ：毁点 $tot -= s[u], s[u]=0$ ；修点 $tot += sum[u]-s[u], s[u]=sum[u]$ ；单边同理。

参考实现

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     ios::sync_with_stdio(0);
6     int n, m; cin >> n >> m;
7     random_device r;
8     default_random_engine e(r());
9     uniform_int_distribution<int> dist(1, INT_MAX);
10    long long S = 0; vector<int> w(n + 1);
11    for (int i = 1; i <= n; i++) {
12        w[i] = dist(e); S += w[i];
13    }
14    vector<long long> sum(n + 1), s(n + 1);
15    for (int i = 0; i < m; i++) {
16        int u, v; cin >> u >> v; sum[v] += w[u];
17    }
18    long long tot = 0;
19    for (int i = 1; i <= n; i++) {
20        s[i] = sum[i]; tot += s[i];
21    }
22    int q; cin >> q;
23    for (int i = 0; i < q; i++) {
24        int t, u, v; cin >> t >> u;
25        if (t == 1) {
26            cin >> v; s[v] -= w[u]; tot -= w[u];
27        } else if (t == 3) {
28            cin >> v; s[v] += w[u]; tot += w[u];
29        } else if (t == 2) {
30            tot -= s[u]; s[u] = 0;
31        } else {
32            tot += sum[u] - s[u]; s[u] = sum[u];
33        }
34        cout << (tot == S ? "YES" : "NO") << '\n';
35    }
36 }
```

实现要点

- $w[u]$ 随机权值, $S = \sum w$; $\text{sum}[u]$ 是 u 所有入边起点的权值和, $s[u]$ 是当前存活入边的权值和。
- 毁点 2: $\text{tot} -= s[u]$, $s[u] = 0$; 修点 4: $\text{tot} += \text{sum}[u] - s[u]$, $s[u] = \text{sum}[u]$ 。
- 单边毁/修 (1、3) 只改一条边的终点 s 与全局 tot 。
- $\text{tot} == S$ 判「出度全 1」; $\text{random_device} + \text{default_random_engine}$ 随机权值, 碰撞概率可忽略 (Monte Carlo)。
- 全程 $O(1)$ 更新、 $O(1)$ 判定, 总复杂度 $O(n + m + q)$ 。

部分分

测试点	$n, m, q \leq$	特殊限制	拿法
1—3	10/20/50	无	每次暴力算出度
4—8	10^3	无	暴力 + 邻接表
9—10	5×10^5	没有 t=2、t=4	只维护单边，直接算出度
11—12	5×10^5	没有 t=4	拆点/懒标记
13—16	10^5	无	随机哈希判出度
17—20	5×10^5	无	满分做法

这道题的难点在「批量毁/修一个点的入边」会波及很多出度，随机哈希把一个全局判定压成 $O(1)$ 。

take away

- **先找模型**：T3 的分差在「能不能想到那一步转化」，想出来代码通常不长

五类套路，六道题各归一类

- **贪心与双指针**：把取数过程看成两臂合并成回文（回文）
- **DP 与贡献**：相邻相等直接吃、配对转移（染色）；乘数与后缀贡献分离（函数调用）
- **模拟**：内存对齐逐条模拟（结构体）
- **哈希与字符串**：签名 + Trie 后缀链接（谐音替换）；随机哈希把出度判定压成 $O(1)$ （星战）
- **一套查错习惯**：哈希冲突与取模、对齐与偏移、多组数据的清空