算法模板

2021-05-12T00:00:00.000Z

Contents

1 图论

1.1 网络流

1.1.1 EK 算法

复杂度 $O(nm^2)$

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>

using namespace std;

const int maxn = 1010;
const int maxm = 20000;
const int INF = 0x3f3f3f3f;

int from[maxm], to[maxm], cap[maxm];
int fir[maxn], nxt[maxm];
int tot;

void add(int u, int v, int c) {
    tot++;
    from[tot] = u;
    to[tot] = v;
    cap[tot] = c;

    nxt[tot] = fir[u];
    fir[u] = tot;
}

int n, m, s, t;

int pre[maxn], flow[maxn];
bool vis[maxn];
int q[maxn], l, r;

bool bfs() {
    memset(vis, false, sizeof vis);

    flow[s] = INF;
    vis[s] = true;

    l = r = 0;
    q[++r] = s;

    while (l < r) {
        int u = q[++l];
        for (int e = fir[u]; e; e = nxt[e])
            if (cap[e] > 0) {
                int v = to[e];
                if (!vis[v]) {
                    flow[v] = min(flow[u], cap[e]);
                    q[++r] = v;
                    vis[v] = true;
```

```
                    pre[v] = e;

                    if (v == t) return true;
                }
            }
    }
    return false;
}

int EK() {
    int res = 0;

    while (bfs()) {
        int k = flow[t];
        res += k;

        for (int u = t; u != s; u = from[pre[u]]) {
            cap[pre[u]] -= k;
            cap[pre[u] ^ 1] += k;
        }
    }
    return res;
}

int main() {

    scanf("%d%d%d%d", &n, &m, &s, &t);

    int u, v, w;
    tot = 1;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w);
        add(v, u, 0);
    }

    printf("%d\n", EK());

    return 0;
}
```

1.1.2 Dinic 算法

复杂度 $O(n^2m)$ (笑话)

```
#include <algorithm>
#include <cstdio>
#include <cstring>

using namespace std;

const int maxn = 1010;
const int maxm = 20000;
const int INF = 0x3f3f3f3f;

int from[maxm], to[maxm], cap[maxm];
```

```
int fir[maxn], nxt[maxm];
int tot;

void add(int u, int v, int c) {
    tot++;
    from[tot] = u;
    to[tot] = v;
    cap[tot] = c;

    nxt[tot] = fir[u];
    fir[u] = tot;
}

int n, m, s, t;

int pre[maxn], flow[maxn];
bool vis[maxn];
int q[maxn], l, r;

bool bfs() {
    memset(vis, false, sizeof vis);

    flow[s] = INF;
    vis[s] = true;

    l = r = 0;
    q[++r] = s;

    while (l < r) {
        int u = q[++l];
        for (int e = fir[u]; e; e = nxt[e])
            if (cap[e] > 0) {
                int v = to[e];
                if (!vis[v]) {
                    flow[v] = min(flow[u], cap[e]);
                    q[++r] = v;
                    vis[v] = true;
                    pre[v] = e;

                    if (v == t) return true;
                }
            }
    }
    return false;
}

int EK() {
    int res = 0;

    while (bfs()) {
        int k = flow[t];
        res += k;

        for (int u = t; u != s; u = from[pre[u]]) {
            cap[pre[u]] -= k;
```

```
            cap[pre[u] ^ 1] += k;
        }
    }
    return res;
}

int main() {

    scanf("%d%d%d%d", &n, &m, &s, &t);

    int u, v, w;
    tot = 1;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w);
        add(v, u, 0);
    }

    printf("%d\n", EK());

    return 0;
}
```

1.1.3 最小费用最大流

将 EK 算法中的 bfs 改成 spfa 即可

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

using namespace std;

const int maxn = 500;
const int maxm = 30000 + 100;
const int INF = 0x3f3f3f3f;

int from[maxm], to[maxm], cap[maxm], len[maxm];
int fir[maxn], nxt[maxm], tot;

void add(int a, int b, int c, int l) {
    tot++;
    from[tot] = a, to[tot] = b, cap[tot] = c, len[tot] = l;
    nxt[tot] = fir[a], fir[a] = tot;

    tot++;
    from[tot] = b, to[tot] = a, cap[tot] = 0, len[tot] = -l;
    nxt[tot] = fir[b], fir[b] = tot;
}
int n, m, s, t;

long long maxflow, cost;

int d[maxn], pre[maxn];
bool inq[maxn];
bool spfa() {
    memset(d, 0x3f, sizeof(d));
    memset(inq, 0, sizeof(inq));
```

```
    d[s] = 0;
    queue<int> Q;
    Q.push(s);
    inq[s] = true;

    while (Q.size()) {
        int u = Q.front();
        Q.pop();
        inq[u] = false;

        for (int e = fir[u]; e; e = nxt[e])
            if (cap[e]) {
                int v = to[e];
                if (d[v] > d[u] + len[e]) {
                    d[v] = d[u] + len[e];
                    pre[v] = e;

                    if (!inq[v]) {
                        Q.push(v);
                        inq[v] = false;
                    }
                }
            }
    }
    return d[t] < INF;
}

void mcmf() {
    maxflow = cost = 0;
    while (spfa()) {
        int f = INF;
        for (int u = t; u != s; u = from[pre[u]])
            f = min(f, cap[pre[u]]);

        maxflow += f;
        cost += (long long)f * d[t];
        for (int u = t; u != s; u = from[pre[u]]) {
            cap[pre[u]] -= f;
            cap[pre[u] ^ 1] += f;
        }
    }
}

int main() {

    scanf("%d%d", &n, &m);
    s = 1, t = n;

    int a, b, c, l;
    tot = 1;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d%d", &a, &b, &c, &l);
        add(a, b, c, l);
    }
```

```
    mcmf();
    printf("%lld %lld\n", maxflow, cost);

    return 0;
}
```

2 数据结构

2.1 树状数组

2.1.1 1. 单点修改，区间查询

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000000 + 100;
typedef long long LL;

LL C[maxn];

int n, q;

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

LL sum(int x) {
    LL res = 0;
    for (; x; x -= lowbit(x))
        res += C[x];
    return res;
}

void add(int x, LL v) {
    for (; x <= n; x += lowbit(x))
        C[x] += v;
}

int main() {

    int v, op, l, r;
    cin >> n >> q;

    for (int i = 1; i <= n; i++) {
        cin >> v;
        add(i, v);
    }

    while (q--) {
        cin >> op >> l >> r;

        if (op == 2) {
            cout << sum(r) - sum(l - 1) << endl;
        } else {
            add(l, r);
```

```
        }
    }

    return 0;
}
```

2.1.2 2. 区间修改，单点查询（差分）

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

using namespace std;
typedef long long LL;
const int maxn = 1000000 + 100;

LL a[maxn], C[maxn];
int n, q;

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

LL sum(int x) {
    LL res = 0;
    for (; x; x -= lowbit(x))
        res += C[x];

    return res;
}

void add(int x, LL v) {
    for (; x <= n; x += lowbit(x))
        C[x] += v;
}

int main() {
    cin >> n >> q;

    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = n; i >= 1; i--) {
        a[i] -= a[i - 1];
        add(i, a[i]);
    }

    int op, l, r, v;
    for (int i = 1; i <= q; i++) {
        cin >> op;

        if (op == 1) {
            cin >> l >> r >> v;
            add(l, v);
            add(r + 1, -v);
        }

        if (op == 2) {
```

```
            cin >> v;
            cout << sum(v) << endl;
        }
    }
    return 0;
}
```

2.1.3 3. 区间修改，区间查询

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

typedef long long LL;
using namespace std;
const int maxn = 1000000 + 100;

//delta[i] 表示 a[i], a[i + 1] .... a[n]  都要加上一个数 delta[i]
//树状数组 C1 用于维护 delta[i], 树状数组 C2 维护 i * delta[i]
//前缀和 sum[i] = (i + 1) sum_delta[i] - sum_i*delta[i]

LL C1[maxn], C2[maxn];
int n;

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

void add(int pos, LL v) {
    for (int i = pos; i <= n; i += lowbit(i)) {
        C1[i] += v;
    }
    for (int i = pos; i <= n; i += lowbit(i)) {
        C2[i] += v * pos;
    }
}

LL sum(int pos) {
    LL res = 0;
    for (int i = pos; i; i -= lowbit(i)) {
        res += (pos + 1) * C1[i];
    }
    for (int i = pos; i; i -= lowbit(i)) {
        res -= C2[i];
    }
    return res;
}

int main() {
    int q;
    cin >> n >> q;

    int op, l, r;
    LL v;

    for (int i = 1; i <= n; i++) {
        cin >> v;
```

```
        add(i, v);
        add(i + 1, -v);
    }

    while(q--) {
        cin >> op;
        if(op == 1) {
            cin >> l >> r >> v;
            add(l, v);
            add(r + 1, -v);
        }
        else if(op == 2) {
            cin >> l >> r;
            cout << sum(r) - sum(l - 1) << endl;
        }
    }
    return 0;
}
```

2.2 线段树

2.2.1 1. 区间加法，区间求和

```
struct SegmentTree {
    LL sumv[maxn * 4], addv[maxn * 4]; //原数组大小的四倍

    void maintain(int o) {
        sumv[o] = sumv[lc] + sumv[rc];
    }

    void pushdown(int o,int L,int R) { //标记下传
        int M = L+R>>1;
        addv[lc] += addv[o];
        sumv[lc] += addv[o] *( M-L +1);

        addv[rc] += addv[o];
        sumv[rc] += addv[o] *(R-M);

        addv[o] = 0;

    }

    void build(int o, int L, int R) {
        if (L == R) {
            sumv[o] = v[L];
            return;
        }
        int M = L + R >> 1;
        build(lc, L, M);
        build(rc, M + 1, R);
        maintain(o);
    }

    LL query(int o, int L, int R, int l, int r) {
        if (l <= L && r >= R) return sumv[o];
        pushdown(o,L,R);
```

```
        int M = L + R >> 1;
        LL sum = 0;
        if(l <= M) sum += query(lc,L,M,l,r);
        if(r > M) sum += query(rc,M+1,R,l,r);
        return sum;
    }

    void update(int o,int L,int R,int l,int r,LL x){
        if(l <= L && r >= R){
            addv[o] += x;
            sumv[o] += (R-L+1) * x;
            return;
        }
        pushdown(o,L,R);
        int M = L+R>>1;
        if(l <= M) update(lc,L,M,l,r,x);
        if(r > M ) update(rc,M+1,R,l,r,x);
        maintain(o);
    }


} T;
```

2.2.2 2. 区间加法，乘法，区间求和

```
struct SegmentTree {
    LL sumv[maxn], addv[maxn], mulv[maxn];

    void maintain(int o) {
        sumv[o] = sumv[lc] + sumv[rc];
    }

    void pushdown(int o, int L, int R) {
        int M = L + R >> 1;
        if (mulv[o] != 1) {
            mulv[lc] *= mulv[o];
            addv[lc] *= mulv[o];
            sumv[lc] *= mulv[o];

            mulv[rc] *= mulv[o];
            addv[rc] *= mulv[o];
            sumv[rc] *= mulv[o];

            mulv[o] = 1;
        }
        if (addv[o]) {
            addv[lc] += addv[o];
            sumv[lc] += addv[o] * (M - L + 1);

            addv[rc] += addv[o];
            sumv[rc] += addv[o] * (R - M);

            addv[o] = 0;
        }
    }
```

```
    void build(int o, int L, int R) {
        addv[o] = 0;
        mulv[o] = 1;
        if (L == R) {
            sumv[o] = v[L];
            return;
        }
        int M = L + R >> 1;
        build(lc, L, M);
        build(rc, M + 1, R);
        maintain(o);
    }

    LL query(int o, int L, int R, int l, int r) {
        if (l <= L && r >= R) return sumv[o];
        pushdown(o, L, R);
        int M = L + R >> 1;
        LL sum = 0;
        if (l <= M) sum += query(lc, L, M, l, r);
        if (r > M) sum += query(rc, M + 1, R, l, r);
        return sum;
    }

    void update_add(int o, int L, int R, int l, int r, int x) {
        if (l <= L && r >= R) {
            addv[o] += x;
            sumv[o] += x * (R - L + 1);
            return;
        }
        pushdown(o, L, R);
        int M = L + R >> 1;
        if (l <= M) update_add(lc, L, M, l, r, x);
        if (r > M) update_add(rc, M + 1, R, l, r, x);
        maintain(o);
    }

    void update_mul(int o, int L, int R, int l, int r, int x) {
        if (l <= L && r >= R) {
            mulv[o] *= x;
            addv[o] *= x;
            sumv[o] *= x;
            return;
        }
        pushdown(o, L, R);
        int M = L + R >> 1;
        if (l <= M) update_mul(lc, L, M, l, r, x);
        if (r > M) update_mul(rc, M + 1, R, l, r, x);
        maintain(o);
    }
} T;
```

2.3 平衡树

2.3.1 Treap

操作
- 1 x 插入 x
- 2 x 删除 x，如果有多个，只删除 1 个
- 3 x 查询 x 的排名
- 4 x 查询排名为 x 的数
- 5 x 查询 x 的前驱（小于 x 的数中最大的）
- 6 x 查询 x 的后继（大于 x 的数中最小的）

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

using namespace std;

const int maxn = 100000 + 100;
const int INF = 0x3f3f3f3f;

//二叉搜索树 + 大根堆
struct Treap {

    int lc[maxn], rc[maxn];
    int val[maxn], rnd[maxn];
    int size[maxn], cnt[maxn];
    int tot, root;

    void maintain(int o) {
        size[o] = cnt[o] + size[lc[o]] + size[rc[o]];
    }

    int newnode(int v) {
        tot++;
        val[tot] = v, rnd[tot] = rand();
        size[tot] = cnt[tot] = 1;
        return tot;
    }

    void zig(int& p) { //右旋
        int q = lc[p];
        lc[p] = rc[q];
        rc[q] = p;
        p = q;
        maintain(rc[p]), maintain(p);
    }

    void zag(int& p) { //左旋
        int q = rc[p];
        rc[p] = lc[q];
        lc[q] = p;
        p = q;
        maintain(lc[p]), maintain(p);
    }

    void build() {
        root = newnode(-INF);
```

```
    insert(root, INF);
}

void insert(int& p, int v) {
    if (!p) {
        p = newnode(v);
        return;
    }
    if (val[p] == v) {
        cnt[p]++;
        size[p]++;
        return;
    }
    if (v < val[p]) {
        insert(lc[p], v);
        if (rnd[lc[p]] > rnd[p])
            zig(p);
    } else {
        insert(rc[p], v);
        if (rnd[rc[p]] > rnd[p])
            zag(p);
    }
    maintain(p);
}

void remove(int& p, int v) {
    if (!p) return;
    if (v == val[p]) {
        if (cnt[p] > 1) {
            cnt[p]--;
            size[p]--;
            return;
        }
        if (lc[p] && rc[p]) { //有两个儿子：将 p 旋到下层
            if (rnd[lc[p]] > rnd[rc[p]]) {
                zig(p);
                remove(rc[p], v);
            } else {
                zag(p);
                remove(lc[p], v);
            }
        } else { //只有一个儿子（或没有儿子），直接用儿子替代 p
            p = lc[p] | rc[p];
        }
        maintain(p);
        return;
    }

    if (v < val[p]) {
        remove(lc[p], v);
    } else {
        remove(rc[p], v);
    }
    maintain(p);
}
```

```
    int getRankByVal(int& p, int v) {
        if (!p) return 1;
        if (v == val[p])
            return size[lc[p]] + 1;
        else if (v < val[p])
            return getRankByVal(lc[p], v);
        else
            return getRankByVal(rc[p], v) + size[lc[p]] + cnt[p];
    }

    int getValByRank(int& p, int k) {
        if (k <= size[lc[p]])
            return getValByRank(lc[p], k);
        else if (k <= size[lc[p]] + cnt[p])
            return val[p];
        else
            return getValByRank(rc[p], k - size[lc[p]] - cnt[p]);
    }

    int getPre(int v) { //查询前驱
        int cur = root;
        int res = 1; // val[1] = -INF
        while (cur) {
            if (val[cur] < v && val[cur] > val[res]) res = cur;
            cur = val[cur] < v ? rc[cur] : lc[cur];
        }
        return val[res];
    }

    int getNxt(int v) {
        int cur = root;
        int res = 2; // val[2] = INF
        while (cur) {
            if (val[cur] > v && val[cur] < val[res]) res = cur;
            cur = val[cur] > v ? lc[cur] : rc[cur];
        }
        return val[res];
    }
} T;


int main() {
    int m;
    cin >> m;
    int op, x;
    T.build();
    while (m--) {
        cin >> op >> x;
        if (op == 1) {
            T.insert(T.root, x);
        } else if (op == 2) {
            T.remove(T.root, x);
        } else if (op == 3) {
            cout << T.getRankByVal(T.root, x) - 1 << endl;
```

```
        } else if (op == 4) {
            cout << T.getValByRank(T.root, x + 1) << endl;
        } else if (op == 5) {
            cout << T.getPre(x) << endl;
        } else if (op == 6) {
            cout << T.getNxt(x) << endl;
        }
    }

    return 0;
}
```

2.3.2 Splay

2.3.2.0.1 文艺平衡树 初始序列为 1 2 3 … n

每次操作输入 l r, 将 a[l], a[l + 1] … a[r] 翻转

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

using namespace std;

const int maxn = 100000 + 100;

int n, m;

struct Splay {
    int val[maxn], size[maxn];
    bool rev[maxn];
    int ch[maxn][2], p[maxn];
    int root, tot;

    int newnode(int v) {
        tot++;
        val[tot] = v, size[tot] = 1;
        return tot;
    }

    void maintain(int x) {
        size[x] = size[ch[x][0]] + size[ch[x][1]] + 1;
    }

    void pushdown(int x) {
        if (rev[x]) {
            swap(ch[x][0], ch[x][1]);
            rev[ch[x][0]] ^= 1;
            rev[ch[x][1]] ^= 1;

            rev[x] ^= 1;
        }
    }
    int chk(int x) {
        return ch[p[x]][1] == x;
    }
    //将节点 x 向上旋一层
    void rotate(int x) {
        int y = p[x], z = p[y];
```

```
        int k = chk(x);
        ch[z][chk(y)] = x, p[x] = z;
        ch[y][k] = ch[x][!k], p[ch[y][k]] = y;
        ch[x][!k] = y, p[y] = x;
        maintain(y), maintain(x);
    }
    //将节点 x 旋到节点 k 的下面，如果 k=0 则将 x 旋到根
    void splay(int x, int k) {
        while (p[x] != k) {
            int y = p[x], z = p[y];
            if (z != k)
                if (chk(x) == chk(y))
                    rotate(y);
                else
                    rotate(x);
            rotate(x);
        }
        if (!k) root = x;
    }
    //插入一个数 （用于创建初始序列）
    void insert(int v) {
        int cur = root, pa = 0;
        while (cur)
            pa = cur, cur = ch[cur][v > val[cur]];
        cur = newnode(v);
        if (pa) ch[pa][v > val[pa]] = cur, p[cur] = pa;
        splay(cur, 0);
    }
    //查询序列中第 k 个数对应的节点
    int find_k(int k) {
        int cur = root;
        while (true) {
            pushdown(cur);
            if (k <= size[ch[cur][0]])
                cur = ch[cur][0];
            else if (k <= size[ch[cur][0]] + 1)
                return cur;
            else
                k -= size[ch[cur][0]] + 1, cur = ch[cur][1];
        }
    }
    //打印序列
    void print(int x) {
        pushdown(x);
        if (ch[x][0]) print(ch[x][0]);
        if (val[x] >= 1 && val[x] <= n) printf("%d ", val[x]);
        if (ch[x][1]) print(ch[x][1]);
    }

} T;

int main() {

    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n + 1; i++) // 0 和 n + 1 为哨兵
```

```
        T.insert(i);

    int l, r;
    while (m--) {
        scanf("%d%d", &l, &r);
        l = T.find_k(l);
        r = T.find_k(r + 2);
        T.splay(l, 0);
        T.splay(r, l);
        T.rev[T.ch[r][0]] ^= 1;
    }
    T.print(T.root);
    printf("\n");

    return 0;
}
```

编号	名称	格式	说明
1	插入	INSERT $posi$ tot c_1 $c_2 \cdots c_{tot}$	在当前数列的第 $posi$ 个数字后插入 tot 个数字：$c_1, c_2 \cdots c_{tot}$；若在数列首插入，则 $posi$ 为 0
2	删除	DELETE $posi$ tot	从当前数列的第 $posi$ 个数字开始连续删除 tot 个数字
3	修改	MAKE-SAME $posi$ tot c	从当前数列的第 $posi$ 个数字开始的连续 tot 个数字统一修改为 c
4	翻转	REVERSE $posi$ tot	取出从当前数列的第 $posi$ 个数字开始的 tot 个数字，翻转后放入原来的位置
5	求和	GET-SUM $posi$ tot	计算从当前数列的第 $posi$ 个数字开始的 tot 个数字的和并输出
6	求最大子列和	MAX-SUM	求出当前数列中和最大的一段子列，并输出最大和

Figure 1: img

2.3.2.0.2 NOI2005 维护数列

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>

#define lc ch[x][0]
#define rc ch[x][1]
```

```cpp
using namespace std;

const int maxn = 500000 + 100;
int w[maxn];
struct Splay {
    int ch[maxn][2], p[maxn];
    int val[maxn], sumv[maxn];
    int maxsum[maxn], presum[maxn], sufsum[maxn];
    bool rev[maxn], setd[maxn];
    int size[maxn];
    int root, tot;
    queue<int> buff;

    void flip(int x) {
        rev[x] ^= 1;
        swap(lc, rc);
        swap(presum[x], sufsum[x]);
    }
    void setto(int x, int v) {
        setd[x] = true;
        val[x] = v;
        sumv[x] = v * size[x];
        if (v > 0) {
            maxsum[x] = presum[x] = sufsum[x] = v * size[x];
        } else {
            maxsum[x] = v;
            presum[x] = sufsum[x] = 0;
        }
    }

    void maintain(int x) {
        size[x] = 1 + size[lc] + size[rc];
        sumv[x] = sumv[lc] + sumv[rc] + val[x];
        presum[x] = max(presum[lc], sumv[lc] + val[x] + presum[rc]);
        sufsum[x] = max(sufsum[rc], sumv[rc] + val[x] + sufsum[lc]);
        maxsum[x] = max(max(maxsum[lc], maxsum[rc]), sufsum[lc] + val[x] + presum[rc]);
    }

    void pushdown(int x) {
        if (setd[x]) {
            setd[x] = rev[x] = false;
            if (lc) setto(lc, val[x]);
            if (rc) setto(rc, val[x]);
        }

        if (rev[x]) {
            rev[x] = false;
            if (lc) flip(lc);
            if (rc) flip(rc);
        }
    }

    int chk(int x) {
        return ch[p[x]][1] == x;
    }
```

```
void rotate(int x) {
    int y = p[x], z = p[y];
    int k = chk(x);
    ch[z][chk(y)] = x, p[x] = z;
    ch[y][k] = ch[x][!k], p[ch[y][k]] = y;
    ch[x][!k] = y, p[y] = x;
    maintain(y), maintain(x);
}

void splay(int x, int k) {
    while (p[x] != k) {
        int y = p[x], z = p[y];
        if (z != k)
            if (chk(x) == chk(y))
                rotate(y);
            else
                rotate(x);
        rotate(x);
    }
    if (!k) root = x;
}

int newnode(int v, int _p) {
    int x;
    if (buff.size())
        x = buff.front(), buff.pop();
    else
        x = ++tot;

    lc = rc = 0;
    p[x] = _p;

    size[x] = 1;
    val[x] = sumv[x] = maxsum[x] = v;
    presum[x] = sufsum[x] = v > 0 ? v : 0;
    rev[x] = setd[x] = false;
    return x;
}

int get_k(int k) {
    int x = root;
    while (x) {
        pushdown(x);
        if (k <= size[lc])
            x = lc;
        else if (k == size[lc] + 1)
            return x;
        else
            k -= size[lc] + 1, x = rc;
    }
    return -1;
}

int build(int l, int r, int _p) {
```

```
        int m = l + r >> 1;
        int x = newnode(w[m], _p);
        if (l < m) lc = build(l, m - 1, x);
        if (r > m) rc = build(m + 1, r, x);
        maintain(x);
        return x;
    }
} T;

void dfs(int x) { //回收节点
    if (T.lc) dfs(T.lc);
    if (T.rc) dfs(T.rc);
    T.buff.push(x);
}

int readIn() {
    int x = 0;
    int flag = 1;
    char c = getchar();
    while (c < '0' || c > '9') {
        if (c == '-')
            flag = -flag;
        c = getchar();
    }
    while (c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * flag;
}

int main() {
    int n, m;
    n = readIn(), m = readIn();
    for (int i = 1; i <= n; i++)
        w[i] = readIn();
    w[0] = w[n + 1] = -1e8;
    T.maxsum[0] = -1e8;

    T.root = T.build(0, n + 1, 0);

    char op[20];
    int pos, k, v;
    while (m--) {
        scanf("%s", op);
        if (!strcmp(op, "INSERT")) {
            pos = readIn(), k = readIn();
            for (int i = 1; i <= k; i++)
                w[i] = readIn();
            int l = T.get_k(pos + 1), r = T.get_k(pos + 2);
            T.splay(l, 0), T.splay(r, l);
            T.ch[r][0] = T.build(1, k, r);
            T.maintain(r), T.maintain(l);
        } else if (!strcmp(op, "DELETE")) {
            pos = readIn(), k = readIn();
```

```
            int l = T.get_k(pos), r = T.get_k(pos + k + 1);
            T.splay(l, 0), T.splay(r, l);
            dfs(T.ch[r][0]);
            T.ch[r][0] = 0;
            T.maintain(r), T.maintain(l);
        } else if (!strcmp(op, "MAKE-SAME")) {
            pos = readIn(), k = readIn(), v = readIn();
            int l = T.get_k(pos), r = T.get_k(pos + k + 1);
            T.splay(l, 0), T.splay(r, l);
            T.setto(T.ch[r][0], v);
            T.maintain(r), T.maintain(l);
        } else if (!strcmp(op, "REVERSE")) {
            pos = readIn(), k = readIn();
            int l = T.get_k(pos), r = T.get_k(pos + k + 1);
            T.splay(l, 0), T.splay(r, l);
            T.flip(T.ch[r][0]);
            T.maintain(r), T.maintain(l);
        } else if (!strcmp(op, "GET-SUM")) {
            pos = readIn(), k = readIn();
            int l = T.get_k(pos), r = T.get_k(pos + k + 1);
            T.splay(l, 0), T.splay(r, l);
            printf("%d\n", T.sumv[T.ch[r][0]]);
        } else if (!strcmp(op, "MAX-SUM")) {
            printf("%d\n", T.maxsum[T.root]);
        }
    }

    return 0;
}
```

2.4 树链剖分（轻重链剖分）

给定一棵树，每个点有 1 个权值，支持以下 4 种操作
1. 将 u->v 路径上的节点权值加上 k
2. 查询 u->v 路径上节点权值的和
3. 将以 u 和根的子树上每个节点权值加上 k
4. 查询以 u 为根的子树上每个节点的权值之和

```
#include <algorithm>
#include <cstdio>
#include <cstring>

#define lc (o << 1)
#define rc (o << 1 | 1)

using namespace std;
typedef long long LL;

const int maxn = 100000 + 10;
const int maxm = 2 * maxn;

int a[maxn];
int to[maxm], nxt[maxm], fir[maxn], tot;

int sz[maxn], son[maxn], dep[maxn], father[maxn];
```

```cpp
int top[maxn], pos[maxn], b[maxn]; // 所在重链的顶节点，在新序列中的位置，新序列中的值
int cnt;
int n;
void add_edge(int u, int v) {
    tot++;
    to[tot] = v, nxt[tot] = fir[u], fir[u] = tot;
}
/*
dfs1: 计算子树大小 sz, 重儿子 son, 深度 dep, 父节点 father
*/
void dfs1(int u, int f) {
    sz[u] = 1;
    father[u] = f;
    dep[u] = dep[f] + 1;
    for (int e = fir[u]; e; e = nxt[e]) {
        int v = to[e];
        if (v == f) continue;
        dfs1(v, u);
        sz[u] += sz[v];
        if (sz[son[u]] < sz[v]) son[u] = v;
    }
}
/*
dfs2: 计算节点所在重链的顶端节点 top, 节点在序列中的位置 pos, 序列中的值 b
*/
void dfs2(int u, int t) {
    top[u] = t;
    pos[u] = ++cnt;
    b[cnt] = a[u];

    if (!son[u]) return;
    dfs2(son[u], t); //重儿子

    for (int e = fir[u]; e; e = nxt[e]) {
        int v = to[e];
        if (v == son[u] || v == father[u]) continue;
        dfs2(v, v);
    }
}

//线段树
struct SegmentTree {
    LL sumv[maxn * 4], addv[maxn * 4];

    void maintain(int o) {
        sumv[o] = sumv[lc] + sumv[rc];
    }

    void pushdown(int o, int L, int R) {
        if (addv[o]) {
            int M = L + R >> 1;
            addv[lc] += addv[o], sumv[lc] += addv[o] * (M - L + 1);
            addv[rc] += addv[o], sumv[rc] += addv[o] * (R - M);
            addv[o] = 0;
        }
```

```
    }

    void build(int o, int L, int R) {
        if (L == R) {
            sumv[o] = b[L];
            addv[o] = 0;
            return;
        }
        int M = L + R >> 1;
        build(lc, L, M);
        build(rc, M + 1, R);
        maintain(o);
    }

    void update(int o, int L, int R, int l, int r, LL v) {
        if (l <= L && r >= R) {
            sumv[o] += v * (R - L + 1);
            addv[o] += v;
            return;
        }
        pushdown(o, L, R);
        int M = L + R >> 1;
        if (l <= M) update(lc, L, M, l, r, v);
        if (r > M) update(rc, M + 1, R, l, r, v);
        maintain(o);
    }

    LL query(int o, int L, int R, int l, int r) {
        if (l <= L && r >= R) return sumv[o];
        pushdown(o, L, R);
        int M = L + R >> 1;
        LL res = 0;
        if (l <= M) res += query(lc, L, M, l, r);
        if (r > M) res += query(rc, M + 1, R, l, r);
        return res;
    }

} T;
/*
    将路径转化为 O(log(n)) 个区间
    u -> v
    将 u 和 v 中 top[i] 深的向上跳，直到 u 和 v 在同一条重链
*/
void update_path(int u, int v, int k) {
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        T.update(1, 1, n, pos[top[u]], pos[u], k);
        u = father[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    T.update(1, 1, n, pos[v], pos[u], k);
}

LL query_path(int u, int v) {
    LL res = 0;
```

```
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        res += T.query(1, 1, n, pos[top[u]], pos[u]);
        u = father[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    res += T.query(1, 1, n, pos[v], pos[u]);
    return res;
}

void update_tree(int u, int k) {
    T.update(1, 1, n, pos[u], pos[u] + sz[u] - 1, k);
}
LL query_tree(int u) {
    return T.query(1, 1, n, pos[u], pos[u] + sz[u] - 1);
}

int main() {

    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", a + i);
    }
    int op, u, v, k;
    for (int i = 1; i < n; i++) {
        scanf("%d%d", &u, &v);
        add_edge(u, v);
        add_edge(v, u);
    }

    dfs1(1, 0); // sz, son, dep,  father
    dfs2(1, 1); // top, pos, b

    T.build(1, 1, n);

    int q;
    scanf("%d", &q);

    while (q--) {

        scanf("%d", &op);

        if (op == 1) {
            scanf("%d%d%d", &u, &v, &k);
            update_path(u, v, k);
        } else if (op == 2) {
            scanf("%d%d", &u, &k);
            update_tree(u, k);
        } else if (op == 3) {
            scanf("%d%d", &u, &v);
            LL res = query_path(u, v);
            printf("%lld\n", res);
        } else if (op == 4) {
            scanf("%d", &u);
            LL res = query_tree(u);
```

```
            printf("%lld\n", res);
        }
    }
    return 0;
}
```

3 数学

3.1 数论

3.1.1 Baby Step Giant Step 算法

Baby Step, Giant Step 算法:
求解不定方程 $a^x \equiv b \pmod{p}$, $(a, p$ 互质$)$
设 $x = it - j$ $(1 \le i \le t, 0 \le j \le t)$ $(t^2 \ge p)$ 注意 i 不能从 0 开始，否则会产生负数解
j 必须取到 t，否则可能漏解 $x = 0$
$a^{it} \equiv ba^j \pmod{p}$
先枚举右边并存到哈希表，然后枚举左边的值，并在哈希表中查找解。

```
LL bsgs(LL a, LL b, LL p) {
    b %= p;
    unordered_map<LL, LL> hash;
    LL t = sqrt(p) + 1;
    LL x = 1;
    for (LL j = 0; j < t; j++) {
        LL val = b * x % p;
        hash[val] = j;
        x = x * a % p;
    }
    hash[b * x % p] = t;
    a = x;
    for (LL i = 1; i <= t; i++) {
        if (hash.count(x)) {
            return i * t - hash[x];
        }
        x = x * a % p;
    }
    return -1;
}
```

3.1.2 扩展 BSGS

题目同上，但不保证 a p 互质

$a^x = b \ mod \ p$
1. 先判断 $x = 0$ 是不是解, 是则直接返回
2. 设 $d = gcd(a, p)$，如果 $b \nmid d$, 则无解，否则 $\frac{a}{d} a^{x-1} = \frac{b}{d} \ mod \ \frac{p}{d}$
3. $a^{x-1} = \frac{b}{d}(\frac{a}{d})^{-1} \ mod \ \frac{p}{d}$，递归求解

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

using namespace std;
typedef long long LL;
const LL INF = 0x3f3f3f3f3f3f3f3f;

LL exgcd(LL a, LL b, LL& x, LL& y) {
    if (b == 0) {
```

```
        x = 1, y = 0;
        return a;
    } else {
        LL d = exgcd(b, a % b, y, x);
        y -= a / b * x;
        return d;
    }
}

LL bsgs(LL a, LL b, LL p) {
    b %= p;

    unordered_map<LL, LL> hash;
    LL x = 1;
    LL t = sqrt(p) + 1;

    for (int j = 0; j < t; j++) {
        hash[x * b % p] = j;
        x = x * a % p;
    }
    hash[x * b % p] = t;

    a = x;
    for (int i = 1; i <= t; i++) {
        if (hash.count(x)) return i * t - hash[x];
        x = x * a % p;
    }
    return -INF;
}

LL exbsgs(LL a, LL b, LL p) {
    if (1 % p == b % p) return 0;

    LL x, y, d;
    d = exgcd(a, p, x, y);
    if (d == 1) {
        return bsgs(a, b, p);
    } else {
        if (b % d) return -INF;
        p /= d;
        //a / d 模 p 的逆元
        exgcd(a / d, p, x, y);
        x = (x % p + p) % p;
        return 1 + exbsgs(a, b / d * x % p, p);
    }
}

int main() {
    LL a, p, b;
    while (cin >> a >> p >> b, a || p || b) {
        LL res = exbsgs(a, b, p);
        if (res < 0)
            cout << "No Solution" << endl;
        else
            cout << res << endl;
```

```
    }

    return 0;
}
```

3.2 多项式卷积

3.2.1 快速傅里叶变换 FFT

```
#include <algorithm>
#include <cmath>
#include <cstdio>

using namespace std;

const int maxn = 300000 + 100;
const double pi = acos(-1);

struct Complex {
    double x, y;

    Complex() {
        x = y = 0;
    }
    Complex(double x, double y) : x(x), y(y) {}

    Complex operator+(const Complex& b) const {
        return Complex(x + b.x, y + b.y);
    }

    Complex operator-(const Complex& b) const {
        return Complex(x - b.x, y - b.y);
    }

    Complex operator*(const Complex& b) const {
        return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
    }
} a[maxn], b[maxn];

int rev[maxn];
int n, m, N, l;

void fft(Complex* A, int N, int type) {

    for (int i = 0; i < N; i++)
        if (rev[i] < i)
            swap(A[i], A[rev[i]]);

    for (int len = 2; len <= N; len <<= 1) {
        int m = len >> 1;
        Complex wn(cos(2 * pi / len), type * sin(2 * pi / len));
        for (int j = 0; j < N; j += len) {
            Complex w(1, 0);

            for (int i = 0; i < m; i++) {
                Complex u = A[j + i];
```

```
                Complex v = A[j + i + m];

                A[j + i] = u + w * v;
                A[j + i + m] = u - w * v;

                w = w * wn;
            }
        }
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n; i++)
        scanf("%lf", &a[i].x);
    for (int i = 0; i <= m; i++)
        scanf("%lf", &b[i].x);
    N = 1, l = 0;
    while (N < n + m + 1) {
        N <<= 1;
        l++;
    }

    for (int i = 0; i < N; i++)
        rev[i] = rev[i >> 1] >> 1 | (i & 1) << (l - 1);

    fft(a, N, 1);
    fft(b, N, 1);

    for (int i = 0; i < N; i++)
        a[i] = a[i] * b[i];

    fft(a, N, -1);

    for (int i = 0; i <= n + m; i++)
        printf("%d ", (int)(a[i].x / N + 0.5));

    return 0;
}
```

3.2.2 快速数论变换 NTT

```
#include <algorithm>
#include <cmath>
#include <cstdio>

using namespace std;

typedef long long LL;

const LL P = 998244353;
const LL g = 3;
const int maxn = 300000 + 100;

int n, m, N, l;
```

```
LL a[maxn], b[maxn];
int rev[maxn];

LL pow_mod(LL a, LL b, LL p) {
    LL res = 1;
    while (b) {
        if (b & 1)
            res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}

void fft(LL* A, int N, int type) {
    for (int i = 0; i < N; i++)
        if (rev[i] < i)
            swap(A[i], A[rev[i]]);

    for (int len = 2; len <= N; len <<= 1) {
        int m = len / 2;
        LL wn = pow_mod(g, (P - 1) / len, P);
        if (type == -1)
            wn = pow_mod(wn, P - 2, P);
        for (int j = 0; j < N; j += len) {
            LL w = 1;

            for (int i = 0; i < m; i++) {
                LL u = A[j + i];
                LL v = A[j + i + m];

                A[j + i] = (u + w * v % P) % P;
                A[j + i + m] = (u - w * v % P + P) % P;

                w = w * wn % P;
            }
        }
    }
}

int main() {

    scanf("%d%d", &n, &m);

    for (int i = 0; i <= n; i++)
        scanf("%lld", a + i);
    for (int i = 0; i <= m; i++)
        scanf("%lld", b + i);

    N = 1, l = 0;
    while (N < n + m + 1) {
        N <<= 1;
        l++;
    }
```

```
    for (int i = 0; i < N; i++)
        rev[i] = rev[i >> 1] >> 1 | (i & 1) << (l - 1);

    fft(a, N, 1);
    fft(b, N, 1);

    for (int i = 0; i < N; i++)
        a[i] = a[i] * b[i] % P;

    fft(a, N, -1);

    LL aN = pow_mod(N, P - 2, P);

    for (int i = 0; i <= n + m; i++)
        printf("%lld ", a[i] * aN % P);

    return 0;
}
```

3.2.3 多项式求逆

如果 $F(x) * G(x) \equiv 1 \mod x^n$ (系数对 998255435 取模)，则称多项式 $G(x)$ 是多项式 $F(x)$ 的逆。

如果 $F(x) * H(x) \equiv 1 \ mod \ x^t$
$G(x) = 2H(x) - F(x) * H^2(x)$
则 $F(x) * G(x) \equiv 1 \ mod \ x^{2*t}$
从 $t = 1$ 开始向上递推，直到 t 不小于 n

```
int main() {
    int n;
    n = readIn();

    for (int i = 0; i < n; i++)
        F[i] = readIn();

    G[0] = pow_mod(F[0], P - 2, P);
    int t = 1;
    //G[x] * F[x] = 1 (mod x^t)
    while (t < n) {
        t <<= 1;
        int N = 1, l = 0;

        while (N <= t) {
            N <<= 1;
            l++;
        }

        for (int i = 0; i < N; i++)
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << (l - 1);

        for (int i = 0; i < t; i++)
            T[i] = F[i];
        for (int i = t; i < N; i++)
            T[i] = 0;
```

```
        ntt(G, N, 1);
        ntt(T, N, 1);

        for (int i = 0; i < N; i++) {
            G[i] = (2 * G[i] % P - (T[i] * G[i] % P) * G[i] % P + P) % P;
        }

        ntt(G, N, -1);
        LL aN = pow_mod(N, P - 2, P);
        for (int i = 0; i < N; i++) {
            G[i] = G[i] * aN % P;
        }
        for (int i = t; i < N; i++)
            G[i] = 0;
    }

    for (int i = 0; i <= n; i++)
        printf("%lld ", G[i]);
    printf("\n");

    return 0;
}
```

3.2.4 NTT 素数原根表

$P - 1 = r * 2^k$
P	r	k	g
998244353	119	23	3
1004535809	479	21	3

4 其它

4.1 字符串算法

4.1.1 KMP 字符串匹配

```
// Luogu P3375

#include <cstdio>
#include <cstring>

const int maxn = 1000000 + 100;
int fail[maxn];

char T[maxn], S[maxn]; // 文本串，模式串

void getFail() {
    int n = strlen(S + 1);
    fail[1] = 0;
    for (int i = 2; i <= n; i++) {
        int j = fail[i - 1];
        while (j && S[j + 1] != S[i])
            j = fail[j];
        if (S[j + 1] == S[i]) j++;
```

```
        fail[i] = j;
    }
}

int main() {
    scanf("%s%s", T + 1, S + 1);
    int n = strlen(S + 1), m = strlen(T + 1);
    getFail();
    int j = 0;
    for (int i = 1; i <= m; i++) {
        while (j && S[j + 1] != T[i])
            j = fail[j];
        if (S[j + 1] == T[i]) {
            j++;
            if (j == n) {
                printf("%d\n", i - n + 1);
                j = fail[j];
            }
        }
    }
    for(int i = 1; i <= n; i++)
        printf("%d ", fail[i]);

    return 0;
}
```

4.1.2 AC 自动机

查询每个模式串在文本串中出现的次数

```
//Luogo P5357

#include <cstdio>
#include <cstring>
#include <queue>

using namespace std;

const int maxn = 200000 + 100;
const int maxm = 2000000 + 100;

int trie[maxn][26], fail[maxn], pos[maxn], cnt[maxn], tot;
char s[maxm];


void insert(char* s, int id) {
    int len = strlen(s), u = 0;
    for (int i = 0; i < len; i++) {
        if (!trie[u][s[i] - 'a']) trie[u][s[i] - 'a'] = ++tot;
        u = trie[u][s[i] - 'a'];
    }
    pos[id] = u;
}

void getFail() {
    queue<int> Q;
```

```
    for (int i = 0; i < 26; i++)
        if (trie[0][i])
            Q.push(trie[0][i]);

    while (Q.size()) {
        int u = Q.front();
        Q.pop();

        for (int i = 0; i < 26; i++) {
            if (trie[u][i]) {
                fail[trie[u][i]] = trie[fail[u]][i];
                Q.push(trie[u][i]);
            } else
                trie[u][i] = trie[fail[u]][i];
        }
    }
}

int fir[maxn], nxt[maxn], to[maxn];
int idx;
void add(int a, int b) {
    idx++;
    to[idx] = b, nxt[idx] = fir[a], fir[a] = idx;
}

void dfs(int u) {
    for (int e = fir[u]; e; e = nxt[e]) {
        dfs(to[e]);
        cnt[u] += cnt[to[e]];
    }
}

void query(char* s) {
    int n = strlen(s);
    int u = 0;
    for (int i = 0; i < n; i++) {
        u = trie[u][s[i] - 'a'];
        cnt[u]++;
    }
    for (int i = 1; i <= tot; i++)
        add(fail[i], i);
    dfs(0);
}

int main() {
    int n;
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        scanf("%s", s);
        insert(s, i);
    }
    getFail();
    scanf("%s", s);
    query(s);
```

```
    for (int i = 1; i <= n; i++)
        printf("%d\n", cnt[pos[i]]);

    return 0;
}
```

4.2 莫队

4.2.1 普通莫队

题目: HH 的项链
查询: 区间 $[l, r]$ 是有多少种不同的颜色
莫队排序:
先考虑左端点所在块编号
再考虑右端点编号

块长度 $len = \sqrt{n^2/m}$
时间复杂度 $O(\sqrt{n^2/m})$

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>

using namespace std;

const int maxn = 50000 + 100;
const int maxm = 200000 + 100;
const int maxs = 1000000 + 100;

int color[maxn];
int cnt[maxs];
int n, m, len;

int ans[maxm];

inline void add(int i, int& res) {
    if (cnt[color[i]] == 0)
        res++;
    cnt[color[i]]++;
}

inline void del(int i, int& res) {
    if (cnt[color[i]] == 1)
        res--;
    cnt[color[i]]--;
}

inline int get(int i) {
    return i / len;
}

struct Query {
    int id, l, r;

    bool operator<(const Query& phs) {
```

```
        if (get(l) == get(phs.l)) {
            return r < phs.r;
        }
        return get(l) < get(phs.l);
    }

} q[maxm];

int read() {
    int x = 0;
    char c = getchar();
    while (c < '0' || c > '9') {
        c = getchar();
    }
    while (c >= '0' && c <= '9') {
        x = (x << 3) + (x << 1) + c - '0';
        c = getchar();
    }
    return x;
}

int main() {

    n = read();
    for (int i = 1; i <= n; i++)
        color[i] = read();

    m = read();
    for (int i = 1; i <= m; i++) {
        q[i].id = i;
        q[i].l = read();
        q[i].r = read();
    }

    len = sqrt((double) n * n / m);

    sort(q + 1, q + 1 + m);

    int i = 1, j = 0;
    int res = 0;
    for (int k = 1; k <= m; k++) {
        while (i < q[k].l)
            del(i++, res);
        while (i > q[k].l)
            add(--i, res);
        while (j < q[k].r)
            add(++j, res);
        while (j > q[k].r)
            del(j--, res);
        ans[q[k].id] = res;
    }

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

```
    return 0;
}
```

4.2.2 带修改的莫队

题目: 数颜色 Luogu P3939
按每次修改操作划分时间戳
排序:
1. 左端点块编号
2. 右端点块编号
3. 时间戳

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

using namespace std;

const int maxn = 10000 + 10;

int color[maxn];
int cnt[1000000 + 100];

int len;

int get(int x) {
    return x / len;
}

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

    bool operator<(const Query& phs) const {
        int la = get(l);
        int ra = get(r);
        int lb = get(phs.l);
        int rb = get(phs.r);

        if (la != lb) return la < lb;
        if (ra != rb) return ra < rb;
        return t < phs.t;
    }

} query[maxn];

struct Replace {
    int pos;
    int col;
} rp[1010];

void add(int v, int& res) {
    if (cnt[v] == 0)
        res++;
    cnt[v]++;
}
void del(int v, int& res) {
    if (cnt[v] == 1)
        res--;
```

```
    cnt[v]--;
}

int ans[maxn];

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    for(int i = 1;i <= n; i++)
        scanf("%d", color + i);

    char op[2];
    int l, r;

    int cq = 0;
    int ct = 0;

    for (int i = 1; i <= m; i++) {
        scanf("%s%d%d", op, &l, &r);
        if (*op == 'Q') {
            cq++;
            query[cq] = {cq, l, r, ct};
        } else {
            rp[++ct] = {l, r};
        }
    }
    len = pow(n * ct, 1.f / 3) + 1;
    sort(query + 1, query + cq + 1);

    int res = 0;
    int i = 1, j = 0;
    int t = 0;
    for (int k = 1; k <= cq; k++) {

        int id = query[k].id;
        int l = query[k].l;
        int r = query[k].r;

        while (i > l)
            add(color[--i], res);
        while (i < l)
            del(color[i++], res);
        while (j < r)
            add(color[++j], res);
        while (j > r)
            del(color[j--], res);

        while (t < query[k].t) {
            t++;
            if (rp[t].pos >= l && rp[t].pos <= r) {
                del(color[rp[t].pos], res);
                add(rp[t].col, res);
            }
```

```
            swap(rp[t].col, color[rp[t].pos]);
        }
        while (t > query[k].t) {
            if (rp[t].pos >= l && rp[t].pos <= r) {
                del(color[rp[t].pos], res);
                add(rp[t].col, res);
            }
            swap(rp[t].col, color[rp[t].pos]);
            t--;
        }
        ans[id] = res;
    }

    for (int i = 1; i <= cq; i++)
        printf("%d\n", ans[i]);

    return 0;
}
```

4.2.3 带回滚的莫队

题目: 历史研究 luogu AT1219

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

using namespace std;

typedef long long LL;

const int maxn = 100000 + 100;

int color[maxn];
vector<int> temp;

int cnt[maxn];
LL ans[maxn];

int len;

inline int get(int x) {
    return x / len;
}

struct Query {
    int id, l, r;

    bool operator<(const Query& phs) {
        int la = get(l);
        int lb = get(phs.l);
        if (la != lb) return la < lb;
        return r < phs.r;
    }
} query[maxn];

void add(int x, LL& res) {
    cnt[x]++;
```

```
    if ((LL)temp[x] * cnt[x] > res)
        res = (LL)temp[x] * cnt[x];
}

int readIn() {
    int x = 0;
    char c = getchar();
    while (c < '0' || c > '9')
        c = getchar();

    while (c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x;
}

int main() {

    int n, m;
    n = readIn();
    m = readIn();
    len = sqrt(n);

    for (int i = 1; i <= n; i++) {
        color[i] = readIn();
        temp.push_back(color[i]);
    }

    //离散化
    sort(temp.begin(), temp.end());
    temp.erase(unique(temp.begin(), temp.end()), temp.end());
    for (int i = 1; i <= n; i++) {
        color[i] = lower_bound(temp.begin(), temp.end(), color[i]) - temp.begin();
    }

    for (int i = 1; i <= m; i++) {
        query[i].id = i;
        cin >> query[i].l >> query[i].r;
    }

    sort(query + 1, query + 1 + m);

    for (int x = 1; x <= m;) {
        int y = x;

        //块号
        int block = get(query[x].l);

        //块的右端点
        int right = (len - 1) + block * len;

        while (y <= m && query[y].l <= right)
            y++;
```

```
        while (x < y && query[x].r <= right) {
            LL res = 0;
            for (int i = query[x].l; i <= query[x].r; i++)
                add(color[i], res);
            ans[query[x].id] = res;

            for (int i = query[x].l; i <= query[x].r; i++)
                cnt[color[i]]--;

            x++;
        }

        int r = right;
        LL res = 0;
        while (x < y) {
            while (r < query[x].r) {
                add(color[++r], res);
            }

            LL backup = res;

            for (int i = right; i >= query[x].l; i--)
                add(color[i], res);

            ans[query[x].id] = res;

            res = backup;
            for (int i = right; i >= query[x].l; i--)
                cnt[color[i]]--;

            x++;
        }

        memset(cnt, 0, sizeof(cnt));
    }

    for (int i = 1; i <= m; i++)
        cout << ans[i] << endl;

    return 0;
}
```

4.2.4 树上莫队

题目:
查询树上 (u -> v) 路径上不同权值的个数
做法:
转化为 dfs 序上的区间问题，然后莫队

first[u], last[u] 分别为 u 在 dfs 序上前后两次出现的位置，刚 (u -> v) (保证 u 在 dfs 序中的第一次出现先于 v) 可转化为:
1. 如果 lca(u, v) == u: first[u] ~ first[v] 中只出现一次的节点
2. 如果 lca(u, v) != u: last[u] ~ first[v] 中只出现一次的节点 + lca

```
#include <algorithm>
#include <cmath>
#include <cstdio>
```

```cpp
#include <cstring>
#include <vector>

using namespace std;

const int maxn = 100000 + 100;

int color[maxn];
vector<int> nums;

//邻接表
int to[maxn], fir[maxn];
int nxt[maxn], idx;

int n, m, len;

//莫队
int cnt[maxn], vis[maxn];
int ans[maxn];

inline int get(int x) {
    return x / len;
}

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

    bool operator<(const Query& phs) const {
        int la = get(l);
        int lb = get(phs.l);
        if (la != lb) return la < lb;
        return r < phs.r;
    }

} query[maxn];

inline void add_edge(int u, int v) {
    idx++;
    to[idx] = v;
    nxt[idx] = fir[u];
    fir[u] = idx;
}

//LCA
int dep[maxn], st[maxn][16];

//欧拉序列
int seq[maxn], top;
int first[maxn], last[maxn];

void dfs(int u, int father) {

    seq[++top] = u;
    first[u] = top;
```

```
    for (int e = fir[u], v = to[e]; e; e = nxt[e], v = to[e])
        if (v != father) {
            dep[v] = dep[u] + 1;
            st[v][0] = u;
            for (int k = 1; k <= 15; k++)
                st[v][k] = st[st[v][k - 1]][k - 1];
            dfs(v, u);
        }
    seq[++top] = u;
    last[u] = top;
}

inline int LCA(int a, int b) {
    if (dep[a] < dep[b])
        swap(a, b);

    for (int k = 15; k >= 0; k--)
        if (dep[st[a][k]] >= dep[b])
            a = st[a][k];

    if (a == b)
        return a;

    for (int k = 15; k >= 0; k--)
        if (st[a][k] != st[b][k]) {
            a = st[a][k];
            b = st[b][k];
        }

    return st[a][0];
}

inline void add(int x, int& res) {
    vis[x] ^= 1;

    if (vis[x]) {
        cnt[color[x]]++;
        if (cnt[color[x]] == 1)
            res++;
    } else {
        cnt[color[x]]--;
        if (cnt[color[x]] == 0)
            res--;
    }
}

int main() {

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++) {
        scanf("%d", color + i);
        nums.push_back(color[i]);
    }
    //离散化
```

```
sort(nums.begin(), nums.end());
nums.erase(unique(nums.begin(), nums.end()), nums.end());
for (int i = 1; i <= n; i++)
    color[i] = lower_bound(nums.begin(), nums.end(), color[i]) - nums.begin();

//读入树的边
int a, b;
for (int i = 1; i < n; i++) {
    scanf("%d%d", &a, &b);
    add_edge(a, b);
    add_edge(b, a);
}

//求欧拉序列并初始化 st 表
dep[1] = 1;
dfs(1, 0);

//读入询问并排序
for (int i = 1; i <= m; i++) {
    scanf("%d%d", &a, &b);
    if (first[a] > first[b])
        swap(a, b);
    int lca = LCA(a, b);
    if (lca == a) {
        query[i] = {i, first[a], first[b], 0};
    } else {
        query[i] = {i, last[a], first[b], lca};
    }
}

len = sqrt(top) + 1;
sort(query + 1, query + 1 + m);

int res = 0;
int i = 1, j = 0;
for (int k = 1; k <= m; k++) {
    int id = query[k].id, l = query[k].l, r = query[k].r, p = query[k].p;

    while (j < r)
        add(seq[++j], res);
    while (j > r)
        add(seq[j--], res);
    while (i > l)
        add(seq[--i], res);
    while (i < l)
        add(seq[i++], res);

    if (p)
        add(p, res);
    ans[id] = res;
    if (p)
        add(p, res);
}

for (int i = 1; i <= m; i++)
```

```
        printf("%d\n", ans[i]);

    return 0;
}
```

4.2.5 二次离线莫队

题目: Luogu P4887
查询区间 [l, r] 中满足 a[i] xor a[j] 的二进制表示中恰好有 k 个 1 的对数。

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <vector>

using namespace std;

typedef long long LL;

const int maxn = 100000 + 100;

int w[maxn];

struct Query {
    int id, l, r;
    LL res;
} query[maxn];

struct Range {
    int id, l, r, t;
};

vector<Range> range[maxn];

int n, m, k, len;

//f[i] 1 ~ i 中与 w[i + 1] 配对的个数
//g[x] 前 i 个数中，与 x 配对的个数
//f[i] = g[w[i + 1]]
int f[maxn], g[maxn];
vector<int> nums;

inline int readIn() {
    int x = 0;
    char c = getchar();
    while (c < '0' || c > '9')
        c = getchar();
    while (c >= '0' && c <= '9') {
        x = (x << 3) + (x << 1) + c - '0';
        c = getchar();
    }
    return x;
}
```

```cpp
int get(int x) {
    return x / len;
}

bool cmp(const Query& a, const Query& b) {
    int la = get(a.l);
    int lb = get(b.l);
    if (la != lb) return la < lb;
    return a.r < b.r;
}

int get_count(int x) { //x 二进制表示中 1 的个数
    int res = 0;
    while (x)
        res += (x & 1), x >>= 1;
    return res;
}

LL ans[maxn];

int main() {
    n = readIn();
    m = readIn();
    k = readIn();

    for (int i = 1; i <= n; i++)
        w[i] = readIn();

    for (int i = 1; i <= m; i++) {
        query[i].id = i;
        query[i].l = readIn();
        query[i].r = readIn();
    }

    len = sqrt(n) + 1;

    sort(query + 1, query + 1 + m, cmp);

    for (int i = 0; i < (1 << 14); i++)
        if (get_count(i) == k)
            nums.push_back(i);

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

        for (int t : nums)
            g[w[i] ^ t]++;

        f[i] = g[w[i + 1]];
    }

    for (int i = 1, L = 1, R = 0; i <= m; i++) {
        int l = query[i].l, r = query[i].r;

        if (R < r)
            range[L - 1].push_back({i, R + 1, r, -1});
```

```
        while (R < r)
            query[i].res += f[R], R++;

        if (R > r)
            range[L - 1].push_back({i, r + 1, R, 1});
        while (R > r)
            query[i].res -= f[R - 1], R--;
        if (L < l)
            range[R].push_back({i, L, l - 1, -1});
        while (L < l)
            query[i].res += f[L - 1] + !k, L++;

        if (L > l)
            range[R].push_back({i, l, L - 1, 1});
        while (L > l)
            query[i].res -= f[L - 2] + !k, L--;
    }

    memset(g, 0, sizeof(g));

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

        for (int t : nums)
            g[w[i] ^ t]++;

        for (Range& rg : range[i]) {
            int id = rg.id, l = rg.l, r = rg.r;
            for (int x = l; x <= r; x++)
                query[id].res += (LL)rg.t * g[w[x]];
        }
    }

    for (int i = 2; i <= m; i++)
        query[i].res += query[i - 1].res;

    for (int i = 1; i <= m; i++)
        ans[query[i].id] = query[i].res;

    for(int i = 1; i <= m; i++)
        printf("%lld\n", ans[i]);
    return 0;
}
```