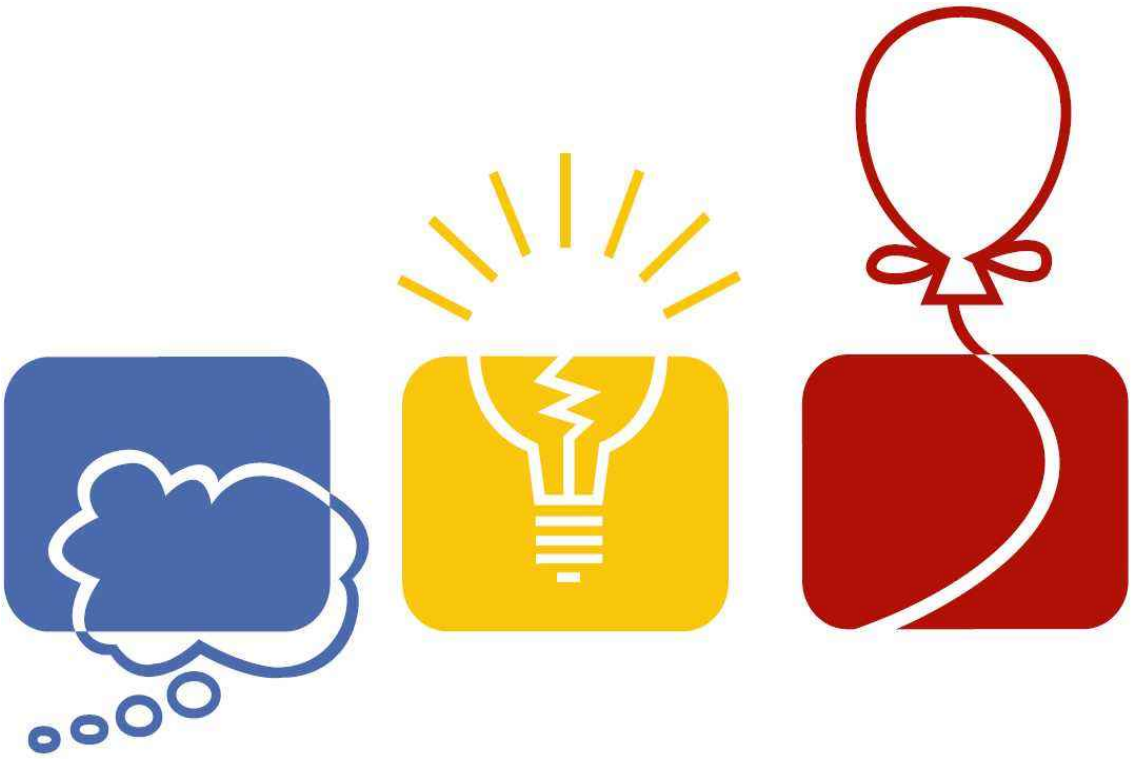
Orzjh 的 XCPC 算法模板

Orzjh

2023 年 1 月 30 日



目录

1 动态规划

1.1 LIS & LCS & LCIS

```
/*
LIS O(nlogn)实现
第一问求最长单调不上升子序列
第二问求最长单调上升子序列

input: 389 207 155 300 299 170 158 65
output: 6 2

f[i]代表当最长单调子序列长度为i时最优的末尾元素
len为最长单调子序列的长度
以最长不上升子序列为例
①当a[i] <= f[len] 说明可以接在后面
②当a[i] > f[len] 在f中找到第一个小于a[i]的数 并替换为a[i]
*/

int LIS() {
    while(cin >> a[++n]) ; n--;
    for(int i = 1; i <= n; i++) cout << a[i] << " "; cout << endl;
    printf("%d\n%d", LIS_Solve1(), LIS_Solve2());
    //problem1
    int len = 1; f[1] = a[1];
    for(int i = 2; i <= n; i++) {
        if(a[i] <= f[len]) f[++len] = a[i];
        else f[upper_bound(f + 1, f + 1 + len, a[i], greater<int>() ) - f] = a[i];
    }
    return len;
    //problem2
    int len = 1; f[1] = a[1];
    for(int i = 2; i <= n; i++) {
        if(a[i] > f[len]) f[++len] = a[i];
        else f[lower_bound(f + 1, f + 1 + len, a[i]) - f] = a[i];
    }
    return len;
}

int LCS() {
    scanf("%s%s", S + 1, T + 1);
    int sLen = strlen(S + 1), tLen = strlen(T + 1);
    for(int i = 1; i <= sLen; i++) {
        for(int j = 1; j <= tLen; j++) {
            if(S[i] == T[j]) dp[i][j] = dp[i - 1][j - 1] + 1;
            else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    // 记录路径
    int p = 0, px = sLen, py = tLen;
    while(px != 0 && py != 0) {
        if(dp[px][py] == dp[px][py - 1]) py--;
        else if(dp[px][py] == dp[px - 1][py]) px--;
        else if(dp[px][py] == dp[px - 1][py - 1] + 1) {
            ans[++p] = S[px];
            px--, py--;
        }
    }
    for(int i = p; i >= 1; i--) printf("%c", ans[i]);
}

void LCIS() {  // O(nm)
    a[0] = b[0] = -1;
    scanf("%d", &n); for(int i = 1; i <= n; i++) scanf("%d", &a[i]);
    scanf("%d", &m); for(int i = 1; i <= m; i++) scanf("%d", &b[i]);
```

```

    for(int i = 1; i <= n; i++) {
        int curmax = dp[i - 1][0] + 1, curpos = 0;
        for(int j = 1; j <= m; j++) {
            if(a[i] == b[j]) {
                if(curmax > dp[i][j]) dp[i][j] = curmax, pre[i][j] = curpos;
            } else dp[i][j] = dp[i - 1][j], pre[i][j] = pre[i - 1][j];

            if(b[j] < a[i] && curmax < dp[i - 1][j] + 1) curmax = dp[i - 1][j] + 1, curpos = j;
        }
    }

    int ans = 0, pos = 0;
    for(int i = 1; i <= m; i++) if(ans < dp[n][i]) ans = dp[n][i], pos = i;
    printf("%d\n", ans);

    while(pos) vec.push_back(b[pos]), pos = pre[n][pos];
    reverse(vec.begin(), vec.end());
    for(auto i : vec) printf("%d ", i);
}
```

1.2 背包问题

```
/*
N种物品 容量为V的背包 第i种物品的代价为w[i] 价值为v[i] 数量为m[i]
*/

void Prework() {
    dp[0] = 0;
    for(int i = 1; i <= V; i++) {
        //dp[i] = INF; //要求必须装满背包
        dp[i] = 0; //不一定要装满背包
    }
}

void ZeroOnePack() { //01背包
    Prework();
    for(int i = 1; i <= N; i++) {
        for(int j = V; j >= w[i]; j--) {
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
        }
    }
}

void CompletePack() { //完全背包
    Prework();
    for(int i = 1; i <= N; i++) {
        for(int j = w[i]; j <= V; j++) {
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
        }
    }
}

void MultiplePack() { //多重背包 利用二进制优化
    Prework();

    int cnt = 0;
    for(int i = 1; i <= N; i++) {
        int cur = 1; //将cur个相同物品合并
        int cur_w, cur_v;
        while(m[i] >= cur) {
            cur_w = w[i] * cur;
            cur_v = v[i] * cur;
            for(int j = V; j >= cur_w; j--) {
```

```
            dp[j] = max(dp[j], dp[j - cur_w] + cur_v);
        }
        m[i] -= cur;
        cur <<= 1;
    }

    cur_w = w[i] * m[i];
    cur_v = v[i] * m[i];
    for(int j = V; j >= cur_w; j--) {
        dp[j] = max(dp[j], dp[j - cur_w] + cur_v);
    }
  }

}

void MixedPack() { //混合背包
  for(int i = 1; i <= n; i++) {
    if(m[i] == 0) {
        //代表无限个 采用完全背包策略
        for(int j = w[i]; j <= V; j++) {
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
        }
    } else if(m[i] == 1) {
        //01背包策略
        for(int j = V; j >= w[i]; j--) {
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
        }
    } else if(m[i] > 1) {
        //多重背包策略
        int cur = 1; //将cur个相同物品合并
        int cur_w, cur_v;
        while(m[i] >= cur) {
            cur_w = w[i] * cur;
            cur_v = v[i] * cur;
            for(int j = V; j >= cur_w; j--) {
                dp[j] = max(dp[j], dp[j - cur_w] + cur_v);
            }
            m[i] -= cur;
            cur <<= 1;
        }

        cur_w = w[i] * m[i];
        cur_v = v[i] * m[i];
        for(int j = V; j >= cur_w; j--) {
            dp[j] = max(dp[j], dp[j - cur_w] + cur_v);
        }
    }
  }
}

void TwoDimensionPack() {
  /*
  二维费用背包问题
  设第二维容量为T 第i个物品的代价为g[i]

  01背包下变量j和k采用逆序循环
  完全背包下采用顺序循环
  多重背包时拆分物品
  */

  for(int i = 1; i <= n; i++) { //二维01背包
    for(int j = V; j >= w[i]; j--) {
        for(int k = T; k >= g[i]; k--) {
            dp[j][k] = max(dp[j][k], dp[j - w[i]][k - g[i]]);
        }
```

```
        }
    }

}

void DivideGroupPack() { //分组背包
    /*
    这些物品被划分为若干组，每组中的物品互相冲突，最多选一件。
    求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

    设第i件物品的类型为type[i]，共有k个类型。
    每组物品有若干种策略：是选择本组的某一件，还是一件都不选。
    设dp[i][j]表示前i组物品花费代价j所获得的最大价值
    dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[cur]] + v[cur]);
    cur代表属于组别i的当前物品
    可用滚动数组优化 优化时注意三层循环的顺序不能更改
    */

    for(int i = 1; i <= N; i++) Items[ type[i] ].push_back(i); //进行分类

    for(int i = 1; i <= K; i++) {
        int num = Items[i].size();
        for(int j = V; j >= 0; j--) {
            for(int k = 0; k < num; k++) {
                //对k的循环要在对j的循环里面 这样能保证每个组别最多选一个物品
                int cur = Items[i][k];
                if(w[cur] > j) continue;
                dp[j] = max(dp[j], dp[j - w[cur]] + v[cur]);
            }
        }
    }

}

void PackOnTree(int u) { //树上背包 利用上下界优化
    Size[u] = 1;
    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        PackOnTree(v);
        for(int j = min(m + 1, Size[u] + Size[v]); j >= 0; j--) {
            for(int k = max(1, j - Size[u]); k < j && k <= Size[v]; k++) {
                f[u][j] = max(f[u][j], f[u][j - k] + f[v][k]);
            }
        }
        Size[u] += Size[v];
    }
}

void RollbackPack() { //回退背包
    // select
    for(int i = son[u]; i >= 1; i--) {
        for(int j = sze[u]; j >= sze[v]; j--) f[i][j] = (f[i][j] + f[i - 1][j - sze[v] ]) % MOD;
    }

    // delete
    for(int i = 1; i <= son[u]; i++) {
        for(int j = sze[v]; j <= sze[u]; j++) f[i][j] = (f[i][j] - f[i - 1][j - sze[v] ] + MOD) % MOD;
    }
}
```

1.3 斜率优化

其一般形式为 $b_i = min_{j<i}\{y_j - k_i x_j\}$ 或 $b_i = max_{j<i}\{y_j - k_i x_j\}$

```
int head, tail, Q[MAXN];
```

```
int X(int i) {}
int Y(int i) {}
void Init() {}

int Find(int k) {
    int l = head, r = tail;
    while(l < r) {
        int mid = l + r + 1 >> 1;
        int i = Q[mid - 1], j = Q[mid];
        double slope = 1.0 * ( Y(j) - Y(i) ) / ( X(j) - X(i) );
        // 求min 维护下凸包
        if(slope > 1.0 * k) r = mid - 1;
        else l = mid;
        // 求max 维护上凸包
        if(slope < 1.0 * k) r = mid - 1;
        else l = mid;
    }
    return Q[l];
}

bool Check(int i, int j, int k) {
    double slope_ij = 1.0 * ( Y(j) - Y(i) ) / ( X(j) - X(i) );
    double slope_jk = 1.0 * ( Y(j) - Y(k) ) / ( X(j) - X(k) );
    return slope_jk > slope_ij; // 求min 维护下凸包
    return slope_jk > slope_ij; // 求max 维护上凸包
}

void Solve() {
    Init();
    head = 1, tail = 0, Q[++tail] = 0;
    for(int i = 1; i <= n; i++) {
        int j = Find(K[i]); // K[i]为斜率
        用j更新f[i];
        while(head < tail && !Check(Q[tail - 1], Q[tail], i) ) --tail;
        Q[++tail] = i;
    }
    return 0;
}
```

1.4 悬线法

```
/*
悬线法的用途：针对求给定矩阵中满足某条件的极大矩阵，比如“面积最大的长方形、正方形” “周长最长的矩形”等等。适合障碍点较密集的情况。
设给定一个n*m的矩形，其中存在障碍点若干。
维护三个二维数组，left,right,up数组。
left[i][j]：代表从(i, j)能到达的最左位置 初始left[i][j] = j;
right[i][j]：代表从(i, j)能到达的最右位置 初始right[i][j] = j;
up[i][j]：代表从(i, j)向上扩展最长长度 初始up[i][j] = 1;
*/

void Solve() {
    for(int i = 1; i <= n; i++) for(int j = 2; j <= m; j++) left[i][j] = right[i][j] = j, up[i][j] = 1;

    for(int i = 1; i <= n; i++) {
        for(int j = 2; j <= m; j++) if(map[i][j]和map[i][j - 1]不是障碍点) left[i][j] = left[i][j - 1];
        for(int j = m - 1; j >= 1; j--) if(map[i][j]和map[i][j + 1]不是障碍点) right[i][j] = right[i][j + 1];
    }

    int ans = 0;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            if(map[i][j]和map[i - 1][j]都不是限制点) {
                up[i][j] = up[i - 1][j] + 1;
```

```
            left[i][j] = max(left[i][j], left[i - 1][j]);
            right[i][j] = min(right[i][j], right[i - 1][j]);
        }
        int width = (right[i][j] - left[i][j] + 1);
        ans = max(ans, width * up[i][j]); //求矩形
    }
  }
}
```

1.5 最大子段和

```
// 在数列的一维方向找到一个连续的子数列，使该子数列的和最大
dp[i] = max(dp[i - 1] + a[i], a[i]);
```

1.6 SOS_DP

```
for(int i = 0; i < n; i++) {
  for(int j = 0; j < (1 << n); j++) {
    if((j >> i) & 1) f[j] += f[j ^ (1 << i)];
  }
}
```

2 数据结构

2.1 并查集

```
struct DSU { // 按秩合并 可撤销
  struct Node {
    int u, v, val;
  };

  int f[MAXN], dep[MAXN], top = 0;
  Node s[MAXN];

  void Init(int n) { for(int i = 1; i <= n; i++) f[i] = i, dep[i] = 0; }

  int Find(int u) { return u == f[u] ? u : Find(f[u]); }

  void Union(int u, int v) { // 按秩合并
    u = Find(u), v = Find(v);
    if(u == v) return ;
    if(dep[u] > dep[v]) swap(u, v);
    int val = (dep[u] == dep[v]);
    s[++top] = (Node){u, v, val};
    f[u] = v, dep[v] += val;
  }

  bool Same(int u, int v) { return Find(u) == Find(v); }

  void Undo(int cur) {
    while(top > cur) {
      int u = s[top].u, v = s[top].v, val = s[top].val; // dep[u] <= dep[v]
      f[u] = u, dep[v] -= val, top--;
    }
  }
}dsu;
```

2.2 可持久化并查集

```cpp
//#pragma GCC optimize(2)
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 200005;
const int MOD = 1e9 + 7;

struct Node {
    int fa, dep, ls, rs;
}tree[MAXN * 40];

int cnt, n, m, root[MAXN];

int Build(int l, int r) {
    int dir = ++cnt;
    if(l == r) { tree[dir].fa = l; return dir; }
    int mid = l + r >> 1;
    tree[dir].ls = Build(l, mid);
    tree[dir].rs = Build(mid + 1, r);
    return dir;
}

int ModifyDep(int rt, int l, int r, int loc, int val) {
    int dir = ++cnt;
    tree[dir] = tree[rt];
    if(l == r) { tree[dir].dep += val; return dir; }
    int mid = (l + r) >> 1;
    if(loc <= mid) tree[dir].ls = ModifyDep(tree[dir].ls, l, mid, loc, val);
    if(loc > mid) tree[dir].rs = ModifyDep(tree[dir].rs, mid + 1, r, loc, val);
    return dir;
}

int ModifyFa(int rt, int l, int r, int loc, int val) {
    int dir = ++cnt;
    tree[dir] = tree[rt];
    if(l == r) { tree[dir].fa = val; return dir; }
    int mid = (l + r) >> 1;
    if(loc <= mid) tree[dir].ls = ModifyFa(tree[dir].ls, l, mid, loc, val);
    if(loc > mid) tree[dir].rs = ModifyFa(tree[dir].rs, mid + 1, r, loc, val);
    return dir;
}

int QueryNode(int rt, int l, int r, int loc) {
    if(l == r) return rt;
    int mid = (l + r) >> 1;
    if(loc <= mid) return QueryNode(tree[rt].ls, l, mid, loc);
    if(loc > mid) return QueryNode(tree[rt].rs, mid + 1, r, loc);
}

int Find(int rt, int u) { // 返回值为祖先节点在主席树上对应节点编号
    int cur = QueryNode(rt, 1, n, u);
    if(u == tree[cur].fa) return cur;
    return Find(rt, tree[cur].fa);
}

void Union(int p, int u, int v) { // 按秩合并
    u = Find(root[p], u), v = Find(root[p], v);
    if(u == v) return ;
    if(tree[u].dep > tree[v].dep) swap(u, v);
    root[p] = ModifyFa(root[p], 1, n, tree[u].fa, tree[v].fa);
    if(tree[u].dep == tree[v].dep) root[p] = ModifyDep(root[p], 1, n, tree[v].fa, 1);
}

signed main()
```

```
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> n >> m; root[0] = Build(1, n);
    for(int i = 1; i <= m; i++) {
        root[i] = root[i - 1];
        int op, a, b, k; cin >> op;
        if(op == 1) cin >> a >> b, Union(i, a, b);
        else if(op == 2) cin >> k, root[i] = root[k];
        else cin >> a >> b, cout << (tree[ Find(root[i], a) ].fa == tree[ Find(root[i], b) ].fa) << "\n";
    }
    return 0;
}
```

2.3 K-D Tree

```
//#pragma GCC optimize(2)
#include<bits/stdc++.h>
#define x1 x123456789
#define y1 y123456789
using namespace std;

const int MAXN = 5e5 + 5;
const double alpha = 0.725;

struct node {
    int x, y, v;
}a[MAXN];

int n, cnt;
int rt, L[MAXN], R[MAXN], D[MAXN], U[MAXN], d[MAXN], sze[MAXN], sum[MAXN], ls[MAXN], rs[MAXN];
int g[MAXN], t;

bool cmp1(int x, int y) { return a[x].x < a[y].x; }
bool cmp2(int x, int y) { return a[x].y < a[y].y; }

void maintain(int p) {
    sze[p] = sze[ ls[p] ] + sze[ rs[p] ] + 1;
    sum[p] = sum[ ls[p] ] + sum[ rs[p] ] + a[p].v;
    L[p] = R[p] = a[p].x;
    D[p] = U[p] = a[p].y;
    if(ls[p]) {
        L[p] = min(L[p], L[ ls[p] ]), R[p] = max(R[p], R[ ls[p] ]);
        D[p] = min(D[p], D[ ls[p] ]), U[p] = max(U[p], U[ ls[p] ]);
    }
    if(rs[p]) {
        L[p] = min(L[p], L[ rs[p] ]), R[p] = max(R[p], R[ rs[p] ]);
        D[p] = min(D[p], D[ rs[p] ]), U[p] = max(U[p], U[ rs[p] ]);
    }
}

int build(int l, int r) {
    if(l > r) return 0;
    int mid = l + r >> 1;
    double ave1 = 0, ave2 = 0, var1 = 0, var2 = 0;
    for(int i = l; i <= r; i++) ave1 += a[ g[i] ].x, ave2 += a[ g[i] ].y;
    ave1 /= (r - l + 1), ave2 /= (r - l + 1);
    for(int i = l; i <= r; i++) {
        var1 += (a[ g[i] ].x - ave1) * (a[ g[i] ].x - ave1);
        var2 += (a[ g[i] ].y - ave2) * (a[ g[i] ].y - ave2);
    }
    if(var1 > var2) {
        nth_element(g + l, g + mid, g + r + 1, cmp1);
        d[ g[mid] ] = 1;
    } else {
```

```
50        nth_element(g + l, g + mid, g + r + 1, cmp2);
51        d[ g[mid] ] = 2;
52    }
53    ls[ g[mid] ] = build(l, mid - 1);
54    rs[ g[mid] ] = build(mid + 1, r);
55    maintain(g[mid]);
56    return g[mid];
57 }
58
59 void print(int p) {
60    if(!p) return ;
61    print(ls[p]);
62    g[++t] = p;
63    print(rs[p]);
64 }
65
66 void rebuild(int &p) {
67    t = 0;
68    print(p);
69    p = build(1, t);
70 }
71
72 bool bad(int p) {
73    return alpha * sze[p] <= (double) max(sze[ ls[p] ], sze[ rs[p] ]);
74 }
75
76 void insert(int &p, int k) {
77    if(!p) {
78        p = k;
79        maintain(p);
80        return ;
81    }
82    if(d[p] == 1) {
83        if(a[k].x <= a[p].x) insert(ls[p], k);
84        else insert(rs[p], k);
85    } else {
86        if(a[k].y <= a[p].y) insert(ls[p], k);
87        else insert(rs[p], k);
88    }
89    maintain(p);
90    if( bad(p) ) rebuild(p);
91 }
92
93 int query(int p, int xl, int yl, int xr, int yr) {
94    if(!p || xr < L[p] || xl > R[p] || yr < D[p] || yl > U[p]) return 0;
95    if(xl <= L[p] && R[p] <= xr && yl <= D[p] && U[p] <= yr) return sum[p];
96    int res = 0;
97    if(xl <= a[p].x && a[p].x <= xr && yl <= a[p].y && a[p].y <= yr) res += a[p].v;
98    return query(ls[p], xl, yl, xr, yr) + query(rs[p], xl, yl, xr, yr) + res;
99 }
100
101 signed main()
102 {
103    int lastans = 0, op, x1, y1, x2, y2, A;
104    scanf("%d", &n); cnt = 0;
105    while( scanf("%d", &op) ) {
106        if(op == 1) {
107            cin >> x1 >> y1 >> A;
108            x1 ^= lastans;
109            y1 ^= lastans;
110            A ^= lastans;
111            a[++cnt] = (node){x1, y1, A};
112            insert(rt, cnt);
113        }
114        if(op == 2) {
```

```
            cin >> x1 >> y1 >> x2 >> y2;
            x1 ^= lastans;
            y1 ^= lastans;
            x2 ^= lastans;
            y2 ^= lastans;
            printf("%d\n", lastans = query(rt, x1, y1, x2, y2) );
        }
        if(op == 3) break;
    }
    return 0;
}
```

2.4 珂朵莉树

```
#pragma GCC optimize(2)
#include<bits/stdc++.h>
#define ll long long
using namespace std;

const int MAXN = 1e5 + 5;

namespace ODT {
    struct node {
        ll l, r;
        mutable ll v;
        node(ll l, ll r, ll v) : l(l), r(r), v(v) {}
        bool operator < (const node& a) const { return l < a.l; }
    };

    set<node> tree;
    int n, q, sum;

    set<node>::iterator split(ll pos) {
        auto it = tree.lower_bound( node(pos, 0, 0) );
        if(it != tree.end() && it->l == pos) return it;
        it--;
        ll l = it->l, r = it->r, v = it->v;
        tree.erase(it);
        tree.insert( node(l, pos - 1, v) );
        return tree.insert( node(pos, r, v) ).first;
    }

    void assign(ll l, ll r, ll v) {
        int tot = 0, len = 0;
        auto end = split(r + 1), begin = split(l);

        for(auto it = begin; it != end; it++) {
            len += (it->r - it->l + 1);
            tot += it->v * (it->r - it->l + 1);
        }

        tree.erase(begin, end);
        tree.insert( node(l, r, v) );

        if(v == 1) sum += (len - tot);
        else sum -= tot;
    }

    // codeforces 915e
    void solve() {
        cin >> n >> q;
        tree.insert( node(1, n, 1) );
        sum = n;
        while(q--) {
```

```
            int l, r, k; cin >> l >> r >> k;
            assign(l, r, k == 1 ? 0 : 1);
            cout << sum << "\n";
        }
    }
}
using namespace ODT;

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    ODT::solve();
    return 0;
}
```

2.5 树状数组

```
int lowbit(int x) { return x & (-x); }

void Modify(int x, int k) {
    while(x <= n) c[x] += k, x += lowbit(x);
}

int Query(int x) {
    int res = 0;
    while(x > 0) res += c[x], x -= lowbit(x);
    return res;
}

// 树状数组二分 O(logn)求全局第k小
int Kth(int k) {
    int cnt = 0, res = 0;
    for(int i = log2(n); i >= 0; --i) {
        res += (1 << i);
        if(res >= n || cnt + c[res] >= k) res -= (1 << i);
        else cnt += c[res];
    }
    return res + 1;
}
```

2.6 ST 表

```
void Init() {
    for(int k = 1; (1 << k) <= n; k++) {
        for(int i = 1; i <= n; i++) {
            st[i][k] = max(st[i][k - 1], st[i + (1 << (k - 1) )][k - 1]);
        }
    }
}

int Query(int l, int r) {
    int t = floor(log2(r - l + 1));
    return max(st[l][t], st[r - (1 << t) + 1][t]);
}
```

2.7 单调队列

```
void QueryMin() {
    memset(Index, 0, sizeof(Index)); memset(Value, 0, sizeof(Value));
    int head = 1, tail = 0;
    for(int i = 1; i <= n; i++) {
```

```
        while(head <= tail && Index[head] < i - k + 1) head++;
        while(head <= tail && Value[tail] > a[i]) tail--;
        Index[++tail] = i;
        Value[tail] = a[i];
    }
}

void QueryMax() {
    memset(Index, 0, sizeof(Index)); memset(Value, 0, sizeof(Value));
    int head = 1, tail = 0;
    for(int i = 1; i <= n; i++) {
        while(head <= tail && Index[head] < i - k + 1) head++;
        while(head <= tail && Value[tail] < a[i]) tail--;
        Index[++tail] = i;
        Value[tail] = a[i];
    }
}

void Init() {
    scanf("%d%d", &n, &k);
    for(int i = 1; i <= n; i++) scanf("%d", &a[i]);
    QueryMin(); QueryMax();
}
```

2.8 单调栈

```
/*
定义函数f[i]代表数列中第i个元素之后第一个大于a[i]的元素的下标 求(f数组)
*/
void Query() {
    int top = 0;
    for(int i = 1; i <= n; i++) {
        while(top > 0 && Value[top] < a[i]) { //单调栈保持降序
            f[Index[top]] = i;
            top--;
        }
        Value[++top] = a[i];
        Index[top] = i;
    }
}
```

2.9 块状数据结构

2.9.1 分块

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = (int)50005;
int a[MAXN], tag[MAXN], ValueAllOne[MAXN], id[MAXN], sum[MAXN], n, S;
vector<int> b[MAXN];
pair<int, int> block[MAXN];

void Init() {
    S = sqrt(n);
    memset(id, -1, sizeof(id));
    for(int i = 1; i <= n; i++) id[i] = (i - 1) / S + 1;

    int j = 1;
    for(int i = id[1]; i <= id[n]; i++) {
        // 维护区间和
        block[i].first = j;
        for(; id[j] == i; j++) sum[i] += a[j];
        block[i].second = j - 1;
```

```
19
20          // 维护区间排序
21          for(; id[j] == i; j++) b[i].push_back(a[j]);
22          sort(b[i].begin(), b[i].end());
23      }
24  }
25
26  void PushDownSqrt(int cid) {
27      int l = block[cid].first, r = block[cid].second;
28      if(ValueAllOne[cid] == 1 || tag[cid] == 0) return ;
29
30      bool flag = true;
31      for(int i = l; i <= r; i++) {
32          for(int j = 1; j <= tag[cid]; j++) {
33              if(a[i] == 1) break;
34              a[i] = floor(sqrt(1.0 * a[i]));
35          }
36          if(a[i] > 1) flag = false;
37      }
38
39      tag[cid] = 0;
40      if(flag) ValueAllOne[cid] = 1;
41      sum[cid] = 0;
42      for(int i = l; i <= r; i++) sum[cid] += a[i];
43  }
44
45  void ModifySqrt(int l, int r) { // 区间元素都开根号
46      int lid = id[l], rid = id[r];
47      if(lid == rid) {
48          SqrtPushDown(lid);
49          for(int i = l; i <= r; i++) {
50              int cur = floor(sqrt(1.0 * a[i]));
51              sum[lid] -= a[i], sum[lid] += cur;
52              a[i] = cur;
53          }
54      } else {
55          PushDown(lid); PushDown(rid);
56          for(int i = l; id[i] == lid; i++) {
57              int cur = floor(sqrt(1.0 * a[i]));
58              sum[lid] -= a[i], sum[lid] += cur;
59              a[i] = cur;
60          }
61          for(int i = r; id[i] == rid; i--) {
62              int cur = floor(sqrt(1.0 * a[i]));
63              sum[rid] -= a[i], sum[rid] += cur;
64              a[i] = cur;
65          }
66          for(int i = lid + 1; i < rid; i++) ++tag[i];
67      }
68  }
69
70  int QuerySum(int l, int r) { // 区间求和
71      int lid = id[l], rid = id[r];
72      if(lid == rid) {
73          PushDown(lid);
74          int res = 0;
75          for(int i = l; i <= r; i++) res += a[i];
76          return res;
77      } else {
78          int res = 0;
79          PushDown(lid); PushDown(rid);
80          for(int i = l; id[i] == lid; i++) res += a[i];
81          for(int i = r; id[i] == rid; i--) res += a[i];
82          for(int i = lid + 1; i < rid; i++) PushDown(i), res += sum[i];
83          return res;
```

```
    }
}

void ModifyAdd(int l, int r, int c) { // 区间元素都加上c
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        b[lid].clear();
        for(int i = l; i <= r; i++) {
            a[i] += c;
            b[lid].push_back(a[i]);
        }
        for(int i = l - 1; id[i] == lid; i--) b[lid].push_back(a[i]);
        for(int i = r + 1; id[i] == lid; i++) b[lid].push_back(a[i]);
        sort(b[lid].begin(), b[lid].end());
    } else {
        b[lid].clear();
        for(int i = l; id[i] == lid; i++) {
            a[i] += c;
            b[lid].push_back(a[i]);
        }
        for(int i = l - 1; id[i] == lid; i--) b[lid].push_back(a[i]);
        sort(b[lid].begin(), b[lid].end());

        b[rid].clear();
        for(int i = r; id[i] == rid; i--) {
            a[i] += c;
            b[rid].push_back(a[i]);
        }
        for(int i = r + 1; id[i] == rid; i++) b[rid].push_back(a[i]);
        sort(b[rid].begin(), b[rid].end());

        for(int i = lid + 1; i < rid; i++) tag[i] += c;
    }
}

int QuerySumOfSmaller(int l, int r, int c) { // 区间查询小于c的数字个数
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        int res = 0;
        for(int i = l; i <= r; i++) {
            res += (a[i] + tag[lid] < c);
        }
        return res;
    } else {
        int res = 0;
        for(int i = l; id[i] == lid; i++) {
            res += (a[i] + tag[lid] < c);
        }
        for(int i = r; id[i] == rid; i--) {
            res += (a[i] + tag[rid] < c);
        }
        for(int i = lid + 1; i < rid; i++) {
            res += lower_bound(b[i].begin(), b[i].end(), c - tag[i]) - b[i].begin();
        }
        return res;
    }
}

int QueryPre(int l, int r, int c) { // 区间内查询c的前驱(比其小的最大元素)
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        int res = -INF;
        for(int i = l; i <= r; i++) {
            if(a[i] + tag[lid] < c) res = max(res, a[i] + tag[lid]);
        }
```

```
        return res;
    } else {
        int res = -INF;
        for(int i = l; id[i] == lid; i++) {
            if(a[i] + tag[lid] < c) res = max(res, a[i] + tag[lid]);
        }
        for(int i = r; id[i] == rid; i--) {
            if(a[i] + tag[rid] < c) res = max(res, a[i] + tag[rid]);
        }
        for(int i = lid + 1; i < rid; i++) {
            int cur = lower_bound(b[i].begin(), b[i].end(), c - tag[i]) - b[i].begin() - 1;
            if(cur >= 0) res = max(res, b[i][cur] + tag[i]);
        }
        return res;
    }
}

signed main()
{
    Init();
    //Solve();
    return 0;
}
```

2.9.2 分块求区间众数

```
//#pragma GCC optimize(2)
#include<bits/stdc++.h>
#define int long long
#define pir make_pair
#define pii pair<int, int>
#define fi first
#define se second
using namespace std;
const int MAXN = 40005;
const int MAXS = 205;
const int MOD = 1e9 + 7;

int n, m, s, a[MAXN], val[MAXN], id[MAXN], l[MAXS], r[MAXS], c[MAXN], vis[MAXN], sum[MAXS][MAXN], ans[MAXS][MAXS
    ];
map<int, int> Map;

void Init() {
    int cnt = 0;
    for(auto &i : Map) i.se = ++cnt, val[i.se] = i.fi;
    for(int i = 1; i <= n; i++) a[i] = Map[ a[i] ];

    s = sqrt(n);
    for(int i = 1; i <= n; i++) id[i] = (i - 1) / s + 1, r[ id[i] ] = i;
    for(int i = n; i >= 1; i--) l[ id[i] ] = i;

    for(int i = 1; i <= n; i++) sum[ id[i] ][ a[i] ]++;
    for(int i = 1; i <= id[n]; i++) for(int j = 1; j <= n; j++) sum[i][j] += sum[i - 1][j];

    for(int i = 1; i <= id[n]; i++) {
        int res = 0;
        for(int j = i; j >= 1; j--) {
            for(int k = l[j]; k <= r[j]; k++) {
                c[ a[k] ]++;
                if(c[ a[k] ] > c[res] || (c[ a[k] ] == c[res] && a[k] < res) ) res = a[k];
            }
            ans[j][i] = res;
        }
        for(int j = 1; j <= n; j++) c[j] = 0;
```

```
    }
}

int Query(int L, int R) {
    int lid = id[L], rid = id[R], res = 0;
    if(rid - lid + 1 <= 2) {
        res = 0;
        for(int i = L; i <= R; i++) {
            c[ a[i] ]++;
            if(c[ a[i] ] > c[res] || (c[ a[i] ] == c[res] && a[i] < res) ) res = a[i];
        }
        for(int i = L; i <= R; i++) c[ a[i] ] = 0;
    } else {
        res = ans[lid + 1][rid - 1];
        c[res] += sum[rid - 1][res] - sum[lid][res];
        vis[res] = 1;

        for(int i = L; i <= r[lid]; i++) c[ a[i] ]++;
        for(int i = l[rid]; i <= R; i++) c[ a[i] ]++;

        for(int i = L; i <= r[lid]; i++) {
            if(!vis[ a[i] ]) {
                c[ a[i] ] += sum[rid - 1][ a[i] ] - sum[lid][ a[i] ];
                vis[ a[i] ] = 1;
                if(c[ a[i] ] > c[res] || (c[ a[i] ] == c[res] && a[i] < res) ) res = a[i];
            }
        }
        for(int i = l[rid]; i <= R; i++) {
            if(!vis[ a[i] ]) {
                c[ a[i] ] += sum[rid - 1][ a[i] ] - sum[lid][ a[i] ];
                vis[ a[i] ] = 1;
                if(c[ a[i] ] > c[res] || (c[ a[i] ] == c[res] && a[i] < res) ) res = a[i];
            }
        }

        for(int i = L; i <= r[lid]; i++) c[ a[i] ] = vis[ a[i] ] = 0;
        for(int i = l[rid]; i <= R; i++) c[ a[i] ] = vis[ a[i] ] = 0;
        c[ ans[lid + 1][rid - 1] ] = vis[ ans[lid + 1][rid - 1] ] = 0;
    }
    return res;
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i], Map[ a[i] ] = 1;
    Init();

    int x = 0;
    while(m--) {
        int l, r; cin >> l >> r;
        l = (l + x - 1) % n + 1, r = (r + x - 1) % n + 1;
        if(l > r) swap(l, r);
        x = val[ Query(l, r) ];
        cout << x << "\n";
    }
    return 0;
}
```

2.9.3 块状链表

```
//#pragma GCC optimize(2)
#include<bits/stdc++.h>
```

```
using namespace std;
const int MAXN = 2010, MAXM = 2010;

struct Node {
    char ch[MAXM + 5];
    int sze, lnode, rnode;
    void push_back(char c) { ch[sze++] = c; }
}p[MAXN + 5];

int pos1 = 0, pos2 = 0; //光标在块中/块内位置
int id[MAXN + 5], idx = 0; //可分配的编号池

void Add(int u, int v) { //将节点v插到节点u的右边
    p[v].rnode = p[u].rnode;
    p[ p[v].rnode ].lnode = v;
    p[u].rnode = v;
    p[v].lnode = u;
}

void Del(int u) { //删除节点u
    p[ p[u].lnode ].rnode = p[u].rnode;
    p[ p[u].rnode ].lnode = p[u].lnode;
    p[u].lnode = p[u].rnode = p[u].sze = 0;
    id[++idx] = u;
}

void Merge() {
    for(int k = p[0].rnode; k; k = p[k].rnode) {
        while(p[k].rnode && p[k].sze + p[ p[k].rnode ].sze < MAXM) {
            int rnode = p[k].rnode;
            if(pos1 == rnode) pos1 = k, pos2 += p[k].sze; // 与下一条语句顺序不能调换
            for(int i = 0; i < p[rnode].sze; i++) p[k].push_back( p[rnode].ch[i] );
            Del(rnode);
        }
    }
}

void Move(int k) { //移动到第k个字符后面
    pos1 = p[0].rnode;
    while (k > p[pos1].sze) k -= p[pos1].sze, pos1 = p[pos1].rnode;
    pos2 = k - 1;
}

void Insert(string S, int n) { //在光标后面插入字符串S, 长度为n
    if(pos2 + 1 != p[pos1].sze) { //分裂
        int u = id[idx--];
        for(int i = pos2 + 1; i < p[pos1].sze; i++) p[u].push_back( p[pos1].ch[i] );
        p[pos1].sze = pos2 + 1;
        Add(pos1, u);
    }
    int cur = pos1, i = 0;
    while(i < n) {
        int u = id[idx--];
        for(; i < n && p[u].sze < MAXN; i++) p[u].push_back(S[i]);
        Add(cur, u);
        cur = u;
    }
    Merge();
}

void Delete(int n) { //删除光标后的n个字符
    if(pos2 + 1 + n <= p[pos1].sze) {
        for(int i = pos2 + 1, j = pos2 + 1 + n; j < p[pos1].sze; i++, j++) {
            p[pos1].ch[i] = p[pos1].ch[j];
        }
```

```
        p[pos1].sze -= n;
    } else {
        n -= (p[pos1].sze - pos2 - 1);
        p[pos1].sze = pos2 + 1;
        while(p[pos1].rnode && n >= p[ p[pos1].rnode ].sze) {
            n -= p[ p[pos1].rnode ].sze;
            Del(p[pos1].rnode);
        }
        int u = p[pos1].rnode;
        for(int i = 0, j = n; j < p[u].sze; i++, j++) {
            p[u].ch[i] = p[u].ch[j];
        }
        p[u].sze -= n;
    }
    Merge();
}

void Get(int n) { //获取光标后n个字母
    if(pos2 + 1 + n <= p[pos1].sze) {
        for(int i = pos2 + 1; i <= pos2 + n; i++) cout << p[pos1].ch[i];
    } else {
        n -= (p[pos1].sze - pos2 - 1);
        for(int i = pos2 + 1; i < p[pos1].sze; i++) cout << p[pos1].ch[i];
        int cur = pos1;
        while(p[cur].rnode && n >= p[ p[cur].rnode ].sze) {
            n -= p[ p[cur].rnode ].sze;
            for(int i = 0; i < p[ p[cur].rnode ].sze; i++) cout << p[ p[cur].rnode ].ch[i];
            cur = p[cur].rnode;
        }
        int u = p[cur].rnode;
        for(int i = 0; i < n; i++) cout << p[u].ch[i];
    }
    cout << "\n";
}

void Prev() { //光标前移
    if(pos2) pos2--;
    else pos1 = p[pos1].lnode, pos2 = p[pos1].sze - 1;
}

void Next() { //光标后移
    if(pos2 != p[pos1].sze - 1) pos2++;
    else pos1 = p[pos1].rnode, pos2 = 0;
}

void Solve() {
    for(int i = 1; i <= MAXN; i++) id[++idx] = i;
    Insert("\n", 1); Move(1); //预防越界
    int t; cin >> t;
    while(t--) {
        string op, S;
        int n;
        cin >> op;
        if(op == "Insert") {
            cin >> n;
            int cur = 0; S = "";
            while(cur < n) {
                char ch = getchar();
                if(32 <= ch && ch <= 126) S.push_back(ch), cur++;
            }
            Insert(S, n);
        }
        else if(op == "Move") cin >> n, Move(n + 1);
        else if(op == "Delete") cin >> n, Delete(n);
        else if(op == "Get") cin >> n, Get(n);
```

```
        else if(op == "Prev") Prev();
        else if(op == "Next") Next();
    }
}

signed main() { Solve(); return 0; }
```

2.9.4 块状数组

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = (int)50005;
int a[MAXN], tag[MAXN], ValueAllOne[MAXN], id[MAXN], sum[MAXN], n, S;
vector<int> b[MAXN];
pair<int, int> block[MAXN];

void Init() {
    S = sqrt(n);
    memset(id, -1, sizeof(id));
    for(int i = 1; i <= n; i++) id[i] = (i - 1) / S + 1;

    int j = 1;
    for(int i = id[1]; i <= id[n]; i++) {
        // 维护区间和
        block[i].first = j;
        for(; id[j] == i; j++) sum[i] += a[j];
        block[i].second = j - 1;

        // 维护区间排序
        for(; id[j] == i; j++) b[i].push_back(a[j]);
        sort(b[i].begin(), b[i].end());
    }
}

void PushDownSqrt(int cid) {
    int l = block[cid].first, r = block[cid].second;
    if(ValueAllOne[cid] == 1 || tag[cid] == 0) return ;

    bool flag = true;
    for(int i = l; i <= r; i++) {
        for(int j = 1; j <= tag[cid]; j++) {
            if(a[i] == 1) break;
            a[i] = floor(sqrt(1.0 * a[i]));
        }
        if(a[i] > 1) flag = false;
    }

    tag[cid] = 0;
    if(flag) ValueAllOne[cid] = 1;
    sum[cid] = 0;
    for(int i = l; i <= r; i++) sum[cid] += a[i];
}

void ModifySqrt(int l, int r) { // 区间元素开根号
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        SqrtPushDown(lid);
        for(int i = l; i <= r; i++) {
            int cur = floor(sqrt(1.0 * a[i]));
            sum[lid] -= a[i], sum[lid] += cur;
            a[i] = cur;
        }
    } else {
        PushDown(lid); PushDown(rid);
```

```
        for(int i = l; id[i] == lid; i++) {
            int cur = floor(sqrt(1.0 * a[i]));
            sum[lid] -= a[i], sum[lid] += cur;
            a[i] = cur;
        }
        for(int i = r; id[i] == rid; i--) {
            int cur = floor(sqrt(1.0 * a[i]));
            sum[rid] -= a[i], sum[rid] += cur;
            a[i] = cur;
        }
        for(int i = lid + 1; i < rid; i++) ++tag[i];
    }
}

int QuerySum(int l, int r) { // 区间求和
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        PushDown(lid);
        int res = 0;
        for(int i = l; i <= r; i++) res += a[i];
        return res;
    } else {
        int res = 0;
        PushDown(lid); PushDown(rid);
        for(int i = l; id[i] == lid; i++) res += a[i];
        for(int i = r; id[i] == rid; i--) res += a[i];
        for(int i = lid + 1; i < rid; i++) PushDown(i), res += sum[i];
        return res;
    }
}

void ModifyAdd(int l, int r, int c) { // 区间元素都+c
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        b[lid].clear();
        for(int i = l; i <= r; i++) {
            a[i] += c;
            b[lid].push_back(a[i]);
        }
        for(int i = l - 1; id[i] == lid; i--) b[lid].push_back(a[i]);
        for(int i = r + 1; id[i] == lid; i++) b[lid].push_back(a[i]);
        sort(b[lid].begin(), b[lid].end());
    } else {
        b[lid].clear();
        for(int i = l; id[i] == lid; i++) {
            a[i] += c;
            b[lid].push_back(a[i]);
        }
        for(int i = l - 1; id[i] == lid; i--) b[lid].push_back(a[i]);
        sort(b[lid].begin(), b[lid].end());

        b[rid].clear();
        for(int i = r; id[i] == rid; i--) {
            a[i] += c;
            b[rid].push_back(a[i]);
        }
        for(int i = r + 1; id[i] == rid; i++) b[rid].push_back(a[i]);
        sort(b[rid].begin(), b[rid].end());

        for(int i = lid + 1; i < rid; i++) tag[i] += c;
    }
}

int QuerySumOfSmaller(int l, int r, int c) { // 区间查询小于c的数字个数
    int lid = id[l], rid = id[r];
```

```
    if(lid == rid) {
        int res = 0;
        for(int i = l; i <= r; i++) {
            res += (a[i] + tag[lid] < c);
        }
        return res;
    } else {
        int res = 0;
        for(int i = l; id[i] == lid; i++) {
            res += (a[i] + tag[lid] < c);
        }
        for(int i = r; id[i] == rid; i--) {
            res += (a[i] + tag[rid] < c);
        }
        for(int i = lid + 1; i < rid; i++) {
            res += lower_bound(b[i].begin(), b[i].end(), c - tag[i]) - b[i].begin();
        }
        return res;
    }
}

int QueryPre(int l, int r, int c) { // 区间内查询c的前驱(比其小的最大元素)
    int lid = id[l], rid = id[r];
    if(lid == rid) {
        int res = -INF;
        for(int i = l; i <= r; i++) {
            if(a[i] + tag[lid] < c) res = max(res, a[i] + tag[lid]);
        }
        return res;
    } else {
        int res = -INF;
        for(int i = l; id[i] == lid; i++) {
            if(a[i] + tag[lid] < c) res = max(res, a[i] + tag[lid]);
        }
        for(int i = r; id[i] == rid; i--) {
            if(a[i] + tag[rid] < c) res = max(res, a[i] + tag[rid]);
        }
        for(int i = lid + 1; i < rid; i++) {
            int cur = lower_bound(b[i].begin(), b[i].end(), c - tag[i]) - b[i].begin() - 1;
            if(cur >= 0) res = max(res, b[i][cur] + tag[i]);
        }
        return res;
    }
}

signed main()
{
    Init();
    //Solve();
    return 0;
}
```

2.9.5 莫队

```
#include<bits/stdc++.h>
//#pragma GCC optimize(2)

using namespace std;

const int MAXN = 50000 + 5;
int BLOCK_SIZE;

struct Query {
    int l, r, id;
```

```
    bool operator < (const Query &a) const {
        if(l / BLOCK_SIZE != a.l / BLOCK_SIZE) return l < a.l;
        return (l / BLOCK_SIZE) & 1 ? r < a.r : r > a.r;
    }
}q[MAXN];

int n, m, col[MAXN], cnt[MAXN];
long long sum = 0;
pair<long long, long long> ans[MAXN];

void Update(int c, int num) {
    sum -= 1ll * cnt[c] * (cnt[c] - 1) / 2;
    cnt[c] += num;
    sum += 1ll * cnt[c] * (cnt[c] - 1) / 2;
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    BLOCK_SIZE = (int)ceil( sqrt(1.0 * n) );
    for(int i = 1; i <= n; i++) cin >> col[i];
    for(int i = 1; i <= m; i++) cin >> q[i].l >> q[i].r, q[i].id = i;
    sort(q + 1, q + m + 1);
    for(int i = 1, l = 1, r = 0; i <= m; i++) {
        if(q[i].l == q[i].r) {
            ans[ q[i].id ] = make_pair(0ll, 1ll);
            continue;
        }
        while(l > q[i].l) Update(col[--l], 1);
        while(r < q[i].r) Update(col[++r], 1);
        while(l < q[i].l) Update(col[l++], -1);
        while(r > q[i].r) Update(col[r--], -1);
        long long tot = 1ll * (r - l + 1) * (r - l) / 2;
        ans[ q[i].id ] = make_pair(sum, tot);
    }
    for(int i = 1; i <= m; i++) {
        if(ans[i].first) {
            long long g = __gcd(ans[i].first, ans[i].second);
            ans[i].first /= g, ans[i].second /= g;
        } else ans[i].second = 1;
        cout << ans[i].first << "/" << ans[i].second << "\n";
    }
    return 0;
}
```

2.10 线段树

```
/* 区间加法 区间乘法 区间求和 */

#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int MAXN = (int)1e5 + 5;

int n, m, MOD;
ll mul[MAXN << 2], add[MAXN << 2], sum[MAXN << 2], a[MAXN];

inline int ls(int x) { return x << 1; }
inline int rs(int x) { return x << 1 | 1; }

void PushUp(int p) {
    sum[p] = (sum[ls(p)] + sum[rs(p)]) % MOD;
}
```

```

void PushDown(int p, int l, int r) {
    int mid = l + r >> 1;
    mul[ls(p)] = mul[ls(p)] * mul[p] % MOD;
    mul[rs(p)] = mul[rs(p)] * mul[p] % MOD;
    add[ls(p)] = (add[ls(p)] * mul[p] % MOD + add[p]) % MOD;
    add[rs(p)] = (add[rs(p)] * mul[p] % MOD + add[p]) % MOD;
    sum[ls(p)] = (sum[ls(p)] * mul[p] % MOD + add[p] * (mid - l + 1) % MOD) % MOD;
    sum[rs(p)] = (sum[rs(p)] * mul[p] % MOD + add[p] * (r - mid) % MOD) % MOD;
    add[p] = 0, mul[p] = 1;
}

void Build(int l, int r, int p) {
    add[p] = 0, mul[p] = 1; // significant
    if(l == r) {
        sum[p] = a[l];
        return ;
    }
    int mid = l + r >> 1;
    Build(l, mid, ls(p));
    Build(mid + 1, r, rs(p));
    PushUp(p);
}

void ModifyAdd(int nl, int nr, int l, int r, int p, ll k) {
    if(nl <= l && nr >= r) {
        add[p] = (add[p] + k) % MOD;
        sum[p] = (sum[p] + 1ll * (r - l + 1) * k % MOD) % MOD;
        return ;
    }
    PushDown(p, l, r);
    int mid = l + r >> 1;
    if(nl <= mid) ModifyAdd(nl, nr, l, mid, ls(p), k);
    if(nr > mid) ModifyAdd(nl, nr, mid + 1, r, rs(p), k);
    PushUp(p);
}

void ModifyMul(int nl, int nr, int l, int r, int p, ll k) {
    if(nl <= l && nr >= r) {
        mul[p] = mul[p] * k % MOD;
        add[p] = add[p] * k % MOD;
        sum[p] = sum[p] * k % MOD;
        return ;
    }
    PushDown(p, l, r);
    int mid = l + r >> 1;
    if(nl <= mid) ModifyMul(nl, nr, l, mid, ls(p), k);
    if(nr > mid) ModifyMul(nl, nr, mid + 1, r, rs(p), k);
    PushUp(p);
}

ll Query(int nl, int nr, int l, int r, int p) {
    if(nl <= l && nr >= r) return sum[p];
    PushDown(p, l, r);
    int mid = l + r >> 1, res = 0;
    if(nl <= mid) res = (res + Query(nl, nr, l, mid, ls(p)) ) % MOD;
    if(nr > mid) res = (res + Query(nl, nr, mid + 1, r, rs(p)) ) % MOD;
    return res;
}

signed main()
{
    scanf("%d%d%d", &n, &m, &MOD);
    for(int i = 1; i <= n; i++) scanf("%lld", &a[i]);
    Build(1, n, 1);
```

```
    while(m--) {
        int op, l, r; ll k;
        scanf("%d%d%d", &op, &l, &r);
        if(op == 1) {
            scanf("%lld", &k);
            ModifyMul(l, r, 1, n, 1, k);
        } else if(op == 2) {
            scanf("%lld", &k);
            ModifyAdd(l, r, 1, n, 1, k);
        } else {
            printf("%lld\n", Query(l, r, 1, n, 1) );
        }

    }
    return 0;
}
```

2.10.1 线段树合并分裂

```
// 对每个点开一棵线段树然后能维护的东西，线段树合并都能维护
#pragma GCC optimize(2)
#include<bits/stdc++.h>
#define int long long
#define pir make_pair
#define pii pair<int, int>
#define fi first
#define se second
using namespace std;
const int MAXN = 1e5 + 5;
const int MOD = 1e9 + 7;

int n, m, Op[MAXN];
set<int> Set;

struct Node {
    int ltype, rtype, ans, sze;
    friend Node operator + (Node a, Node b) {
        if(!a.sze) return b;
        if(!b.sze) return a;
        Node res;
        res.ans = a.ans + b.ans + (a.rtype ^ b.ltype);
        res.ltype = a.ltype;
        res.rtype = b.rtype;
        res.sze = a.sze + b.sze;
        return res;
    }
}tree[MAXN << 2];

struct Segment_Tree {
    int root[MAXN * 40], son[MAXN * 40][3];
    Node sum[MAXN * 40];
    int pool[MAXN * 40], delcnt = 0, cnt = 0;

    void PushUp(int p) { sum[p] = sum[ son[p][0] ] + sum[ son[p][1] ]; }

    int NewNode() { return delcnt ? pool[delcnt--] : ++cnt; }

    void DelNode(int p) {
        pool[++delcnt] = p;
        sum[p] = {0, 0, 0, 0};
        son[p][0] = son[p][1] = 0;
    }

    void Insert(int& p, int l, int r, int loc) {
```

```
        if(!p) p = NewNode();
        if(l == r) {
            sum[p].ltype = sum[p].rtype = (loc & 1);
            sum[p].ans = 0, sum[p].sze = 1;
            return ;
        }
        int mid = l + r >> 1;
        if(loc <= mid) Insert(son[p][0], l, mid, loc);
        else Insert(son[p][1], mid + 1, r, loc);
        PushUp(p);
    }

    int Merge(int u, int v, int l = 1, int r = n) {
        if(!u || !v) return u + v;
        if(l == r) {
            sum[u] = sum[u] + sum[v];
            DelNode(v);
            return u;
        }
        int mid = l + r >> 1;
        son[u][0] = Merge(son[u][0], son[v][0], l, mid);
        son[u][1] = Merge(son[u][1], son[v][1], mid + 1, r);
        DelNode(v);
        PushUp(u);
        return u;
    }

    void Split(int u, int& v, int k, int flag) { //把u节点分裂，得到新的放到v里面，分裂前k个数的节点
        if(!k) return ;
        v = NewNode();
        if(k >= sum[ son[u][flag] ].sze) {
            Split(son[u][flag ^ 1], son[v][flag ^ 1], k - sum[ son[u][flag] ].sze, flag);
            swap(son[u][flag], son[v][flag]);
        } else Split(son[u][flag], son[v][flag], k, flag);
        PushUp(u), PushUp(v);
    }
}S;

void Modify(int loc, int l, int r, int p) {
    if(l == r) {
        tree[p] = S.sum[ S.root[loc] ];
        if(Op[loc]) swap(tree[p].ltype, tree[p].rtype);
        return ;
    }
    int mid = l + r >> 1;
    if(loc <= mid) Modify(loc, l, mid, p << 1);
    else Modify(loc, mid + 1, r, p << 1 | 1);
    tree[p] = tree[p << 1] + tree[p << 1 | 1];
}

Node Query(int nl, int nr, int l, int r, int p) {
    if(nl <= l && nr >= r) return tree[p];
    int mid = l + r >> 1;
    if(nl > mid) return Query(nl, nr, mid + 1, r, p << 1 | 1);
    else if(nr <= mid) return Query(nl, nr, l, mid, p << 1);
    else return Query(nl, nr, l, mid, p << 1) + Query(nl, nr, mid + 1, r, p << 1 | 1);
}

set<int>::iterator Split(int x) {
    auto pos = Set.lower_bound(x), tmp = pos;
    if(*pos == x) return pos;
    pos--;
    S.Split(S.root[*pos], S.root[x], *tmp - x, Op[*pos] ^ 1);
    Modify(*pos, 1, n, 1);
    Op[x] = Op[*pos];
```

```
    Modify(x, 1, n, 1);
    Set.insert(x);
    return Set.lower_bound(x);
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> n >> m; Set.insert(n + 1);
    for(int i = 1; i <= n; i++) {
        int x; cin >> x;
        Set.insert(i);
        S.Insert(S.root[i], 1, n, x);
        Modify(i, 1, n, 1);
    }
    while(m--) {
        int op, l, r; cin >> op >> l >> r;
        auto pl = Split(l), pr = Split(r + 1);
        if(op == 3) {
            cout << Query(l, r, 1, n, 1).ans + 1 << "\n";
        } else {

            for(auto i = ++pl; i != pr; i++) {
                S.Merge(S.root[l], S.root[*i]);
                S.root[*i] = 0;
                Modify(*i, 1, n, 1);
            }
            Set.erase(pl, pr);
            Op[l] = op - 1;
            Modify(l, 1, n, 1);
        }
    }
    return 0;
}
```

2.10.2 扫描线

```
// 求n个矩形的面积并
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int MAXN = (int)1e5 + 5;

struct ScanLine {
    int x, lowy, highy, io; //io记录 入边/出边
    ScanLine(){}
    ScanLine(int x, int y1, int y2, int io) : x(x), lowy(y1), highy(y2), io(io){}
    bool operator < (const ScanLine &a) const { return x < a.x; }
}line[MAXN << 1];

int n, ans = 0, tot, cnt, yy[MAXN << 1];
int length[MAXN << 3], tag[MAXN << 3];

int ls(int x) { return x << 1; }
int rs(int x) { return x << 1 | 1; }

void PushUp(int p, int l, int r) {
    if(tag[p]) length[p] = yy[r] - yy[l];
    else if(l + 1 == r) length[p] = 0;
    else length[p] = length[ls(p)] + length[rs(p)];
}

void Build(int l, int r, int p) { //注意叶子节点的结构 (用区间表示 如[1,2] [2,3])
    if(l >= r) return ;
```

```
    tag[p] = length[p] = 0;
    if(l + 1 == r) return ;
    int mid = (l + r) >> 1;
    Build(l, mid, ls(p));
    Build(mid, r, rs(p));
}

void Update(int nl, int nr, int l, int r, int p, int io) {
    if(nl > r || nr < l) return ;
    if(nl <= l && nr >= r) {
        tag[p] += io;
        PushUp(p, l, r);
        return ;
    }
    if(l + 1 == r) return ;
    int mid = (l + r) >> 1;
    if(nl <= mid) Update(nl, nr, l, mid, ls(p), io);
    if(nr >= mid) Update(nl, nr, mid, r, rs(p), io);
    PushUp(p, l, r);
}

signed main()
{
    scanf("%lld", &n);
    cnt = 0;
    for(int i = 1; i <= n; i++) {
        int x1, y1, x2, y2;
        scanf("%lld%lld%lld%lld", &x1, &y1, &x2, &y2); //(x1, y1)为左下角坐标, (x2, y2)为右上角坐标
        line[++cnt] = ScanLine(x1, y1, y2, 1);
        yy[cnt] = y1;
        line[++cnt] = ScanLine(x2, y1, y2, -1);
        yy[cnt] = y2;
    }
    sort(yy + 1, yy + 1 + cnt);
    sort(line + 1, line + 1 + cnt);
    tot = unique(yy + 1, yy + 1 + cnt) - (yy + 1); // 离散化, tot记录去重后共多少y值
    ans = 0;
    Build(1, tot, 1);
    for(int i = 1; i <= cnt; i++) {
        ans += length[1] * (line[i].x - line[i - 1].x);
        int yl = lower_bound(yy + 1, yy + 1 + tot, line[i].lowy) - yy;
        int yr = lower_bound(yy + 1, yy + 1 + tot, line[i].highy) - yy;
        Update(yl, yr, 1, tot, 1, line[i].io);
    }
    printf("%lld", ans);
    return 0;
}
```

2.10.3 主席树

```
/*
1. 在某个历史版本上修改某一个位置上的值
2. 访问某个历史版本上的某一位置的值
*/

#include<bits/stdc++.h>
using namespace std;
const int MAXN = (int)1e6 + 5;
struct Node {
    int ls, rs, val;
}tree[MAXN << 5];
int n, m, a[MAXN], cnt, root[MAXN];

int Build(int l, int r) {
```

```
    int dir = ++cnt;
    if(l == r) {
        tree[dir].val = a[l];
        return dir;
    }
    int mid = (l + r) >> 1;
    tree[dir].ls = Build(l, mid);
    tree[dir].rs = Build(mid + 1, r);
    return dir;
}

int Modify(int p, int l, int r, int loc, int val) {
    int dir = ++cnt;
    tree[dir] = tree[p];
    if(l == r) {
        tree[dir].val = val;
        return dir;
    }
    int mid = (l + r) >> 1;
    if(loc <= mid) tree[dir].ls = Modify(tree[dir].ls, l, mid, loc, val);
    if(loc > mid) tree[dir].rs = Modify(tree[dir].rs, mid + 1, r, loc, val);
    return dir;
}

int Query(int p, int l, int r, int loc) {
    if(l == r) return tree[p].val;
    int mid = (l + r) >> 1;
    if(loc <= mid) return Query(tree[p].ls, l, mid, loc);
    if(loc > mid) return Query(tree[p].rs, mid + 1, r, loc);
}

signed main()
{
    scanf("%lld%lld", &n, &m);
    for(int i = 1; i <= n; i++) scanf("%lld", &a[i]);
    cnt = 0;
    root[0] = Build(1, n);
    for(int i = 1; i <= m; i++) {
        int v, op, loc, val;
        scanf("%lld%lld%lld", &v, &op, &loc);
        if(op == 1) {
            scanf("%lld", &val);
            root[i] = Modify(root[v], 1, n, loc, val);
        } else {
            printf("%lld\n", Query(root[v], 1, n, loc));
            root[i] = root[v];
        }
    }
    return 0;
}
```

2.10.4 主席树求静态区间第 k 小

```
/*
对权值线段树可持久化 再通过树上二分的方式找到[l,r]内第k小值
*/
#include<bits/stdc++.h>
using namespace std;
const int MAXN = (int)2e5 + 5;

struct Query {
    int x, y, l, r, k;
}q[MAXN];

```

```
void Init() { // 离散化
    sort(b + 1, b + 1 + tot);
    int cnt = 0;
    a[0].val = -0x3f3f3f3f; // significant
    for(int i = 1; i <= n; i++) {
        if(a[i].val != a[i - 1].val) ++cnt;
        a[i].hval = cnt;
        Hash[cnt] = a[i].val;
    }
    sort(a + 1, a + 1 + n, cmp2);
}

int Build(int l, int r) {
    int dir = ++cnt;
    tree[dir].val = 0;
    if(l == r) return dir;
    int mid = l + r >> 1;
    tree[dir].ls = Build(l, mid);
    tree[dir].rs = Build(mid + 1, r);
    return dir;
}

int Modify(int p, int l, int r, int loc) {
    int dir = ++cnt;
    tree[dir] = tree[p];
    tree[dir].val++;
    if(l == r) return dir;
    int mid = l + r >> 1;
    if(loc <= mid) tree[dir].ls = Modify(tree[dir].ls, l, mid, loc);
    if(loc > mid) tree[dir].rs = Modify(tree[dir].rs, mid + 1, r, loc);
    return dir;
}

int Query(int dl, int dr, int l, int r, int k) {
    if(l == r) return l;
    int mid = l + r >> 1;
    int x = tree[ tree[dr].ls ].val - tree[ tree[dl].ls ].val;
    if(x >= k) return Query(tree[dl].ls, tree[dr].ls, l, mid, k);
    else return Query(tree[dl].rs, tree[dr].rs, mid + 1, r, k - x);
}

signed main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i], b[++tot] = a[i];
    for(int i = 1; i <= m; i++) {
        char ch = getchar();
        if(ch == 'Q') cin >> q[i].l >> q[i].r >> q[i].k;
        else cin >> q[i].x >> q[i].y, b[++tot] = q[i].y;
    }
    Init();

    return 0;
}
```

2.10.5 李超线段树

```
#include<bits/stdc++.h>
#define int long long
#define fi first
#define se second
#define pir make_pair
#define reg register
#define pdi pair<double, int>
```

```
using namespace std;
const int MAXN = 100005;
const int INF = 0x7fffffff;
const int MOD1 = 39989;
const int MOD2 = (int)1e9;

struct Tree {
    int l, r, id;
}tree[MAXN << 2];
struct Seg {
    double k, b;
    double f(int x) { return 1.0 * k * x + b; }
    Seg() {}
    Seg(int x0, int y0, int x1, int y1) {
        if(x0 == x1) k = 0, b = max(y0, y1);
        else {
            k = 1.0 * (y0 - y1) / (x0 - x1);
            b = y0 - 1.0 * k * x0;
        }
    }
}s[MAXN];
int n;

inline int ls(int x) { return x << 1; }
inline int rs(int x) { return x << 1 | 1; }
pdi Max(pdi u, pdi v) {
    if(u.fi > v.fi) return u;
    else if(u.fi < v.fi) return v;
    else {
        if(u.se < v.se) return u;
        else return v;
    }
}

void Build(int l, int r, int p) {
    tree[p].l = l;
    tree[p].r = r;
    tree[p].id = 0;
    if(l == r) return ;
    int mid = l + r >> 1;
    Build(l, mid, ls(p));
    Build(mid + 1, r, rs(p));
}

void Modify(int u, int nl, int nr, int p) {
    int l = tree[p].l, r = tree[p].r, v = tree[p].id, mid = l + r >> 1;
    double fu = s[u].f(mid), fv = s[v].f(mid);
    if(nl <= l && nr >= r) {
        if(l == r) {
            if(fu > fv) tree[p].id = u;
            return ;
        }
        if(s[u].k > s[v].k) {
            if(fu > fv) tree[p].id = u, Modify(v, nl, nr, ls(p));
            else Modify(u, nl, nr, rs(p));
        } else if(s[u].k < s[v].k) {
            if(fu > fv) tree[p].id = u, Modify(v, nl, nr, rs(p));
            else Modify(u, nl, nr, ls(p));
        } else {
            if(s[u].b > s[v].b) tree[p].id = u;
        }
        return ;
    }
    if(nl <= mid) Modify(u, nl, nr, ls(p));
    if(nr > mid) Modify(u, nl, nr, rs(p));
```

```
}

pdi Query(int loc, int p) {
    int l = tree[p].l, r = tree[p].r, id = tree[p].id;
    if(l == r) return pir(s[id].f(loc), id);
    int mid = l + r >> 1;
    pdi u = pir(s[id].f(loc), id), v;
    if(loc <= mid) v = Query(loc, ls(p));
    else v = Query(loc, rs(p));
    return Max(u, v);
}

void Modify(int x0, int y0, int x1, int y1, int id) {
    s[id] = Seg(x0, y0, x1, y1);
    Modify(id, x0, x1, 1);
}

signed main()
{
    scanf("%lld", &n);
    Build(1, MOD1, 1);
    int last = 0, cnt = 0;
    for(int i = 1; i <= n; i++) {
        int op, k, x0, y0, x1, y1;
        scanf("%lld", &op);
        if(op == 0) {
            scanf("%lld", &k);
            k = (k + last - 1 + MOD1) % MOD1 + 1;
            last = Query(k, 1).se;
            printf("%lld\n", last);
        } else {
            scanf("%lld%lld%lld%lld", &x0, &y0, &x1, &y1);
            x0 = (x0 + last - 1 + MOD1) % MOD1 + 1;
            x1 = (x1 + last - 1 + MOD1) % MOD1 + 1;
            y0 = (y0 + last - 1 + MOD2) % MOD2 + 1;
            y1 = (y1 + last - 1 + MOD2) % MOD2 + 1;
            if(x0 > x1) swap(x0, x1), swap(y0, y1);
            Modify(x0, y0, x1, y1, ++cnt);
        }
    }
    return 0;
}
```

2.10.6 可持久化线段树（标记永久化）

```

#include<bits/stdc++.h>
#define int long long
using namespace std;

const int MAXN = 100000 + 5;
const int MOD = 998244353;

struct node {
    int s, e, p;
} a[MAXN];

struct treenode {
    int ls, rs, sum = 0, cnt = 0;
} tree[MAXN * 50];

int n, m, tot = 0, root[MAXN];
map<int, int> Map;

```

```cpp
int modify(int p, int nl, int nr, int v, int l = 1, int r = n) {
    int dir = ++tot;
    tree[dir] = tree[p];
    if(nl <= l && nr >= r) {
        tree[dir].sum += v;
        tree[dir].cnt += 1;
        return dir;
    }
    int mid = l + r >> 1;
    if(nl <= mid) tree[dir].ls = modify(tree[dir].ls, nl, nr, v, l, mid);
    if(nr > mid) tree[dir].rs = modify(tree[dir].rs, nl, nr, v, mid + 1, r);
    return dir;
}

pair<int, int> query(int p, int loc, int csum = 0, int ccnt = 0, int l = 1, int r = n) {
    if(l == r) return {csum + tree[p].sum, ccnt + tree[p].cnt};
    int mid = l + r >> 1;
    if(loc <= mid) return query(tree[p].ls, loc, csum + tree[p].sum, ccnt + tree[p].cnt, l, mid);
    else return query(tree[p].rs, loc, csum + tree[p].sum, ccnt + tree[p].cnt, mid + 1, r);
}

signed main() {
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> m >> n;
    for(int i = 1; i <= m; i++) cin >> a[i].s >> a[i].e >> a[i].p, Map[ a[i].p ] = 1;
    sort(a + 1, a + 1 + m, [&](const node &a, const node &b) { return a.p < b.p; });
    for(int i = 1; i <= m; i++) root[i] = modify(root[i - 1], a[i].s, a[i].e, a[i].p);
    int pre = 1;
    for(int i = 1; i <= n; i++) {
        int x, a, b, c, k; cin >> x >> a >> b >> c;
        k = 1 + (a * pre % c + b) % c;
        int l = 1, r = m;
        while(l < r) {
            int mid = l + r - 1 >> 1;
            if(query(root[mid], x).second < k) l = mid + 1;
            else r = mid;
        }
        pre = query(root[l], x).first;
        cout << pre << "\n";
    }
    return 0;
}
```

2.11 平衡树

2.11.1 普通平衡树

您需要写一种数据结构，来维护一些数，其中需要提供以下操作：

1. 插入 x 数
2. 删除 x 数 (若有多个相同的数，因只删除一个)
3. 查询 x 数的排名 (排名定义为比当前数小的数的个数 +1)
4. 查询排名为 x 的数
5. 求 x 的前驱 (前驱定义为小于 x，且最大的数)
6. 求 x 的后继 (后继定义为大于 x，且最小的数)

input:
10
1 106465
4 1
1 317721
1 460929
1 644985
1 84185
1 89851

6 81968
1 492737
5 493598

output:
106465
84185
492737

2.11.2 文艺平衡树

题目描述
这是一道模板题。
您需要写一种数据结构，来维护一个序列，其中需要提供以下操作：
翻转一个区间，例如原有序序列是 5 4 3 2 1，翻转区间是 [2,4] 的话，结果是 5 2 3 4 1。

输入格式
第一行为 n,m ，表示初始序列有 n 个数，这个序列依次是 1,2,...n-1,n，m 表示翻转操作次数。
接下来 m 行每行两个数 l,r

输出格式
输出一行 n 个数字，表示原始序列经过 m 次变换后的结果。

2.11.3 Treap 普通平衡树

```cpp
#include<bits/stdc++.h>

using namespace std;

const int SIZE = (int)1e5 + 5;
const int INF = 0x3f3f3f3f;

struct Treap {
    int l, r; // 左右子节点
    int val, dat; // 关键码、权值
    int cnt, size; // 副本数、子树大小
}a[SIZE];
int tot = 0, root, n;

int New(int val) { // 创建新节点
    a[++tot].val = val;
    a[tot].dat = rand();
    a[tot].cnt = a[tot].size = 1;
    return tot;
}

void Update(int p) { // 更新子树大小
    a[p].size = a[ a[p].l ].size + a[ a[p].r ].size + a[p].cnt;
}

void Build() { // 建树
    New(-INF), New(INF);
    root = 1, a[1].r = 2;
    Update(root);
}

void zig(int &p) { // 右旋
    int q = a[p].l;
    a[p].l = a[q].r, a[q].r = p, p = q;
    Update(a[p].r), Update(p);
}

void zag(int &p) { // 左旋
```

```
    int q = a[p].r;
    a[p].r = a[q].l, a[q].l = p, p = q;
    Update(a[p].l), Update(p);
}

void Insert(int &p, int val) { // 插入val数
    if(p == 0) {
        p = New(val);
        return ;
    }
    if(val == a[p].val) {
        a[p].cnt++, Update(p);
        return ;
    }

    if(val < a[p].val) {
        Insert(a[p].l, val);
        if( a[p].dat < a[ a[p].l ].dat ) zig(p); // 不满足大根堆性质 右旋
    } else {
        Insert(a[p].r, val);
        if( a[p].dat < a[ a[p].r ].dat ) zag(p); // 不满足大根堆性质 左旋
    }
    Update(p);
}

void Remove(int &p, int val) { // 删除val数
    if(p == 0) return ;
    if(val == a[p].val) {
        if(a[p].cnt > 1) {
            a[p].cnt--, Update(p);
            return ;
        }
        if(a[p].l || a[p].r) { // 不是叶子节点
            if(a[p].r == 0 || a[ a[p].l ].dat > a[ a[p].r ].dat )
                zig(p), Remove(a[p].r, val); // 右旋
            else
                zag(p), Remove(a[p].l, val); // 左旋

            Update(p);
        } else p = 0;
        return ;
    }

    val < a[p].val ? Remove(a[p].l, val) : Remove(a[p].r, val);
    Update(p);
}

int GetRankByVal(int p, int val) { // 查询val数的排名
    if(p == 0) return 0;
    if(val == a[p].val) return a[ a[p].l ].size + 1;
    if(val < a[p].val) return GetRankByVal(a[p].l, val);
    else return GetRankByVal(a[p].r, val) + a[ a[p].l ].size + a[p].cnt;
}

int GetValByRank(int p, int rank) { // 查询排名为rank的数
    if(p == 0) return 0;
    if(a[ a[p].l ].size >= rank) return GetValByRank(a[p].l, rank);
    else if(a[ a[p].l ].size + a[p].cnt >= rank) return a[p].val;
    else return GetValByRank(a[p].r, rank - a[ a[p].l ].size - a[p].cnt);
}

int GetPre(int val) { // 求val的前驱
    int ans = 1; // a[ans].val == -INF;
    int p = root;
    while(p) {
```

```
        if(val == a[p].val) {
            if(a[p].l > 0) {
                p = a[p].l;
                while(a[p].r > 0) p = a[p].r; // 左子树中一直向右走
                ans = p;
            }
            break;
        }
        if(a[p].val < val && a[p].val > a[ans].val) ans = p; // 更新答案

        val < a[p].val ? p = a[p].l : p = a[p].r;
    }
    return a[ans].val;
}

int GetNext(int val) { // 求val的后继
    int ans = 2; // a[ans].val = INF;
    int p = root;
    while(p) {
        if(val == a[p].val) {
            if(a[p].r > 0) {
                p = a[p].r;
                while(a[p].l > 0) p = a[p].l;
                ans = p;
            }
            break;
        }
        if(a[p].val > val && a[p].val < a[ans].val) ans = p;

        val < a[p].val ? p = a[p].l : p = a[p].r;
    }
    return a[ans].val;
}

signed main()
{
    Build();
    srand((unsigned)time(NULL));
    scanf("%d", &n);
    while(n--) {
        int opt, x;
        scanf("%d%d", &opt, &x);
        switch(opt) {
            case 1 : Insert(root, x); break;
            case 2 : Remove(root, x); break;
            case 3 : printf("%d\n", GetRankByVal(root, x) - 1 ); break; // 存在-INF
            case 4 : printf("%d\n", GetValByRank(root, x + 1) ); break; // 存在-INF
            case 5 : printf("%d\n", GetPre(x) ); break;
            case 6 : printf("%d\n", GetNext(x) ); break;
        }
    }
    return 0;
}
```

2.11.4 Splay 普通平衡树

```
#include<bits/stdc++.h>

using namespace std;

const int SIZE = (int)1e5 + 5;
const int INF = 0x3f3f3f3f;

int son[SIZE][2], fa[SIZE]; // 左右子节点(son[0]代表左儿子, son[1]代表右儿子)、父亲节点
```

```
int val[SIZE]; // 权值
int cnt[SIZE], size[SIZE]; // 副本数、子树大小
int tot = 0, root = 0; // 节点个数、根节点
int n;

void Update(int p) { // 更新子树大小
    size[p] = size[ son[p][0] ] + size[ son[p][1] ] + cnt[p];
}

bool CheckRson(int p) { // 判断节点p是不是右儿子
    return p == son[ fa[p] ][1];
}

void Clear(int p) { // 销毁节点p
    son[p][0] = son[p][1] = fa[p] = val[p] = cnt[p] = size[p] = 0;
}

void Rotate(int p) { //旋转p(根据p的儿子类型判断左旋还是右旋)
    int f = fa[p], gf = fa[f]; // f->father gf->grandfather
    bool isRson = CheckRson(p);
    son[f][isRson] = son[p][isRson ^ 1];
    if(son[p][isRson ^ 1]) fa[ son[p][isRson ^ 1] ] = f;
    son[p][isRson ^ 1] = f;
    fa[f] = p, fa[p] = gf;
    if(gf) son[gf][ f == son[gf][1] ] = p;
    Update(p), Update(f);
}

void Splay(int p, int target = 0) {  // 节点p旋转到节点target target=0表示根节点
    for(int f = fa[p]; (f = fa[p]) != target, f; Rotate(p)) {
        if(fa[f] != target) Rotate( CheckRson(p) == CheckRson(f) ? f : p );
    }
    if(!target) root = p;
}

int GetPre() { // 求x的前驱(将x插入 查询x左子树中最右边的节点 删除x)
    int cur = son[root][0];
    if(!cur) return cur;
    while(son[cur][1]) cur = son[cur][1];
    Splay(cur);
    return cur;
}

int GetNext() { // 求x的前驱(将x插入 查询x右子树中最左边的节点 删除x)
    int cur = son[root][1];
    if(!cur) return cur;
    while(son[cur][0]) cur = son[cur][0];
    Splay(cur);
    return cur;
}

int GetRankByVal(int k) { // 查询k数的排名
    int res = 0, cur = root;
    while(1) {
        if(k < val[cur]) cur = son[cur][0];
        else {
            res += size[ son[cur][0] ];
            if(k == val[cur]) {
                Splay(cur);
                return res + 1;
            }
            res += cnt[cur];
            cur = son[cur][1];
        }
    }
```

```
}

int GetValByRank(int k) { // 查询排名为k的数
    int cur = root;
    while(1) {
        if(son[cur][0] && k <= size[ son[cur][0] ]) cur = son[cur][0];
        else {
            k -= size[ son[cur][0] ] + cnt[cur];
            if(k <= 0) {
                Splay(cur);
                return val[cur];
            }
            cur = son[cur][1];
        }
    }
}

void Insert(int k) { // 插入k数
    if(!root) {
        val[++tot] = k, ++cnt[tot];
        root = tot, Update(root);
        return ;
    }
    int cur = root, f = 0;
    while(1) {
        if(val[cur] == k) {
            ++cnt[cur];
            Update(cur), Update(f);
            Splay(cur);
            break;
        }
        f = cur;
        cur = son[cur][ val[cur] < k ];
        if(cur == 0) {
            val[++tot] = k, ++cnt[tot];
            fa[tot] = f, son[f][ val[f] < k ] = tot;
            Update(tot), Update(f);
            Splay(tot);
            break;
        }
    }
}

void Remove(int k) { // 删除k数
    GetRankByVal(k);
    if(cnt[root] > 1) {
        --cnt[root];
        Update(root);
        return ;
    }
    if(son[root][0] == 0 && son[root][1] == 0) {
        Clear(root);
        root = 0;
    } else if(son[root][0] == 0) {
        int cur = root;
        root = son[root][1];
        fa[root] = 0;
        Clear(cur);
    } else if(son[root][1] == 0) {
        int cur = root;
        root = son[root][0];
        fa[root] = 0;
        Clear(cur);
    } else {
        int cur = root, pre = GetPre();
```

```
        fa[ son[cur][1] ] = pre;
        son[pre][1] = son[cur][1];
        Clear(cur);
        Update(root);
    }
}

signed main()
{
    scanf("%d", &n);
    while(n--) {
        int opt, x;
        scanf("%d%d", &opt, &x);
        switch(opt) {
            case 1 : Insert(x); break;
            case 2 : Remove(x); break;
            case 3 : printf("%d\n", GetRankByVal(x) ); break; // 存在-INF
            case 4 : printf("%d\n", GetValByRank(x) ); break; // 存在-INF
            case 5 : Insert(x), printf("%d\n", val[GetPre()] ), Remove(x); break;
            case 6 : Insert(x), printf("%d\n", val[GetNext()] ), Remove(x); break;
        }
    }
    return 0;
}
```

2.11.5 Splay 文艺平衡树

```
#include<bits/stdc++.h>
using namespace std;
const int SIZE = (int)1e5 + 5;
const int INF = 0x3f3f3f3f;

int son[SIZE][2], fa[SIZE]; // 左右子节点(son[0]代表左儿子, son[1]代表右儿子)、父亲节点
int val[SIZE]; // 权值
int cnt[SIZE], size[SIZE]; // 副本数、子树大小
bool tag[SIZE];  // 标记
int tot = 0, root = 0; // 节点个数、根节点
int n, m, a[SIZE];

void Update(int p) { // 更新子树大小
    size[p] = size[ son[p][0] ] + size[ son[p][1] ] + cnt[p];
}

bool CheckRson(int p) { // 判断节点p是不是右儿子
    return p == son[ fa[p] ][1];
}

void Clear(int p) { // 销毁节点p
    son[p][0] = son[p][1] = fa[p] = val[p] = cnt[p] = size[p] = 0;
}

void Rotate(int p) { //旋转p(根据p的儿子类型判断左旋还是右旋)
    int f = fa[p], gf = fa[f]; // f->father gf->grandfather
    bool isRson = CheckRson(p);
    son[f][isRson] = son[p][isRson ^ 1];
    if(son[p][isRson ^ 1]) fa[ son[p][isRson ^ 1] ] = f;
    son[p][isRson ^ 1] = f;
    fa[f] = p, fa[p] = gf;
    if(gf) son[gf][ f == son[gf][1] ] = p;
    Update(p), Update(f);
}

void Splay(int p, int target = 0) {  // 节点p旋转到节点target的儿子下面 target=0表示根节点
    for(int f; (f = fa[p]) != target; Rotate(p)) {
```

```
        if(fa[f] != target) Rotate( CheckRson(p) == CheckRson(f) ? f : p );
    }
    if(!target) root = p;
}

void PushDown(int p) { // 下传反转标记
    if(p && tag[p]) {
        int ls = son[p][0];
        int rs = son[p][1];
        tag[ls] ^= 1;
        tag[rs] ^= 1;
        swap(son[p][0], son[p][1]);
        tag[p] = 0;
    }
}

int FindNodeByRank(int k) { // 查询值为k的节点
    int cur = root;
    while(1) {
        PushDown(cur);
        if(son[cur][0] && k <= size[ son[cur][0] ]) cur = son[cur][0];
        else {
            k -= size[ son[cur][0] ] + 1;
            if(!k) return cur;
            else cur = son[cur][1];
        }
    }
}

int Build(int l, int r, int f) { // 建树
    if(l > r) return 0;
    int mid = l + r >> 1;
    int cur = ++tot;
    fa[cur] = f;
    cnt[cur] = 1;
    tag[cur] = 0;
    val[cur] = a[mid];
    son[cur][0] = Build(l, mid - 1, cur);
    son[cur][1] = Build(mid + 1, r, cur);
    Update(cur);
    return cur;
}

void Traverse(int p) { // 中序遍历
    if(!p) return ;
    PushDown(p);
    Traverse(son[p][0]);
    if(abs(val[p]) < INF) printf("%d ", val[p]);
    Traverse(son[p][1]);
}

void Reverse(int l, int r) { // 反转区间[l,r]
    int x = FindNodeByRank(l - 1 + 1); // 存在-INF
    int y = FindNodeByRank(r + 1 + 1);
    Splay(x, 0);
    Splay(y, x);
    PushDown(root);
    tag[ son[ son[root][1] ][0] ] ^= 1;
}

signed main()
{
    scanf("%d%d", &n, &m);
    a[1] = -INF, a[n + 2] = INF;
    for(int i = 1; i <= n; i++) a[i + 1] = i;
```

```
    root = Build(1, n + 2, 0);
    while(m--) {
        int l, r;
        scanf("%d%d", &l, &r);
        Reverse(l, r);
    }
    Traverse(root);
    return 0;
}
```

2.11.6 FHQ-Treap

```
#include<bits/stdc++.h>
using namespace std;

const int MaxSize = (int)1e5 + 5;
const int INF = 0x3f3f3f3f;

// T为pair等类型时需要重载运算符
template <typename T, int MAXN>
class Treap {
private:
    struct Node {
        T val;
        int ls, rs, sze, priority;
    } tree[MaxSize];
    int seed, tot, root;
    int Top, Stack[MaxSize];

    int rand() { return seed = (int)(seed * 104831ll % 0x7fffffff); }

    void pushup(int p) {
        if(p) tree[p].sze = tree[ tree[p].ls ].sze + tree[ tree[p].rs ].sze + 1;
    }

    int create(T val) {
        int p = Top ? Stack[Top--] : ++tot;
        tree[p].val = val;
        tree[p].sze = 1;
        tree[p].ls = tree[p].rs = 0;
        tree[p].priority = rand();
        return p;
    }

    // 将根为 p 的子树分裂成 x,y 两部分，x子树中全部小于等于val，y子树中全部大于val
    void split(int p, T val, int &x, int &y) {
        if(!p) return void(x = y = 0);
        if(val >= tree[p].val) {
            x = p;
            split(tree[p].rs, val, tree[p].rs, y);
        } else {
            y = p;
            split(tree[p].ls, val, x, tree[p].ls);
        }
        pushup(p);
    }
    /*
    // 将根为 p 的子树分裂成 x,y 两部分，x子树大小为sze
    void split(int p, int sze, int &x, int &y) {
        if(!p) return void(x = y = 0);
        if(tree[ tree[p].ls ].sze + 1 <= sze) {
            x = p;
            split(tree[p].rs, sze - (tree[ tree[p].ls ].sze + 1), tree[p].rs, y);
        } else {
```

```
53          y = p;
54          split(tree[p].ls, sze, x, tree[p].ls);
55      }
56      pushup(p);
57  }
58  */
59  int merge(int x, int y) {
60      if(!x || !y) return x + y;
61      if(tree[x].priority > tree[y].priority) {
62          tree[x].rs = merge(tree[x].rs, y);
63          pushup(x);
64          return x;
65      } else {
66          tree[y].ls = merge(x, tree[y].ls);
67          pushup(y);
68          return y;
69      }
70  }
71 public:
72  Treap() { seed = (int)(MAXN * 565463ll % 0x7fffffff); }
73
74  void insert(T val) {
75      int x, y;
76      split(root, val - 1, x, y);
77      root = merge( merge(x, create(val) ), y );
78  }
79
80  void remove(T val) {
81      int x, y, z;
82      split(root, val, x, z);
83      split(x, val - 1, x, y);
84      if(y) {
85          Stack[++Top] = y;
86          y = merge(tree[y].ls, tree[y].rs);
87      }
88      root = merge(merge(x, y), z);
89  }
90
91  int rank(T val) {
92      int x, y, res;
93      split(root, val - 1, x, y);
94      res = tree[x].sze + 1;
95      root = merge(x, y);
96      return res;
97  }
98
99  int val(int rank) {
100     int p = root;
101     while(1) {
102         if(tree[ tree[p].ls ].sze + 1 == rank) break;
103         if(tree[ tree[p].ls ].sze + 1 > rank) p = tree[p].ls;
104         else rank -= (tree[ tree[p].ls ].sze + 1), p = tree[p].rs;
105     }
106     return tree[p].val;
107 }
108
109 T prev(T val) {
110     int x, y, p; T res;
111     split(root, val - 1, x, y);
112     p = x;
113     while(tree[p].rs) p = tree[p].rs;
114     res = tree[p].val;
115     root = merge(x, y);
116     return res;
117 }
```

```
    T next(T val) {
        int x, y, p; T res;
        split(root, val, x, y);
        p = y;
        while(tree[p].ls) p = tree[p].ls;
        res = tree[p].val;
        root = merge(x, y);
        return res;
    }

    bool find(T val) {
        int x, y, z;
        split(root, val, x, z);
        split(x, val - 1, x, y);
        bool res = (tree[y].sze > 0);
        root = merge(merge(x, y), z);
        return res;
    }

    int size() { return tree[root].sze; }
}; Treap<int, 100005> fhqTreap;

signed main()
{
    int q, opt, x; scanf("%d", &q);
    while(q--) {
        scanf("%d%d", &opt, &x);
        switch(opt) {
            case 1 : fhqTreap.insert(x); break;
            case 2 : fhqTreap.remove(x); break;
            case 3 : printf("%d\n", fhqTreap.rank(x) ); break;
            case 4 : printf("%d\n", fhqTreap.val(x) ); break;
            case 5 : printf("%d\n", fhqTreap.prev(x) ); break;
            case 6 : printf("%d\n", fhqTreap.next(x) ); break;
        }
    }
    return 0;
}
```

2.12 树套树

2.12.1 树状数组套主席树求动态区间第 k 小

```
#pragma GCC optimize(2)
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 100000 + 5;
const int INF = 0x7fffffff;

struct Query {
    int l, r, k, x, y;
}q[MAXN];

map<int, int> Map;
vector<int> lRoot, rRoot;
int n, m, a[MAXN], cnt = 0, num[MAXN << 1], root[MAXN];
int tot, sum[MAXN << 7], ls[MAXN << 7], rs[MAXN << 7];

void Modify(int &rt, int l, int r, int loc, int val) {
    if(!rt) rt = ++tot;
    sum[rt] += val;
    if(l == r) return ;
    int mid = l + r >> 1;
```

```
    if(loc <= mid) Modify(ls[rt], l, mid, loc, val);
    else Modify(rs[rt], mid + 1, r, loc, val);
}

int QueryKth(int l, int r, int k) {
    if(l == r) return l;
    int mid = l + r >> 1, res = 0;
    for(auto i : rRoot) res += sum[ ls[i] ];
    for(auto i : lRoot) res -= sum[ ls[i] ];
    if(res >= k) {
        for(auto &i : lRoot) i = ls[i];
        for(auto &i : rRoot) i = ls[i];
        return QueryKth(l, mid, k);
    } else {
        for(auto &i : lRoot) i = rs[i];
        for(auto &i : rRoot) i = rs[i];
        return QueryKth(mid + 1, r, k - res);
    }
}

void Modify(int pos, int val) {
    int loc = Map[ a[pos] ];
    for( ; pos <= n; pos += (pos & (-pos))) Modify(root[pos], 1, cnt, loc, val);
}

int Query(int l, int r, int k) {
    lRoot.clear(), rRoot.clear();
    for(int pos = l - 1; pos; pos -= (pos & (-pos))) lRoot.push_back(root[pos]);
    for(int pos = r; pos; pos -= (pos & (-pos))) rRoot.push_back(root[pos]);
    return QueryKth(1, cnt, k);
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i], Map[ a[i] ] = 1;
    for(int i = 1; i <= m; i++) {
        char ch; cin >> ch;
        if(ch == 'Q') cin >> q[i].l >> q[i].r >> q[i].k;
        else cin >> q[i].x >> q[i].y, Map[ q[i].y ] = 1;
    }
    for(auto &i : Map) i.second = ++cnt, num[cnt] = i.first;
    for(int i = 1; i <= n; i++) Modify(i, 1);
    for(int i = 1; i <= m; i++) {
        if(q[i].l) cout << num[ Query(q[i].l, q[i].r, q[i].k) ] << "\n";
        else {
            Modify(q[i].x, -1);
            a[ q[i].x ] = q[i].y;
            Modify(q[i].x, 1);
        }
    }
    return 0;
}
```

3 字符串

3.1 字符串双哈希

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 6e5 + 10;
const int INF = 0x3f3f3f3f;
```

```cpp

namespace TwoHash {
    #define fi first
    #define se second
    #define pii pair<int, int>

    pii Pow[MAXN], Inv[MAXN];
    pii h[MAXN];
    int n;
    string S;

    const int BASE1 = 131;
    const int BASE2 = 129;
    const int MOD1 = 1e9 + 7;
    const int MOD2 = 998244353;

    int qpow(int a, int p, int MOD) {
        a %= MOD, p %= MOD;
        int res = 1;
        while(p) {
            if(p & 1) res = 1ll * res * a % MOD;
            a = 1ll * a * a % MOD;
            p >>= 1;
        }
        return res;
    }

    pii Hash(int c, int p) { return {1ll * c * Pow[p].fi % MOD1, 1ll * c * Pow[p].se % MOD2}; }

    pii Add(pii x, pii y) {
        pii res = {0, 0};
        res.fi = 1ll * (x.fi + y.fi) % MOD1;
        res.se = 1ll * (x.se + y.se) % MOD2;
        return res;
    }

    pii Sub(pii x, pii y) {
        pii res = {0, 0};
        res.fi = 1ll * (x.fi - y.fi + MOD1) % MOD1;
        res.se = 1ll * (x.se - y.se + MOD2) % MOD2;
        return res;
    }

    pii Mul(pii x, pii y) {
        pii res = {0, 0};
        res.fi = 1ll * x.fi * y.fi % MOD1;
        res.se = 1ll * x.se * y.se % MOD2;
        return res;
    }

    pii Div(pii x, pii y) {
        pii res = {0, 0};
        res.fi = 1ll * x.fi * qpow(y.fi, MOD1 - 2, MOD1) % MOD1;
        res.se = 1ll * x.se * qpow(y.se, MOD1 - 2, MOD1) % MOD1;
        return res;
    }

    pii HashVal(int l, int r) { return Mul( Sub(h[r], h[l - 1]), Inv[l - 1]); }

    void Init() {
        Pow[0] = Inv[0] = {1, 1}; Inv[1] = {qpow(BASE1, MOD1 - 2, MOD1), qpow(BASE2, MOD2 - 2, MOD2)};
        for(int i = 1; i <= MAXN - 5; i++) Pow[i] = Mul(Pow[i - 1], {BASE1, BASE2});
        for(int i = 2; i <= MAXN - 5; i++) Inv[i] = Mul(Inv[i - 1], Inv[1]);
    }

```

```
    void Build(string T) {
        n = T.length(); S = " " + T;
        h[0] = {0, 0};
        for(int i = 1; i <= n; i++) h[i] = Add(h[i - 1], Hash(S[i], i) );
    }
}

signed main()
{
    TwoHash::Init();
    set<pii> Set;
    int T; cin >> T;
    while(T--) {
        string S; cin >> S;
        TwoHash::Build(S);
        Set.insert( TwoHash::h[S.length()] );
    }
    cout << Set.size();
    return 0;
}
```

3.2 Trie

```
int Hash(char ch) { return ch - 'a' + 1; }

void Insert(string S) {
    int u = 0;
    for(auto i : S) {
        if(!trie[u][Hash(i)]) trie[u][Hash(i)] = ++cnt;
        u = trie[u][Hash(i)];
    }
    sum[u]++;
}
```

3.3 KMP

```
#include<bits/stdc++.h>

using namespace std;

const int MAXN = 1e6 + 5;

char S1[MAXN], S2[MAXN];
int l1, l2, pmt[MAXN], p; //pmt数组(部分匹配表) 向右偏移一位为next数组 并让next[0] = -1

int main()
{
    cin >> S1 + 1 >> S2 + 1;
    l1 = strlen(S1 + 1), l2 = strlen(S2 + 1);
    p = pmt[0] = 0;
    for(int i = 2; i <= l2; i++) {
        while(p && S2[p + 1] != S2[i]) p = pmt[p];
        if(S2[p + 1] == S2[i]) p++;
        pmt[i] = p;
    }
    p = 0;
    for(int i = 1; i <= l1; i++) {
        while(p && S2[p + 1] != S1[i]) p = pmt[p];
        if(S2[p + 1] == S1[i]) p++;
        if(p == l2) {
            printf("%d\n", i - p + 1);
            p = pmt[p];
```

```
        }
    }
    for(int i = 1; i <= l2; i++) printf("%d ", pmt[i]);
    return 0;
}
```

3.4 exKMP

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = (int)2e7 + 5;

char S[MAXN], T[MAXN];
int z[MAXN], lcp[MAXN];

void Z_Function(char* T) { // z[i] = lcp(s[i ... n-1], s)
    int n = strlen(T);
    z[0] = n;
    int l = 0, r = 0;

    for(int i = 1; i < n; i++) {
        if(i <= r) z[i] = min(z[i - l], r - i + 1);
        while(i + z[i] < n && T[z[i]] == T[i + z[i]]) z[i]++;
        if(i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    //for(int i = 0; i < n; i++) cout << z[i] << " "; cout << endl;
}

void exKMP(char* S, char* T) {
    int sLen = strlen(S);
    int tLen = strlen(T);
    Z_Function(T);

    int p = 0;
    while(S[p] == T[p] && p < min(sLen, tLen)) p++;
    lcp[0] = p;

    int l = 0, r = 0;
    for(int i = 1; i < sLen; i++) {
        if(i <= r) lcp[i] = min(z[i - l], r - i + 1);
        while(i + lcp[i] < sLen && lcp[i] < tLen && S[i + lcp[i]] == T[lcp[i]]) lcp[i]++;
        if(i + lcp[i] - 1 > r) l = i, r = i + lcp[i] - 1;
    }

    //for(int i = 0; i < sLen; i++) cout << lcp[i] << " "; cout << endl;
}

signed main()
{
    scanf("%s%s", S, T);
    exKMP(S, T);
    return 0;
}

/*
input:
aaaabaa
aaaaa

z function : {5 4 3 2 1}
lcp function : {4 3 2 1 0 2 1}
*/
```

3.5 Manacher

```
/*
Manacher算法:
先在两个字符串中插入某个字符('$') 避免分别处理奇回文和偶回文的情况
设置两个指针maxR(前i个字符能回文扩展到的最右端) pos(前i个字符中哪个字符能回文扩展到最右端)

每次扫描到第i个字符时
①如果i<maxR,更新f[i] ②暴力拓展maxR(maxR从起点到终点且不会往回退) ③maxR增大时更新maxR和pos
*/

#include<bits/stdc++.h>
using namespace std;

const int MAXN = (int)1.1e7 + 5;
char S[MAXN << 1], T[MAXN << 1];
int f[MAXN << 1], n;

void Manacher() {
    T[0] = '#'; T[1] = '$';
    for(int i = 1; i <= n; i++) T[i * 2] = S[i], T[i * 2 + 1] = '$';
    n = n * 2 + 1;
    for(int i = 0; i <= n; i++) S[i] = T[i];

    int maxR = 0, pos = 0;
    for(int i = 1; i <= n; i++) {
        if(i < maxR) f[i] = min(f[pos * 2 - i], maxR - i);
        while(i - f[i] - 1 > 0 && i + f[i] + 1 <= n && S[i + f[i] + 1] == S[i - f[i] - 1]) f[i]++;
        if(i + f[i] > maxR) maxR = i + f[i], pos = i;
    }
}

int main()
{
    scanf("%s", S + 1); n = strlen(S + 1);
    Manacher();
    int ans = 0;
    for(int i = 1; i <= n; i++) ans = max(ans, f[i]);
    printf("%d", ans);
    return 0;
}
```

3.6 最小最大表示法

```
/*
S的最小表示: 与S循环同构的所有字符串中字典序最小的字符串 (最大表示同理)
*/
int getMin(string S) {
    int n = S.length(), i = 0, j = 1, k = 0;
    while(i < n && j < n && k < n) {
        if(S[(i + k) % n] == S[(j + k) % n]) k++;
        else {
            if(S[(i + k) % n] > S[(j + k) % n]) i = i + k + 1;
            else j = j + k + 1;

            if(i == j) i++;
            k = 0;
        }
    }
    return min(i, j);
}

int getMax(string S) {
    int n = S.length(), i = 0, j = 1, k = 0;
```

```
    while(i < n && j < n && k < n) {
        if(S[(i + k) % n] == S[(j + k) % n]) k++;
        else {
            if(S[(i + k) % n] < S[(j + k) % n]) i = i + k + 1;
            else j = j + k + 1;

            if(i == j) i++;
            k = 0;
        }
    }
    return min(i, j);
}
```

3.7 AC 自动机

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = (int)1e6 + 5;

char S[MAXN], T[MAXN];
int n, cnt, trie[MAXN][30], fail[MAXN * 30], num[MAXN * 30];
queue<int> Q;

inline int Hash(char x) { return x - 'a' + 1; }

void Trie(char* S) {
    int cur = 1;
    int len = strlen(S);
    for(int i = 0; i < len; i++) {
        int x = Hash(S[i]);
        if(!trie[cur][x]) trie[cur][x] = ++cnt;
        cur = trie[cur][x];
    }
    num[cur]++;
}

void GetFail() {
    for(int i = 1; i <= 26; i++) trie[0][i] = 1;
    Q.push(1);
    fail[1] = 0;
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        int faFail = fail[u];
        for(int i = 1; i <= 26; i++) {
            int v = trie[u][i];
            if(v) fail[v] = trie[faFail][i], Q.push(v);
            else trie[u][i] = trie[faFail][i];
        }
    }
}

int main()
{
    scanf("%d", &n);
    cnt = 1;
    for(int i = 1; i <= n; i++) {
        scanf("%s", S);
        Trie(S);
    }
    GetFail();
    scanf("%s", T);

    int cur = 1, ans = 0, len = strlen(T);
```

```
    for(int i = 0; i < len; i++) {
        cur = trie[cur][Hash(T[i])];
        for(int t = cur; t && ~num[t]; t = fail[t]) ans += num[t], num[t] = -1;
    }
    printf("%d\n", ans);
    return 0;
}
```

3.8 后缀数组

```
/* 注意常数 */
//#pragma GCC optimize(2)
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 10;

char S[MAXN];
int n, m, sa[MAXN], rk[MAXN], oldrk[MAXN << 1], id[MAXN], px[MAXN], cnt[MAXN];

bool cmp(int x, int y, int w) {
    return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
}

void SuffixArray(char *s) {
    // getSA
    n = strlen(s + 1);
    int w, p, i, m = 300, k;
    for(i = 1; i <= n; ++i) ++cnt[ rk[i] = s[i] ];
    for(i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for(i = n; i >= 1; --i) sa[ cnt[ rk[i] ]-- ] = i;

    for(w = 1; ; w <<= 1, m = p) {
        for(p = 0, i = n; i > n - w; --i) id[++p] = i;
        for(i = 1; i <= n; ++i) if(sa[i] > w) id[++p] = sa[i] - w;
        for(i = 0; i <= m; ++i) cnt[i] = 0;
        for(i = 1; i <= n; ++i) ++cnt[ px[i] = rk[ id[i] ] ];
        for(i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
        for(i = n; i >= 1; --i) sa[ cnt[ px[i] ]-- ] = id[i];
        for(i = 1; i <= n; ++i) oldrk[i] = rk[i];
        for(p = 0, i = 1; i <= n; ++i) rk[ sa[i] ] = cmp(sa[i], sa[i - 1], w) ? p : ++p;

        if(p == n) {
            for(int i = 1; i <= n; ++i) sa[ rk[i] ] = i;
            break;
        }
    }
    //for(int i = 1; i <= n; ++i) cout << sa[i] << " "; cout << "\n";

    //getHeight
    for(i = 1, k = 0; i <= n; ++i) {
        if(rk[i] == 0) continue;
        if(k) --k;
        while(S[i + k] == S[ sa[ rk[i] - 1 ] + k ]) ++k;
        height[ rk[i] ] = k;
    }
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> S + 1;
    SuffixArray(S);
    return 0;
```

```
}
```

3.9 后缀自动机

```
#include<bits/stdc++.h>
#pragma GCC optimize("-Ofast")
using namespace std;
const int MAXN = (int)4e5 + 5;

struct state {
    int len, link, size;
    int next[30];
}st[MAXN << 1];

int sz, last, n, p, q;
long long f[MAXN];
char s[MAXN];

void init() {
    for(int i = 0; i < sz; i++) {
        for(int j = 0; j < 26; j++) st[i].next[j] = 0;
        st[i].len = st[i].link = st[i].size = 0;
    }
    sz = 0;

    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void extend(int c) {
    int cur = sz++;
    st[cur].size = 1;
    st[cur].len = st[last].len + 1;
    int p = last;
    while(p != -1 && !st[p].next[c]) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if(p == -1) st[cur].link = 0;
    else {
        int q = st[p].next[c];
        if(st[p].len + 1 == st[q].len) st[cur].link = q;
        else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            for(int i = 0; i < 26; i++) st[clone].next[i] = st[q].next[i];
            st[clone].link = st[q].link;
            while(p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

4 图论

4.1 DFS 序

```
/*
对于节点u 其子树范围 [ dfn[u], dfn[u] + sze[u] - 1 ]
可用线段树等数据结构维护
*/
void DFS(int u, int fa) {
    dfn[u] = ++cnt;
    sze[u] = 1;
    for(auto v : G[u]) {
        if(v == fa) continue;
        DFS(v, u);
        sze[u] += sze[v];
    }
}
```

4.2 LCA

```
int n, m, dep[MAXN], vis[MAXN], lca[MAXN][25];
vector<int> G[MAXN];

void DFS(int u, int ftr) {
    dep[u] = dep[ftr] + 1;
    vis[u] = 1;
    lca[u][0] = ftr;
    for(int i = 1; i <= 20; i++) {
        if(dep[u] < (1 << i)) break;
        lca[u][i] = lca[lca[u][i - 1]][i - 1];
    }
    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if(vis[v]) continue;
        DFS(v, u);
    }
}

int LCA(int x, int y) {
    int u = x, v = y;
    if(dep[u] > dep[v]) swap(u, v); //dep[u] <= dep[v]
    for(int i = 20; i >= 0; i--) {
        if((1 << i) & (dep[v] - dep[u])) v = lca[v][i];
    }
    for(int i = 20; i >= 0; i--) {
        if(lca[u][i] != lca[v][i]) {
            u = lca[u][i];
            v = lca[v][i];
        }
    }
    return u == v ? u : lca[u][0];
}
```

4.3 树的重心

```
int size[MAXN], weight[MAXN], centroid[3];

void GetCentroid(int cur, int fa) {
    size[cur] = 1;
    weight[cur] = 0;
    for(int i = head[cur]; i != -1; i = e[i].nxt) {
        if(e[i].to == fa) continue;
        GetCentroid(e[i].to, cur);
        size[cur] += size[ e[i].to ];
        weight[cur] = max(weight[cur], size[ e[i].to ]);
    }
```

```
    weight[cur] = max(weight[cur], n - size[cur]);
    if(weight[cur] <= n / 2) centroid[ ++centroid[0] ] = cur;
}
```

4.4 树的直径

```
int n, d1[MAXN], d2[MAXN], d = 0;
vector<int> G[MAXN];

void DFS(int u, int fa) {
    d1[u] = d2[u] = 0;
    for(auto v : G[u]) {
        if(v == fa) continue;
        DFS(v, u);
        int cur = d1[v] + 1;
        if(cur > d1[u]) d2[u] = d1[u], d1[u] = cur;
        else if(cur > d2[u]) d2[u] = cur;
    }
    d = max(d, d1[u] + d2[u]);
}
```

4.5 最小生成树

```
int Prim() {
    memset(dist, 0x3f, sizeof(dist));
    int res = 0;
    for(int i = 0; i < n; i++) {
        int t = -1;
        for(int j = 1; j <= n; j++) {
            if(!st[j] && (t == -1 || dist[t] > dist[j])) t = j;
        }
        if(i && dist[t] == INF) return INF; // 无解

        if(i) res += dist[t];
        st[t] = true;

        for(int j = 1; j <= n; j++) dist[j] = min(dist[j], G[t][j]);
    }
    return res;
}

int Kruskal() {
    for(int i = 1; i <= n; i++) f[i] = i;
    sort(e + 1, e + 1 + m, cmp); // 按边权排序
    ans = 0;
    for(int i = 1; i <= m; i++) {
        int fu = find(e[i].u), fv = find(e[i].v);
        if(fu == fv) continue;
        if(fu > fv) swap(fu, fv);
        f[fv] = fu;
        ans += e[i].w;
        e[i].flag = 1;
        G[ e[i].u ].push_back( make_pair(e[i].v, e[i].w) );
        G[ e[i].v ].push_back( make_pair(e[i].u, e[i].w) );
    }
    return ans;
}
```

4.6 严格次小生成树

```cpp
#include<bits/stdc++.h>
#define int long long
using namespace std;

const int MAXN = (int)1e5 + 5;
const int MAXM = (int)3e5 + 5;
struct Node {
   int u, v, w, flag;
}e[MAXM];
int lca[MAXN][30], maxx1[MAXN][30], maxx2[MAXN][30], dep[MAXN], n, m, f[MAXN], ans, vis[MAXN];
vector< pair<int, int> > G[MAXN];

bool cmp(Node &a, Node &b) { return a.w < b.w; }
int find(int x) { return ( x == f[x] ? x : ( f[x] = find(f[x]) ) ); }

void Kruskal() {
   for(int i = 1; i <= n; i++) f[i] = i;
   sort(e + 1, e + 1 + m, cmp);

   ans = 0;
   for(int i = 1; i <= m; i++) {
      int fu = find(e[i].u), fv = find(e[i].v);
      if(fu == fv) continue;
      if(fu > fv) swap(fu, fv);
      f[fv] = fu;
      ans += e[i].w;
      e[i].flag = 1;
      G[ e[i].u ].push_back( make_pair(e[i].v, e[i].w) ), G[ e[i].v ].push_back( make_pair(e[i].u, e[i].w) );
   }
}

void DFS(int u, int ftr) {
   // maxx1维护最大值 maxx2维护次大值
   dep[u] = dep[ftr] + 1;
   vis[u] = 1;
   for(int i = 1; i <= 20; i++) {
      lca[u][i] = lca[ lca[u][i - 1] ][i - 1];
      maxx1[u][i] = max( maxx1[u][i - 1], maxx1[ lca[u][i - 1] ][i - 1] );
      if(maxx1[u][i - 1] == maxx1[ lca[u][i - 1] ][i - 1])
         maxx2[u][i] = max( maxx2[u][i - 1], maxx2[ lca[u][i - 1] ][i - 1] );
      else
         maxx2[u][i] = max( min( maxx1[u][i - 1], maxx1[ lca[u][i - 1] ][i - 1] ), max( maxx2[u][i - 1], maxx2[
              lca[u][i - 1] ][i - 1] ) );
   }

   for(auto i : G[u]) {
      int v = i.first, w = i.second;
      if(vis[v]) continue;
      lca[v][0] = u; maxx1[v][0] = w; maxx2[v][0] = 0;
      DFS(v, u);
   }
}

int LCA(int u, int v) {
   if(dep[u] < dep[v]) swap(u, v);
   for(int i = 20; i >= 0; i--) {
      if((1 << i) & (dep[u] - dep[v])) u = lca[u][i];
   }
   for(int i = 20; i >= 0; i--) {
      if(lca[u][i] != lca[v][i]) {
         u = lca[u][i];
         v = lca[v][i];
      }
   }
   return u == v ? u : lca[u][0];
```

```
}

int Work(int u, int l, int w) {
    int max1 = 0, max2 = 0;
    for(int i = 20; i >= 0; i--) {
        if((1 << i) & (dep[u] - dep[l])) {
            max1 = max(max1, maxx1[u][i]);
            max2 = max(max2, maxx2[u][i]);
            u = lca[u][i];
        }
    }
    if(w - max1 == 0) return w - max2;
    return w - max1;
}

signed main()
{
    scanf("%lld%lld", &n, &m);
    for(int i = 1; i <= m; i++) {
        int u, v, w;
        scanf("%lld%lld%lld", &u, &v, &w);
        e[i].u = u, e[i].v = v, e[i].w = w, e[i].flag = 0;
    }
    Kruskal();
    DFS(1, 0);

    int d = 1e14;
    for(int i = 1; i <= m; i++) {
        if(e[i].flag) continue;
        int u = e[i].u, v = e[i].v, w = e[i].w;
        int l = LCA(u, v);
        d = min(d, Work(u, l, w) );
        d = min(d, Work(v, l, w) );
    }
    printf("%lld", ans + d);
    return 0;
}
```

4.7 Dijkstra

```
#include<bits/stdc++.h>
using namespace std;

const int INF = 0x3f3f3f3f;
const int MAXN = (int)1e5 + 5;
const int MAXM = (int)2e5 + 5;
struct Node {
    int u, w;
    bool operator < (const Node &a) const { return w > a.w; }
};
struct Edge {
    int v, w, nxt;
}e[MAXM];
int n, m, s, cnt = 0, head[MAXN], vis[MAXN], dist[MAXN];
priority_queue<Node> Q;

void Add(int u, int v, int w) {
    e[++cnt].v = v;
    e[cnt].w = w;
    e[cnt].nxt = head[u];
    head[u] = cnt;
}

void Dijkstra(int s) {
```

```
    for(int i = 1; i <= n; i++) vis[i] = 0, dist[i] = INF;
    dist[s] = 0;
    Q.push((Node){s, 0});
    while(!Q.empty()) {
        Node now = Q.top();
        int u = now.u, w = now.w;
        Q.pop();
        if(vis[u]) continue;
        vis[u] = 1;
        for(int i = head[u]; i != -1; i = e[i].nxt) {
            if(dist[e[i].v] > dist[u] + e[i].w) {
                dist[e[i].v] = dist[u] + e[i].w;
                Q.push((Node){e[i].v, dist[e[i].v]});
            }
        }
    }
}

int main(){
    scanf("%d%d%d", &n, &m, &s);
    memset(head, -1, sizeof(head));
    for(int i = 1; i <= m; i++) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        Add(u, v, w);
    }
    Dijkstra(s);
    for(int i = 1; i <= n; i++) printf("%d ", dist[i]);
    return 0;
}
```

4.8 SPFA

```
int n; // 总点数
int h[N], w[N], e[N], ne[N], idx; // 邻接表存边
int dist[N]; // 存储每个点到1号点的最短距离
bool st[N]; // 存储每个点是否在队列中

// 求1号点到n号点的最短距离，如果无法到达返回-1
int SPFA() {
    memset(dist, 0x3f, sizeof dist);
    queue<int> q;
    q.push(1);
    dist[1] = 0, st[1] = true;
    while(!q.empty()) {
        auto t = q.front(); q.pop();
        st[t] = false;
        for(int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];
            if(dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                if(!st[j]) q.push(j), st[j] = true;
            }
        }
    }
    if(dist[n] >= 0x3f3f3f3f) return -1;
    return dist[n];
}
```

4.9 SPFA 负环

```
int n; // 总点数
```

```cpp
int h[N], w[N], e[N], ne[N], idx; // 邻接表存储所有边
int dist[N], cnt[N]; // dist[x]存储1号点到x的最短距离，cnt[x]存储1到x的最短路中经过的点数
bool st[N]; // 存储每个点是否在队列中

// 如果存在负环，则返回true，否则返回false。
bool SPFA() {
    // 不需要初始化dist数组
    // 原理：如果某条最短路径上有n个点（除了自己），那么加上自己之后一共有n+1个点，由抽屉原理一定有两个点相同，所以存在环。

    queue<int> q;
    for (int i = 1; i <= n; i++) q.push(i), st[i] = true;

    while(!q.empty()) {
        auto t = q.front(); q.pop();
        st[t] = false;
        for(int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];
            if(dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                cnt[j] = cnt[t] + 1;
                if(cnt[j] >= n) return true; // 如果从1号点到x的最短路中包含至少n个点（不包括自己），则说明存在环
                if(!st[j]) q.push(j), st[j] = true;
            }
        }
    }
    return false;
}
```

4.10 Floyd

```cpp
void Solve1() {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(i == j) dp[i][j] = 0;
            else dp[i][j] = INF;
        }
    }
    for(int k = 1; k <= n; k++) {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++) {
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
            }
        }
    }
}

/*
已知一个有向图中任意两点之间是否有连边，要求判断任意两点是否连通。
Floyd实现传递闭包 bitset优化
*/
void Solve2()
{
    for(int k = 1; k <= n; k++) {
        for(int i = 1; i <= n; i++) {
            if(G[i][k]) G[i] |= G[k];
        }
    }
    for(int k = 1; k <= n; k++) {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++) {
                G[i][j] |= G[i][k] & G[k][j];
            }
        }
    }
```

```
}
```

4.11 差分约束

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 5005;
const int INF = 0x3f3f3f3f;

struct Node { int v, w; };

vector<Node> e[MAXN];
int head[MAXN], vis[MAXN], cnt[MAXN], dis[MAXN], n, m, ecnt;
queue<int> Q;

bool SPFA(int s) {
    for(int i = 0; i <= n; i++) dis[i] = INF, vis[i] = cnt[i] = 0;
    dis[s] = 0, vis[s] = 1; Q.push(s);
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        vis[u] = 0;
        for(auto i : e[u]) {
            int v = i.v, w = i.w;
            if(dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u] + 1;
                if(cnt[v] >= n + 1) return 0;
                if(!vis[v]) Q.push(v), vis[v] = 1;
            }
        }
    }
    return 1;
}

signed main()
{
    scanf("%d%d", &n, &m);
    ecnt = 0;
    //x_u - x_v <= w
    for(int i = 1; i <= m; i++) {
        int u, v, w; scanf("%d%d%d", &u, &v, &w);
        e[u].push_back((Node){v, w});
    }
    for(int i = 1; i <= n; i++) Add(0, i, 0);
    if(!SPFA(0)) printf("NO\n");
    else for(int i = 1; i <= n; i++) printf("%d ", dis[i]);
    return 0;
}
```

4.12 欧拉路径

```
#pragma GCC optimize(2)
#include<bits/stdc++.h>
using namespace std;

const int MAXN = (int)1e5 + 5;
const int MAXM = (int)2e5 + 5;

stack<int> S;
vector<int> G[MAXN];
int del[MAXN], in[MAXN], out[MAXN], n, m;
```

```

void DFS(int u) {
    for(int i = del[u]; i < G[u].size(); i = del[u]) del[u] = i + 1, DFS(G[u][i]);
    S.push(u);
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin >> u >> v;
        G[u].push_back(v); out[u]++; in[v]++;
    }
    for(int i = 1; i <= n; i++) sort(G[i].begin(), G[i].end());
    int s = 1, cnt0 = 0, cnt1 = 0, flag = 1;
    for(int i = 1; i <= n; i++) {
        if(in[i] != out[i]) flag = 0;
        if(out[i] == in[i] + 1) s = i, cnt1++;
        if(in[i] == out[i] + 1) cnt0++;
    }
    if(!flag && !(cnt0 == cnt1 && cnt0 == 1) ) return cout << "No", 0;
    DFS(s);
    while(!S.empty()) cout << S.top() << " ", S.pop();
    return 0;
}
```

4.13 染色法判别二分图

```
//O(n + m)
int n; // n表示点数
int h[N], e[M], ne[M], idx; // 邻接表存储图
int color[N]; // 表示每个点的颜色，-1表示未染色，0表示白色，1表示黑色
// 参数：u表示当前节点，c表示当前点的颜色
bool dfs(int u, int c) {
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (color[j] == -1) {
            if (!dfs(j, !c)) return false;
        } else if (color[j] == c) return false;
    }
    return true;
}
bool check() {
    memset(color, -1, sizeof color);
    bool flag = true;
    for (int i = 1; i <= n; i ++ ) {
        if (color[i] == -1) if (!dfs(i, 0)) {
            flag = false;
            break;
        }
    }
    return flag;
}
```

4.14 匈牙利算法

```
bool Match(int x) {
    for(int i = 1; i <= m; i++) {
        if(G[x][i] && !vis[i]) {
            vis[i] = 1;
```

```
            if(!a[i] || Match(a[i])) {
                a[i] = x;
                return true;
            }
        }
    }
    return false;
}

int main() {
    scanf("%d%d%d", &n, &m, &e);
    for(int i = 1; i <= e; i++) {
        int u, v; scanf("%d%d", &u, &v);
        G[u][v] = 1;
    }
    int ans = 0;
    for(int i = 1; i <= n; i++) {
        memset(vis, 0, sizeof(vis));
        if(Match(i)) ans++;
    }
    printf("%d", ans);
    return 0;
}
```

4.15 Tarjan

```
const int N;

int dfn[N], low[N], s[N], vis[N], color[N], top = 0, sum = 0, dep = 0;
int n, m;
vector<int> G[N];

// 求强连通分量 复杂度O(E+V)
void Tarjan(int u) {
    dfn[u] = low[u] = ++dep;
    vis[u] = 1;
    s[++top] = u;

    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if(!dfn[v]) Tarjan(v), low[u] = min(low[u], low[v]);
        else if(vis[v]) low[u] = min(low[u], low[v]);
    }

    if(dfn[u] == low[u]) {
        color[u] = ++sum;
        vis[u] = 0;
        while(s[top] != u) {
            color[ s[top] ] = sum;
            vis[ s[top] ] = 0;
            top--;
        }
        top--;
    }
}

// 求割点
vector<int> cut; // 存储所有割点
void Tarjan(int u, bool root = true) {
    int tot = 0;
    low[u] = dfn[u] = ++dep;
    for(auto v : G[u]) {
        if(!dfn[v]) {
            Tarjan(v, false);
```

```
            low[u] = min(low[u], low[v]);
            tot += (low[v] >= dfn[u]); // 统计满足low[v] >= dfn[u]的子节点数目
        } else low[u] = min(low[u], dfn[v]);
    }
    if (tot > root) // 如果是根，tot需要大于1；否则只需大于0
        cut.push_back(u);
}

// 求割桥
vector<pair<int, int>> bridges; // 存割桥
void Tarjan(int u) {
    low[u] = dfn[u] = ++dep;
    for(auto v : G[u]) {
        if(!dfn[v]) {
            fa[v] = u; // 记录父节点
            Tarjan(v);
            low[u] = min(low[u], low[v]);
            if(low[v] > dfn[u]) bridges.emplace_back(u, v);
        } else if (fa[u] != v) // 排除父节点
            low[u] = min(low[u], dfn[v]);
    }
}

void Solve() {
    for(int i = 1; i <= n; i++) if(!dfn[i]) {
        Tarjan(i);
        Tarjan(i, true);
    }
}
```

4.16 Dinic 最大流

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

const int MAXN = 205;
const int MAXM = 5005;

struct Edge {
    int v, w, nxt;
}e[MAXM << 1];

int n, m, s, t, cnt = 1, dep[MAXN], head[MAXN];

void Add(int u, int v, int w) {
    e[++cnt].v = v;
    e[cnt].w = w;
    e[cnt].nxt = head[u];
    head[u] = cnt;
}

bool BFS() {
    for(int i = 1; i <= n + 1; i++) dep[i] = 0;
    dep[s] = 1;
    queue<int> Q;
    Q.push(s);
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        for(int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if(dep[v] == 0 && e[i].w > 0) {
                dep[v] = dep[u] + 1;
```

```
            Q.push(v);
        }
    }
}
return (bool)dep[t];
}

int DFS(int u, int in) {
if(u == t) return in;
int out = 0;
for(int i = head[u]; i != -1 && in > 0; i = e[i].nxt) {
    int v = e[i].v;
    if(dep[v] == dep[u] + 1 && e[i].w > 0) {
        int res = DFS(v, min(e[i].w, in));
        e[i].w -= res;
        e[i ^ 1].w += res; // 反向边(残量网络)
        in -= res;
        out += res;
    }
}
if(out == 0) dep[u] = 0;
return out;
}

signed main()
{
memset(head, -1, sizeof(head));
scanf("%lld%lld%lld%lld", &n, &m, &s, &t);
for(int i = 1; i <= m; i++) {
    int u, v, w; scanf("%lld%lld%lld", &u, &v, &w);
    Add(u, v, w); Add(v, u, 0);
}
int ans = 0;
while(BFS()) { ans += DFS(s, 1e18); }
printf("%lld", ans);
return 0;
}
```

4.17 KM

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

const int MAXN = 505;
const int INF = (int)1e16;

int n, m;
int G[MAXN][MAXN];
int lmatch[MAXN], rmatch[MAXN];
int pre[MAXN];
int lexpect[MAXN], rexpect[MAXN];
int lvis[MAXN], rvis[MAXN];
int slack[MAXN];
queue<int> Q;

void aug(int v) {
int temp;
while(v) {
    temp = lmatch[ pre[v] ];
    lmatch[ pre[v] ] = v;
    rmatch[v] = pre[v];
    v = temp;
```

```
    }
}

void BFS(int s) {
    for(int i = 1; i <= n; i++) lvis[i] = rvis[i] = 0, slack[i] = INF;

    while(!Q.empty()) Q.pop();
    Q.push(s);

    while(1) {
        while(!Q.empty()) {
            int u = Q.front(); Q.pop();
            lvis[u] = 1;
            for(int v = 1; v <= n; v++) {
                if(!rvis[v]) {
                    int gap = lexpect[u] + rexpect[v] - G[u][v];
                    if(slack[v] > gap) {
                        slack[v] = gap;
                        pre[v] = u;
                        if(slack[v] == 0) {
                            rvis[v] = 0;
                            if(!rmatch[v]) { aug(v); return ; }
                            else Q.push(rmatch[v]);
                        }
                    }
                }
            }
        }

        int d = INF;
        for(int i = 1; i <= n; i++)
            if(!rvis[i]) d = min(d, slack[i]);

        for(int i = 1; i <= n; i++) {
            if(lvis[i]) lexpect[i] -= d;

            if(rvis[i]) rexpect[i] += d;
            else slack[i] -= d;
        }

        for(int i = 1; i <= n; i++) {
            if(!rvis[i]) {
                if(slack[i] == 0) {
                    rvis[i] = 1;
                    if(!rmatch[i]) { aug(i); return ; }
                    else Q.push(rmatch[i]);
                }
            }
        }
    }
}

int KM() {
    for(int i = 1; i <= n; i++) lmatch[i] = rmatch[i] = lexpect[i] = rexpect[i] = 0;

    for(int i = 1; i <= n; i++) {
        lexpect[i] = G[i][1];
        for(int j = 2; j <= n; j++) lexpect[i] = max(lexpect[i], G[i][j]);
    }

    for(int i = 1; i <= n; i++) BFS(i);

    int res = 0;
    for(int i = 1; i <= n; i++)
        if(rmatch[i]) res += G[rmatch[i]][i];
```

```

    return res;
}

signed main()
{
    scanf("%lld%lld", &n, &m);
    for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) G[i][j] = -INF;
    for(int i = 1; i <= m; i++) {
        int u, v, w; scanf("%lld%lld%lld", &u, &v, &w);
        G[u][v] = w;
    }
    printf("%lld\n", KM());
    for(int i = 1; i <= n; i++) printf("%lld ", rmatch[i]);
    return 0;
}
```

4.18 轻重链剖分

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

const int MAXN = (int)1e5 + 5;

struct Node {
    int v, nxt;
}e[MAXN << 1];

int sum[MAXN << 2], tag[MAXN << 2];
int n, m, r, MOD, cnt, e_cnt, v[MAXN];
int head[MAXN], vis[MAXN], fa[MAXN], sze[MAXN], son[MAXN], dep[MAXN];
int idx[MAXN], a[MAXN], top[MAXN];

int ls(int x) { return x << 1; }
int rs(int x) { return x << 1 | 1; }

void F(int l, int r, int p, int k) {
    tag[p] += k; tag[p] %= MOD;
    sum[p] += (r - l + 1) * k; sum[p] %= MOD;
}

void PushUp(int p) {
    sum[p] = sum[ls(p)] + sum[rs(p)];
    sum[p] %= MOD;
}

void PushDown(int l, int r, int p) {
    int mid = (l + r) >> 1;
    F(l, mid, ls(p), tag[p]);
    F(mid + 1, r, rs(p), tag[p]);
    PushUp(p);
    tag[p] = 0;
}

void Build(int l, int r, int p) {
    if(l == r) {
        sum[p] = a[l];
        tag[p] = 0;
        return ;
    }
    int mid = (l + r) >> 1;
    Build(l, mid, ls(p));
```

```
    Build(mid + 1, r, rs(p));
    PushUp(p);
}

void SegmentTreeModify(int nl, int nr, int l, int r, int p, int k) {
    if(nl <= l && nr >= r) {
        tag[p] += k; tag[p] %= MOD;
        sum[p] += (r - l + 1) * k; sum[p] %= MOD;
        return ;
    }
    PushDown(l, r, p);
    int mid = (l + r) >> 1;
    if(nl <= mid) SegmentTreeModify(nl, nr, l, mid, ls(p), k);
    if(nr > mid) SegmentTreeModify(nl, nr, mid + 1, r, rs(p), k);
    PushUp(p);
    return ;
}

int SegmentTreeQuery(int nl, int nr, int l, int r, int p) {
    if(nl <= l && nr >= r) {
        return sum[p];
    }
    PushDown(l, r, p);
    int res = 0;
    int mid = (l + r) >> 1;
    if(nl <= mid) res += SegmentTreeQuery(nl, nr, l, mid, ls(p)), res %= MOD;
    if(nr > mid) res += SegmentTreeQuery(nl, nr, mid + 1, r, rs(p)), res %= MOD;
    return res;
}

void Add(int u, int v) {
    e[++e_cnt].v = v;
    e[e_cnt].nxt = head[u];
    head[u] = e_cnt;
}

void DFS1(int u, int ftr) {
    fa[u] = ftr;
    sze[u] = 1;
    dep[u] = dep[ftr] + 1;
    vis[u] = 1;
    int maxsize = -1;
    for(int i = head[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if(vis[v]) continue;
        DFS1(v, u);
        sze[u] += sze[v];
        if(sze[v] > maxsize) {
            son[u] = v;
            maxsize = sze[v];
        }
    }
}

void DFS2(int u, int top_u) {
    idx[u] = ++cnt;
    a[cnt] = v[u];
    top[u] = top_u;
    if(son[u] == 0) return ;
    DFS2(son[u], top_u);
    for(int i = head[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if(v == fa[u] || v == son[u]) continue;
        DFS2(v, v);
    }
```

```
}

void Modify(int u, int v, int w) {
    while(top[u] != top[v]) {
        if(dep[top[u]] < dep[top[v]]) swap(u, v);
        SegmentTreeModify(idx[top[u]], idx[u], 1, n, 1, w);
        u = fa[top[u]];
    }
    if(idx[u] > idx[v]) swap(u, v);
    SegmentTreeModify(idx[u], idx[v], 1, n, 1, w);
}

int Query(int u, int v) {
    int res = 0;
    while(top[u] != top[v]) {
        if(dep[top[u]] < dep[top[v]]) swap(u, v);
        res += SegmentTreeQuery(idx[top[u]], idx[u], 1, n, 1);
        res %= MOD;
        u = fa[top[u]];
    }
    if(idx[u] > idx[v]) swap(u, v);
    res += SegmentTreeQuery(idx[u], idx[v], 1, n, 1);
    return res % MOD;
}

signed main()
{
    scanf("%lld%lld%lld%lld", &n, &m, &r, &MOD);
    cnt = 0, e_cnt = 0; memset(head, -1, sizeof(head));
    for(int i = 1; i <= n; i++) scanf("%lld", &v[i]);
    for(int i = 1; i < n; i++) {
        int u, v; scanf("%lld%lld", &u, &v);
        Add(u, v); Add(v, u);
    }
    DFS1(r, 0); DFS2(r, r); Build(1, n, 1);
    while(m--) {
        int opt, x, y, z; scanf("%lld", &opt);
        if(opt == 1) {
            scanf("%lld%lld%lld", &x, &y, &z);
            Modify(x, y, z);
        } else if(opt == 2) {
            scanf("%lld%lld", &x, &y);
            printf("%lld\n", Query(x, y));
        } else if(opt == 3) {
            scanf("%lld%lld", &x, &z);
            SegmentTreeModify(idx[x], idx[x] + sze[x] - 1, 1, n, 1, z);
        } else {
            scanf("%lld", &x);
            printf("%lld\n", SegmentTreeQuery(idx[x], idx[x] + sze[x] - 1, 1, n, 1) % MOD);
        }
    }
    return 0;
}

/*
input:
5 5 2 24
7 3 7 8 0
1 2
1 5
3 1
4 1
3 4 2
3 2 2
4 5
```

```
1 5 1 3
2 1 3

output: 2 21
*/
```

4.19 树上启发式合并

```cpp
// 轻重链剖分
void DFS(int u, int fa) {
   dfn[u] = ++tot, node[tot] = u, sze[u] = 1;
   for(auto v : G[u]) {
      if(v == fa) continue;
      DFS(v, u);
      if(sze[v] > sze[ son[u] ]) son[u] = v;
      sze[u] += sze[v];
   }
}

void add(int pos) { }
void del(int pos) { }
int getAns() { }

void DSU(int u, int fa, bool st) {
   for(auto v : G[u]) if(v != fa && v != son[u]) DSU(v, u, 0);
   if(son[u]) DSU(son[u], u, 1);
   for(auto v : G[u]) {
      if(v == fa || v == son[u]) continue;
      for(int i = dfn[v]; i < dfn[v] + sze[v]; i++) add(node[i]);
   }
   add(u);
   ans[u] = getAns();
   if(!st) for(int i = dfn[u]; i < dfn[u] + sze[u]; i++) del(node[i]);
}

void Solve() {
   DFS(1, 0);
   DSU(1, 0, 0);
}
```

4.20 点分治

```cpp
#include<bits/stdc++.h>
#define pir make_pair
#define pii pair<int, int>
#define fi first
#define se second
using namespace std;
const int MAXN = 1e4 + 5;
const int MAXV = 1e7 + 5;
const int MOD = 1e9 + 7;

int n, m, ans[MAXN], val[MAXN], sze[MAXN], vis[MAXN], centroid;
int Map[MAXN], Cur[MAXN], MapVal[MAXV];
vector<pii> G[MAXN];

void GetCentroid(int u, int fa, int n) {
   sze[u] = 1;
   int maxx = 0;
   for(auto i : G[u]) {
      int v = i.fi, w = i.se;
      if(v == fa || vis[v]) continue;
```

```
        GetCentroid(v, u, n);
        if(centroid != -1) return ;
        maxx = max(maxx, sze[v]);
        sze[u] += sze[v];
    }
    maxx = max(maxx, n - sze[u]);
    if(maxx <= n / 2) centroid = u, sze[fa] = n - sze[u];
}

void Calc(int u, int fa, int len) {
    if(len > 1e7) return ;
    Cur[ ++Cur[0] ] = len;
    for(auto i : G[u]) {
        int v = i.fi, w = i.se;
        if(vis[v] || v == fa) continue;
        Calc(v, u, len + w);
    }
}

void Calc(int u) {
    Map[ Map[0] = 1 ] = 0, MapVal[0] = 1;
    for(auto i : G[u]) {
        int v = i.fi, w = i.se;
        if(vis[v]) continue;
        Cur[0] = 0; Calc(v, u, w);
        for(int j = 1; j <= Cur[0]; j++)
            for(int k = 1; k <= m; k++)
                if(val[k] - Cur[j] >= 0)
                    ans[k] |= MapVal[ val[k] - Cur[j] ];
        for(int j = 1; j <= Cur[0]; j++) MapVal[ Cur[j] ] = 1, Map[ ++Map[0] ] = Cur[j];
    }
    for(int i = 1; i <= Map[0]; i++) MapVal[ Map[i] ] = 0;
}

void Solve(int u) {
    vis[u] = 1; Calc(u);
    for(auto i : G[u]) {
        int v = i.fi, w = i.se;
        if(vis[v]) continue;
        centroid = -1; GetCentroid(v, 0, sze[v]);
        Solve(centroid);
    }
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> n >> m;
    for(int i = 1; i < n; i++) {
        int u, v, w; cin >> u >> v >> w;
        G[u].push_back( pir(v, w) ), G[v].push_back( pir(u, w) );
    }
    for(int i = 1; i <= m; i++) cin >> val[i];
    centroid = -1;
    GetCentroid(1, 0, n);
    Solve(centroid);
    for(int i = 1; i <= m; i++) cout << (ans[i] ? "AYE" : "NAY") << "\n";
    return 0;
}
```

4.21 虚树

```
//#pragma GCC optimize(2)
#include<bits/stdc++.h>
```

```cpp
#define int long long
#define pir make_pair
#define pii pair<int, int>
#define fi first
#define se second
using namespace std;
const int MAXN = 1e6 + 5;
const int INF = 0x3f3f3f3f;

int n, m, cnt, dfn[MAXN], dep[MAXN], top, s[MAXN], vis[MAXN], lca[MAXN][25];
vector<pii> VG[MAXN], vec;
vector<int> G[MAXN];
unordered_map<int, int> Map;

void DFS(int u, int fa) {
    dfn[u] = ++cnt; dep[u] = dep[fa] + 1;
    lca[u][0] = fa;
    for(int i = 1; i <= 20; i++) lca[u][i] = lca[ lca[u][i - 1] ][i - 1];
    for(auto i : G[u]) if(i != fa) DFS(i, u);
}

int LCA(int u, int v) {
    if(dep[u] > dep[v]) swap(u, v);
    for(int i = 20; i >= 0; i--) if((1 << i) & (dep[v] - dep[u])) v = lca[v][i];
    for(int i = 20; i >= 0; i--) if(lca[u][i] != lca[v][i]) u = lca[u][i], v = lca[v][i];
    return u == v ? u : lca[u][0];
}

int Dist(int u, int v) {

}

void Add(int u, int v) {
    int w = Dist(u, v);
    VG[u].push_back( pir(v, w) );
    VG[v].push_back( pir(u, w) );
    if(!Map[u]) Map[u] = ++cnt;
    if(!Map[v]) Map[v] = ++cnt;
}

void VirtualTreeInsert(int u) {
    if(!top) return void(s[++top] = u);
    int l = LCA(s[top], u);
    while(top > 1 && dep[l] < dep[ s[top - 1] ]) Add(s[top - 1], s[top]), --top;
    if(dep[l] < dep[ s[top] ]) Add(l, s[top]), --top;
    if(!top || s[top] != l) s[++top] = l;
    if(s[top] != u) s[++top] = u;
}

void BuildVirtualTree(vector<pii> vec, int k) {
    sort(vec.begin(), vec.end());
    for(auto i : Map) VG[i.fi].clear();
    top = cnt = 0; Map.clear();
    s[++top] = 1, Map[1] = ++cnt;
    for(auto i : vec) VirtualTreeInsert(i.se);
    for(int i = 1; i < top; i++) Add(s[i], s[i + 1]);
}

signed main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        G[u].push_back(v), G[v].push_back(u);
```

```
    }
    DFS(1, 0); cnt = 0;
    cin >> m;
    while(m--) {
        int k; cin >> k; vec.clear();
        for(int i = 1; i <= k; i++) {
            int cur; cin >> cur;
            vec.push_back( pir(dfn[cur], cur) );
        }
        BuildVirtualTree(vec, k);
    }
    return 0;
}
```

4.22 树哈希

```
// 复杂度O(nlogn), n为节点数量
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1e6 + 5;

vector<int> g[MAXN];
unordered_map<int, int> num;
map<vector<int>, int> Hash;
int hash_cnt;

int dfs(int u, int f) {
    vector<int> vec;
    for(auto &v : g[u]) {
        if(v == f) continue;
        vec.push_back( dfs(v, u) );
    }
    sort(vec.begin(), vec.end() );
    if(Hash[vec] == 0) Hash[vec] = ++hash_cnt;
    return Hash[vec];
}

void init() {
    hash_cnt = 0;
    Hash.clear();
}

signed main() {
    //ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    init();
    int n; cin >> n;
    for(int j = 1; j <= n; j++) {
        int f; cin >> f;
        if(f == 0) continue;
        g[f].push_back(j); g[j].push_back(f);
    }

    /*
    树哈希返回对应子树的哈希值,
    如需要比较两棵树的哈希值, 可以通过求重心的方式固定根节点;
    若重心有两个, 分别固定根节点求哈希值即可。
    */
    int h = dfs(centroid, 0);
    if(num[h] == 0) num[h] = i;
    cout << num[h] << "\n";

    for(int j = 0; j <= n; j++) g[j].clear();
}
```

5 数学

5.1 快速幂

```
// 快速乘 适用正数
int qmul(int a, int b, int MOD) {
    int ans = 0;
    while(b > 0) {
        if(b & 1) ans = (ans + a) % MOD;
        b >>= 1;
        a = (a << 1) % MOD;
    }
    return ans;
}

// 快速幂
int qpow(int a, int p, int MOD) {
    int ans = 1;
    while(p > 0) {
        if(p & 1) ans = ans * a % MOD;
        p >>= 1;
        a = a * a % MOD;
    }
    return ans;
}

// 矩阵快速幂
struct Martix {
    long long c[MAXN][MAXN];
}A, E;

Martix operator * (const Martix &a, const Martix &b) {
    Martix Ans;
    for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) Ans.c[i][j] = 0;
    for(int k = 1; k <= n; k++) {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++) Ans.c[i][j] = (Ans.c[i][j] + a.c[i][k] * b.c[k][j] % MOD) % MOD;
        }
    }
    return Ans;
}

Martix MartixQuickPow(Martix A, int k) {
    Martix ans, now;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) ans.c[i][j] = 0;
        ans.c[i][i] = 1;
    }
    for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) now.c[i][j] = A.c[i][j];
    while(k > 0) {
        if(1 & k) ans = ans * now;
        now = now * now;
        k = k >> 1;
    }
    return ans;
}
```

5.2 欧拉函数

```
int phi(int n) {
    int res = n;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) res = res / i * (i - 1);
        while(n % i == 0) n /= i;
```

```
    }
    if(n > 1) res = res / n * (n - 1);
    return res;
}

int phi[MAXN];
void init(int n) {
    // 复杂度 O(nloglogn)
    for(int i = 1; i <= n; i++) phi[i] = i; // 除1外没有数的欧拉函数是本身，所以如果phi[i] = i则说明未被筛到
    for(int i = 2; i <= n; i++) {
        if(phi[i] == i) { // 未被筛到
            for(int j = i; j <= n; j += i) phi[j] = phi[j] / i * (i - 1); // 所有含有该因子的数都进行一次操作
        }
    }

    // 复杂度 O(n)
    phi[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!isnp[i]) primes.push_back(i), phi[i] = i - 1;
        for(int p : primes) {
            if(p * i > n) break;
            isnp[p * i] = 1;
            if(i % p == 0) {
                phi[p * i] = phi[i] * p;
                break;
            } else {
                phi[p * i] = phi[p] * phi[i];
            }
        }
    }
}

int EulerPow(int a, string b, int MOD) {
    int phiMOD = phi(MOD), power = 0, flag = 0;
    for(auto i : b) {
        power = power * 10 + i - '0';
        if(power >= phiMOD) flag = 1;
        power %= phiMOD;
    }
    if(flag) power += phiMOD;
    return qpow(a, power, MOD);
}
```

5.3 线性筛

```
#include<bits/stdc++.h>

using namespace std;

const int MAXN = 2e7 + 5;

int n, m, isprime[MAXN], prime[MAXN], cnt;

void Init(int n = 2e7) {
    cnt = 0;
    isprime[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!isprime[i]) prime[++cnt] = i;
        for(int j = 1; j <= cnt && i * prime[j] <= n; j++) {
            isprime[i * prime[j]] = 1;
            if(i % prime[j] == 0) break;
        }
    }
    //for(int i = 1; i <= cnt; i++) cout << prime[i] << " "; cout << endl;
```

```
}

int main()
{
    scanf("%d", &n);
    Init(n);
    return 0;
}
```

5.4 整除分块

```
for(int l = 1, r; l <= n; l = r + 1) r = n / (n / l); // [l, r]为当前整除分块 区间内每个 n / i 相同
```

5.5 组合数处理

```
int inv(int x, int MOD) { // 求逆元
    return qpow(x, MOD - 2, MOD);
}

void Init() {
    fact[0] = 1;
    for(int i = 1; i <= n; i++) fact[i] = fact[i - 1] * i % MOD;

    inv[n] = inv(fact[n], MOD);
    for(int i = n - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % MOD;
}

int C(int n, int m) {
    if(m > n) return 0;
    return fact[n] * inv[m] % MOD * inv[n - m] % MOD;
}

int Lucas(int n, int m) {
    if(m == 0) return 1;
    return C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) % MOD;
}
```

6 其他算法

6.1 二分 & 三分

```
const double eps = 1e-10;
double l, r, a[MAXN];
int n;
double F(double x) {}

void Solve() {
    // 单调递增序列a中查找 >=x 的数中最小的一个
    while(l < r) {
        int mid = l + r >> 1;
        if(a[mid] >= x) r = mid;
        else l = mid + 1;
    }

    // 单调递增序列a中查找 <=x 的数中最大的一个
    while(l < r) {
        int mid = l + r + 1 >> 1;
        if(a[mid] <= x) l = mid;
        else r = mid - 1;
    }
```

```
}

void Solve() {
    // 浮点数三分 求单峰函数的极大值
    while(r - l > eps) {
        double lmid = 1.0 * l + 1.0 * (r - l) / 3.0;
        double rmid = 1.0 * r - 1.0 * (r - l) / 3.0;
        if(F(lmid) > F(rmid)) r = rmid;
        else l = lmid;
    }

    // 整数三分 注意特判边界等各种情况 较毒瘤
    while(l < r) {
        int lmid = l + (r - l) / 3;
        int rmid = r - (r - l) / 3;
        if(F(lmid) > F(rmid)) r = rmid;
        else l = lmid;
    }
}
```

6.2 前缀和 & 差分

```
// 二维前缀和 & 二维差分
sum[i][j] = a[i][j] + sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1];
d[i][j] = a[i][j] - a[i - 1][j] - a[i][j - 1] + a[i - 1][j - 1];

int QuerySum() {
    // 以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：
    return sum[x2][y2] - sum[x1 - 1][y2] - sum[x2][y1 - 1] + sum[x1 - 1][y1 - 1];
}
void ModifyD() {
    // 给以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵中的所有元素加上c：
    d[x1][y1] += c, d[x2 + 1][y1] -= c, d[x1][y2 + 1] -= c, d[x2 + 1][y2 + 1] += c;
}

/*
树上差分
注意对差分数组进行前缀和操作，得到原数组时顺序是 自叶子节点开始 到 根结束 !
*/
void SolveNode(int u, int v, int t) { //u--v之间的点全部加上c
    val[u] += t, val[v] += t;
    int lca = LCA(u, v);
    val[lca] -= t, val[fa[lca]] -= t;
}

void SolveEdge(int u, int v, int t) { //u--v之间的边全部加上c
    val[u] += t, val[v] += t;
    val[LCA(u, v)] -= 2 * t;
}
```

6.3 离散化

```
// a[i] 为初始数组,下标范围为 [1, n]
// len 为离散化后数组的有效长度
std::sort(a + 1, a + 1 + n);
len = std::unique(a + 1, a + n + 1) - a - 1;
// 离散化整个数组的同时求出离散化后本质不同数的个数。
std::lower_bound(a + 1, a + len + 1, x) - a; // 查询 x 离散化后对应的编号
```

6.4 排序

```cpp
void QuickSort(int l, int r) { //快排
    int mid = a[(l + r) / 2];
    int i = l, j = r;
    do {
        while(a[i] < mid) i++;
        while(a[j] > mid) j--;
        if(i <= j) swap(a[i], a[j]), i++, j--;
    } while(i <= j);
    if(j > l) QuickSort(l, j);
    if(i < r) QuickSort(i, r);
}

void MergeSort(int l, int r) { //归并排序
    if (l >= r) return;
    int mid = (l + r) >> 1;
    MergeSort(l, mid);
    MergeSort(mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while(i <= mid && j <= r) {
        if(a[i] <= a[j]) tmp[k++] = a[i++];
        else tmp[k++] = a[j++];
    }
    while(i <= mid) tmp[k++] = a[i++];
    while(j <= r) tmp[k++] = a[j++];
    for(i = l, j = 0; i <= r; i++, j++) a[i] = tmp[j];
}
```

6.5 高精度运算

```cpp
// C = A + B, A >= 0, B >= 0
vector<int> add(vector<int> &A, vector<int> &B) {
    if (A.size() < B.size()) return add(B, A);
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i ++ ) {
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }
    if (t) C.push_back(t);
    return C;
}

// C = A - B, TьзуA >= B, A >= 0, B >= 0
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); i ++ ) {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

// C = A * b, A >= 0, b >= 0
vector<int> mul(vector<int> &A, int b) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size() || t; i ++ ) {
```

```
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

// A / b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r) {
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i -- ) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

6.6 CDQ 分治

```
#pragma GCC optimize(2)
#define IOS ios::sync_with_stdio(false); cin.tie(0);
#include<bits/stdc++.h>
#define int long long
using namespace std;

const int MAXN = (int)1e5 + 5;

struct Node {
    int val, del, ans;
}a[MAXN];

int n, m, ans, c[MAXN], pos[MAXN];

bool cmp1(const Node &a, const Node &b) { return a.val < b.val; }
bool cmp2(const Node &a, const Node &b) { return a.del < b.del; }

void Modify(int x, int k) {
    while(x <= n) {
        c[x] += k;
        x += (x & (-x));
    }
}

int Query(int x) {
    int res = 0;
    while(x > 0) {
        res += c[x];
        x -= (x & (-x));
    }
    return res;
}

void Solve(int l, int r) {
    if(l == r) return ;
    int mid = l + r >> 1;
    Solve(l, mid), Solve(mid + 1, r);

    int i = l, j = mid + 1;
    while(i <= mid) {
```

```
        while(a[i].val > a[j].val && j <= r) Modify(a[j].del, 1), ++j;
        a[i].ans += Query(m + 1) - Query(a[i].del), ++i;
    }
    i = l, j = mid + 1;
    while(i <= mid) {
        while(a[i].val > a[j].val && j <= r) Modify(a[j].del, -1), ++j;
        ++i;
    }

    i = mid, j = r;
    while(j > mid) {
        while(a[j].val < a[i].val && i >= l) Modify(a[i].del, 1), --i;
        a[j].ans += Query(m + 1) - Query(a[j].del), --j;
    }
    i = mid, j = r;
    while(j > mid) {
        while(a[j].val < a[i].val && i >= l) Modify(a[i].del, -1), --i;
        --j;
    }
    sort(a + l, a + r + 1, cmp1);
}

signed main()
{
    IOS
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i].val, pos[ a[i].val ] = i;
    for(int i = 1; i <= m; i++) {
        int cur; cin >> cur;
        a[ pos[cur] ].del = i;
    }
    for(int i = 1; i <= n; i++) if(!a[i].del) a[i].del = m + 1;
    for(int i = 1; i <= n; i++) {
        ans += Query(n) - Query(a[i].val);
        Modify(a[i].val, 1);
    }
    for(int i = 1; i <= n; i++) Modify(a[i].val, -1);
    Solve(1, n);
    sort(a + 1, a + n + 1, cmp2);
    for(int i = 1; i <= m; i++) {
        cout << ans << "\n";
        ans -= a[i].ans;
    }
    return 0;
}
```

7 杂项

7.1 STL

```
// 万能（误）算法头文件（部分）
#include <algorithm>
using namespace std;
int main() {
    iterator begin, end; // 指代某种数据结构首尾迭代器
    T i, x, a, b;

    sort(begin, end, <cmp>); // 排序函数，默认从小到大
    // 遇到需要特殊排序的需要编写cmp函数 or 重载内部运算符
    next_permutation(begin, end); // 下一个排列
    prev_permutation(begin, end); // 前一个排列

    set_union(begin(a), end(a), begin(b), end(b), begin(c));
    // 取两个有序序列a、b的并集，存放到c中
```

```
    set_intersection(begin(a), end(a), begin(b), end(b), begin(c));
    // 取两个有序序列a、b的交集，存放到c中
    set_difference(begin(a), end(a), begin(b), end(b), begin(c));
    // 取两个有序序列a、b的差集，存放到c中
    unique(begin, end); // 有序数据去重
    merge(begin(a), end(a), begin(b), end(b), begin(c), cmp);
    // 合并两个有序序列a、b，存放到c中，cmp可定义新序列排列方式

    lower_bound(begin, end, x); // 返回x的前驱迭代器
    // 在普通的升序序列中，x的前驱指的是第一个大于等于x的值
    upper_bound(begin, end, x); // 返回x的后继迭代器
    // 在普通的升序序列中，x的后继指的是第一个大于x的值
    // 上述两个函数时间复杂度为O(log2(n))，内部实现是二分
    // 如果找不到这样的值，会返回end

    find(begin, end, x); // O(n)查找x
    binary_search(begin, end, x) // 二分查找x，返回bool
    min(a, b); max(a, b); // 返回a、b中的最小/最大值
    fill(begin, end, x); // 往容器的[begin, end)内填充x
    swap(a, b); // 交换a、b的值

    return 0;
}

// 动态数组（vector）、双向链表（list）
#include <vector>
#include <list>
using namespace std;
int main() {
    T i;
    unsigned int n, x;
    bool flag;
    iterator it;

    // 动态数组部分
    // 注意vector的空间需要预留两倍大小
    vector<T> v;
    v.push_back(i); // 往数组尾添加一个元素i
    v[x]; // 访问第x - 1个元素
    v.begin(); // 返回头元素的迭代器
    v.end(); // 返回末尾迭代器（尾元素的下一个）
    n = v.size(); // 数组中元素数量
    v.pop_back(); // 删除最后一个元素
    v.erase(it); // 删除某个的元素
    v.insert(x, i); // 在x位置插入元素i
    // erase、insert时间复杂度为O(n)
    v.clear(); // 清空数组，不释放空间
    flag = v.empty(); // 判断数组是否为空（真值）

    // 链表部分
    list<T> li;
    li.push_front(i); // 在链头添加一个元素i
    li.push_back(i); // 在链尾添加一个元素i
    li.pop_front(i); // 删除链表头元素
    li.pop_back(i); // 删除链表尾元素
    li.erase(it); // 删除某个的元素
    li.insert(x, i); // 在x位置插入元素i O(n)
    li.begin(); // 返回头元素的迭代器
    li.end(); // 返回末尾迭代器（尾元素的下一个）
    n = li.size(); // 链表中元素数量
    li.remove(i); // 删除链表中所有值为i的元素
    li.unique(); // 移除所有连续相同元素，留下一个
    li.reverse(); // 反转链表
    li.clear(); // 情况链表，不释放空间

```

```cpp
    return 0;
}

// 普通队列、双端队列、优先队列
#include <queue> // 队列头文件
#include <deque> // 双端队列头文件
using namespace std;
int main() {
    T i;
    unsigned int n, x;
    bool flag;

    // 普通队列部分，注意queue没有迭代器
    queue<T> q, tmp_q; // 定义普通队列
    q.push(i); // 队尾插入元素i
    q.pop(); // 弹出队首元素
    i = q.front(); // 访问队首元素
    i = q.back(); // 访问队尾元素
    n = q,size(); // 队内元素数量
    flag = q.empty(); // 判断队列是否为空（真值）
    q.swap(tmp_q); // 交换两个队列元素

    // 优先队列部分，注意其没有迭代器
    priority_queue<T> pq; // 定义优先队列
    pq.push(i); // 队尾插入元素i
    pq.pop(); // 弹出队首元素
    i = pq.top(); // 访问队首元素
    n = q,size(); // 队内元素数量
    flag = q.empty(); // 判断队列是否为空（真值）
    q.swap(tmp_q); // 交换两个队列元素
    // 注意优先队列内部是使用<运算符，默认大根堆
    // 可以采用重载运算符或加入运算符类自定义排列方式
    // 例：priority_queue<T, vector<T>, greater<T> > 小根堆
    /*
        struct node {
            int x, y;
        };
        bool operator < (node a, node b) {
            // 这里注意是<右边的元素会放在前面
            if(a.x != b.x) return a.x < b.x;
            else return a.y < b.y;
        }
        priority_queue<node>
    */

    // 双端队列部分
    // 注意deque用到了map来映射，时间复杂度上常数略大
    deque<T> dq; // 定义双端队列
    // 可以称为vector、list、queue的结合体
    // 用法类似，这里只给代码不做注释
    dq.push_back(i);
    dq.push_front(i);
    dq.front();
    dq.back();
    dq.pop_front();
    dq.pop_back();
    dq.begin();
    dq.end();
    dq[x];
    n = dq.size();
    flag = dq.empty();
    dq.insert(x, i);

    return 0;
}
```

```cpp

// 栈
#include <stack>
using namespace std;
int main() {
    T i;
    unsigned int n;
    bool flag;

    stack<T> st; // 注意stack没有迭代器
    st.push(i); // 往栈顶加入一个元素
    st.pop(); // 弹出栈顶元素
    i = st.top(); // 获得栈顶元素的值
    flag = st.empty(); // 判断是否为空（真值）
    n = st.size(); // 获得栈内元素个数

    return 0;
}

// pair（成组）、set（有序元素序列）
#include <set>
#include <pair>
using namespace std;
int main() {
    T i;
    T1 t1;
    T2 t2;
    iterator it;
    unsigned int n;
    bool flag;

    // pair是将两种元素组成一对
    pair<T1, T2> p;
    p = make_pair(t1, t2); // 将t1、t2构造成一对
    // pair支持比较，遵循字典序
    p.first; // 访问第一个元素，这里是t1
    p.second; // 访问第二个元素，这里是t2

    // set内部是RB-tree维护
    set<int> st; // 注意，set内元素不重复
    st.insert(i); // 往set内插入一个元素i
        // 时间复杂度O(log2(n)) 这里会返回一个<pair>迭代器
        // first指向插入元素后所在的迭代器
        // second指向是否插入成功（真值）
    st.begin(); // 返回首迭代器
    st.end(); // 范围尾迭代器
    st.erase(it); st.erase(i);
        // 删除某个元素
    st.equal_range(i); // 返回几何中与i相等的上下限两个迭代器
    flag = st.empty();
    n = st.size();
    st.clear();
    // set内置了lower_bound和upeer_bound函数
    // 用法和algorithm的一样

    // 可重复元素set
    multiset<int> mst;
    // 用法与set大致相同
    // 唯一不同只在删除函数上
    mst.erase(i); // 会删除所有值为i的元素

    return 0;
}

// 如果需要给set自定义排序顺序
```

```cpp
struct CMP {
    bool operator() (const int& a, const int& b) const {
        return a > b; // 返回真值则代表左边的值优先级高
    }
};
multiset<int, CMP> mst;

// map（映射）
#include <map>
using namespace std;
int main() {
    T1 t1;
    T2 t2;

    // map将两种元素做映射，一种指向另一种
    // 内部也是RB-tree维护
    map<T1, T2> mp;
    mp[t1] = t2; // 直接让t1对应到t2
    mp[t1]; // 访问t1对应的内容，时间复杂度O(log2(n))
    // 如果t1没有指向任何内容，则会返回T2类型的初始值

    return 0;
}

// 一些C++的功能/特性
#include <bits/stdc++.h> // 标准库头文件
using namespace std;
int main() {

    __int128 a; // 128位整数，最大值大概10^38次方
    // C++11以上可用，无法用标准方法读入

    cin.tie(0); cout.tie(0);
    ios::sync_with_stdio(false);
    // 关闭cin、cout同步流，此举后不可混用scanf/printf

    auto x; // 自动变量，可以是任意属性
    // 举个例子
    std::set<int> st;
    std::for(auto i:st); // C++版for_each
    // 其中i是auto变量，也可改成set<int>::iterator

    // 所有的STL容器push/insert操作都可替换为emplcace
    // 速度上减小常数（不用临时变量）
    // 例：
    int i;
    std::set<int> st;
    st.emplace(i);
    std::vector<int> vc;
    vc.emplace_back(i);

    return 0;
}

// 强大的pb_ds库
#include <bits/stdc++.h>
#include <bits/extc++.h> // 扩展库头文件
/*
 * 这里如果没有bits/extc++.h的话需要
 * ext/pb_ds/tree_policy.hpp
 * ext/pb_ds/assoc_container.hpp
 * ext/pb_ds/priority_queue_policy.hpp
 * ext/pb_ds/trie_policy.hpp
 * ext/rope
 * ...
```

```
 */
using namespace __gnu_pbds;
using namespace __gnu_cxx;
int main() {

    // 哈希表部分，用法与map一样，效率在C++11以下效率高
    // 注意，这部分在namespace __gnu_pbds下
    cc_hash_table<string, int> mp1; // 拉链法
    gp_hash_table<string, int> mp2; // 查探法(快一些)

    // 优先队列部分，比STL中高级
    priority_queue<int, std::greater<int>, TAG> pq;
    /*
     * 第一个参数是数据类型
     * 第二个是排序方式
     * 第三个是堆的类型
     * 其中堆的类型有下面几种
     * pairing_heap_tag
     * thin_heap_tag
     * binomial_heap_tag
     * c_binomial_heap_tag
     * binary_heap_tag
     * 其中pairing_heap_tag最快
     * 并且这个东西是带默认参数的，只需要定义一个int
     */
    // 比STL中的优先队列多了join和迭代器
    // 例子:
    priority_queue<int> pq1, pq2;
    pq1.join(pq2); // 会将pq2合并到pq1上
    pq1.begin(); pq1.end(); // 可遍历

    // 红黑树（平衡树）部分，与set相似，但更快
    tree <
        int,
        null_type,
        std::less<>,
        rb_tree_tag,
        tree_order_statistics_node_update
    > t, tre;
    /*
     * int 关键字类型
     * null_type 无映射（低版本g++为null_mapped_type）
     * less<int> 从小到大排序
     * rb_tree_tag 红黑树（splay_tree_tag splay）
     * tree_order_statistics_node_update 结点更新
     */
    int i, k;
    t.insert(i); // 插入
    t.erase(i); // 删除
    t.order_of_key(i);
    // 询问这个tree中有多少个比i小的元素
    t.find_by_order(k);
    // 找第k + 1小的元素的迭代器,如果order太大会返回end()
    t.join(tre); // tre合并到t上
    t.split(i, tre); // 小于等于i的保留，其余的属于tre
    // 基本操作有size()/empty()/begin()/end()等
    // 同样内置lower_bound/upper_bound

    // 可持久化平衡树部分
    // 注意，这部分在namespace __gun_cxx下
    rope<char> str;
    // 待我学习完后再更新

    return 0;
}
```

7.2 优化

```cpp
// 关闭iostream同步流
std::ios::sync_with_stdio(false); std::cin.tie(0);
// 如果编译开启了 C++11 或更高版本，建议使用 std::cin.tie(nullptr);
// 注意，此后不可和scanf/printf混用

// 普通快读快写
inline void read_int(int &X) {
    X = 0; int w = 0; char ch = 0;
    while(!isdigit(ch)) w |= ch=='-', ch = getchar();
    while( isdigit(ch)) X = (X<<3)+ (X<<1) + (ch-48), ch = getchar();
    X = w ? -X : X;
}

inline void write_int(int x) {
    static int sta[65];
    int top = 0;
    do {
        sta[top++] = x % 10, x /= 10;
    } while(x);
    while(top) putchar(sta[--top] + 48);
}

// fread快读
namespace fastIO {
#define BUF_SIZE 100000
    //fread -> read
    bool IOerror = 0;
    inline char nc() {
        static char buf[BUF_SIZE], *p1 = buf + BUF_SIZE, *pend = buf + BUF_SIZE;
        if (p1 == pend) {
            p1 = buf;
            pend = buf + fread(buf, 1, BUF_SIZE, stdin);
            if (pend == p1) {
                IOerror = 1;
                return -1;
            }
        }
        return *p1++;
    }
    inline bool blank(char ch) {
        return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
    }
    inline void read(int &x) {
        char ch;
        while (blank(ch = nc()));
        if (IOerror) return;
        for (x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0');
    }
#undef BUF_SIZE
};
using namespace fastIO;

// 手动扩栈
#pragma comment(linker, "/STACK:1024000000,1024000000")

// O2 O3优化
#pragma GCC optimize(2)
#pragma GCC optimize(3)
```

7.3 对拍

```
@echo off
```

```
:loop
随机数据生成.exe
暴力.exe
正解.exe
fc 暴力.out 正解.out
if not errorlevel 1 goto loop
pause
:end
```

7.4 Java 大整数类

```java
// Java大整数
import java.util.*;
import java.math.*;

public class BigInt {
    static Scanner in = new Scanner(System.in); // 定义输入对象
    public static void main(String[] args) {
        BigInteger bigInt_1 = new BigInteger("100");
        BigInteger bigInt_2 = BigInteger.valueOf(123);
        //两种定义方式，建议使用第一种
        bigInt_1.add(bigInt_2); // 加法
        bigInt_1.subtract(bigInt_2); // 减法
        bigInt_1.multiply(bigInt_2); // 乘法
        bigInt_1.divide(bigInt_2);// 除法，向下取整
        bigInt_1.divideAndRemainder(bigInt_2);
            // 返回一个BigInteger[]，包含商和余数
        bigInt_1.remainder(bigInt_2); // 取余数，与this同符号
        bigInt_1.mod(bigInt_2); // 取模，模数只能为正整数
        bigInt_1.pow(10); // a^b
        bigInt_1.gcd(bigInt_2); // 最大公约数
        bigInt_1.compareTo(bigInt_2);
            // 比较大小，<0表示this小，0表示相等，>0表示this大
        bigInt_1.equals(bigInt_2); // 真值为相等
        bigInt_1.negate(); // -a
        bigInt_1.abs(); // 绝对值
        bigInt_1.min(bigInt_2); // 最小值
        bigInt_1.max(bigInt_2); // 最大值
        BigInteger a = in.nextBigInteger(); // 读入
        System.out.println(bigInt_1); // 输出
    }
}

// Java大实数
import java.util.*;
import java.math.*;

public class Big {
    static Scanner in = new Scanner(System.in); // 定义输入对象
    public static void main(String[] args) {
        BigDecimal bigDec_1 = new BigDecimal("123.1");
        BigDecimal bigDec_2 = BigDecimal.valueOf(233.213);
        // 唯有除法与BigInteger不同，无限小数需要规定保留位数
        int scale = 3; // 保留位数
        bigDec_1.divide(bigDec_2, scale, RoundingMode.HALF_UP);
        /*
            第三个参数为保留方式，有以下几种：
            CEILING 正无穷大方向取整
            FLOOR 负无穷大方向取整
            DOWN 向 0 的方向取整
            UP 正数向正无穷大取整，负数向负无穷大取整
            HALF_UP 常用的4舍5入
            HALF_DOWN 6,7,8,9 向上取整
            HALF_EVEN 小数位是5时，判断整数部分是奇数就进位，6,7,8,9 进位
```

```
54          */
55      }
56  }
```