# ICPC TEMPLATE

widsnoy

2024 年 5 月 24 日

## 目录

# 目录

# 1 动态规划

## 1.1 回退背包

$dp[i][j]$ 表示选择 $i$ 个物品，体积为 $j$ 的方案数 $f[i][j]$ 表示不考率其中某个物品，选择 $i$ 个物品，体积为 $j$ 的方案数 $dp[i][j] = f[i-1][j-w] + f[i][j] \Rightarrow f[i][j] = dp[i][j] - f[i-1][j-w]$

# 2 图论

## 2.1 最近公共祖先

```
// 倍增
int faz[N][20], dep[N];
void dfs(int u, int fa) {
    faz[u][0] = fa;
    dep[u] = dep[fa] + 1;
    for (int i = 1; i < 20; i++) faz[u][i] = faz[faz[u][i - 1]][i - 1];
    for (int v : G[u]) if (v != fa) {
        dfs(v, u);
    }
}
int LCA(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int d = dep[u] - dep[v];
    for (int i = 0; i < 20; i++) if ((d >> i) & 1) u = faz[u][i];
    if (v == u) return u;
    for (int i = 19; i >= 0; i--) if (faz[u][i] != faz[v][i])
        u = faz[u][i], v = faz[v][i];
    return faz[u][0];
}

//树剖
int dfc, dfn[N], rnk[N], siz[N], top[N], dep[N], son[N], faz[N];
void dfs1(int u, int fa) {
    dep[u] = dep[fa] + 1;
    siz[u] = 1;
    son[u] = -1;
    faz[u] = fa;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs1(v, u);
```

```cpp
        siz[u] += siz[v];
        if (son[u] == -1 || siz[son[u]] < siz[v]) son[u] = v;
    }
}
void dfs2(int u, int fa, int tp) {
    dfn[u] = ++dfc;
    rnk[dfc] = u;
    top[u] = tp;
    if (son[u] != -1) dfs2(son[u], u, tp);
    for (int v : G[u]) {
        if (v == fa || v == son[u]) continue;
        dfs2(v, u, v);
    }
}
int LCA(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]])
            u = faz[top[u]];
        else
            v = faz[top[v]];
    }
    return dep[u] > dep[v] ? v : u;
}

// O(1) query

int dfn[N], faz[N], dep[N], rnk[N], dfc, st[N][20];
void dfs(int u, int fa) {
    dfn[u] = ++dfc; faz[u] = fa; dep[u] = dep[fa] + 1; rnk[dfc] = u;
    for (auto [v, w] : G[u]) if (v != fa) dfs(v, u);
}
int LCA(int u, int v) {
    if (u == v) return u;
    if (dfn[u] > dfn[v]) swap(u, v);
    int l = dfn[u] + 1, r = dfn[v];
    int k = __lg(r - l + 1);
    return dep[st[l][k]] < dep[st[r - (1 << k) + 1][k]] ? faz[st[l][k]] : faz[
       st[r - (1 << k) + 1][k]];
}

int main() {
    dfs(1, 0);
    dep[0] = n + 1;
    for (int i = 1; i <= n; i++) st[i][0] = rnk[i];
```

```
    for (int j = 1; j < 20; j++) {
        for (int i = 1; i <= n; i++) {
            st[i][j] = dep[st[i][j - 1]] <= dep[st[min(n, i + (1 << (j - 1)))
                ][j - 1]] ? st[i][j - 1] : st[min(n, i + (1 << (j - 1)))][j -
                1];
        }
    }
}
```

### 2.2 2-SAT 前缀优化建图

1. 当前点选择说明之前的前缀都未选择

之前的前缀选择说明当前点位被选择

2. 之前的前缀选择说明当前前缀被选择

当前前缀未选择说明之前前缀未选择

3. 当前点选择说明当前前缀选择

当前前缀未选择说明当前点未选择

### 2.3 Kruskal 重构树的性质

不难发现，原图中两个点之间的所有简单路径上最大边权的最小值 = 最小生成树上两个点之间的简单路径上的最大值 = Kruskal 重构树上两点之间的 LCA 的权值。

也就是说，到点 $x$ 的简单路径上最大边权的最小值 $\le val$ 的所有点 $y$ 均在 Kruskal 重构树上的某一棵子树内，且恰好为该子树的所有叶子节点。

我们在 Kruskal 重构树上找到 $x$ 到根的路径上权值 $\le val$ 的最浅的节点。显然这就是所有满足条件的节点所在的子树的根节点。

如果需要求原图中两个点之间的所有简单路径上最小边权的最大值，则在跑 Kruskal 的过程中按边权大到小的顺序加边。

### 2.4 广义圆方树

```
int stk[N], n, m, top, cnt, low[N], dfn[N], dfc;
bool vis[N];
vector<int> G[N], T[N];
```

```cpp

void tarjan(int u) {
    stk[++top] = u;
    low[u] = dfn[u] = ++dfc;
    for (int v : G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == dfn[u]) {
                cnt++;
                for (int x = 0; x != v; --top) {
                    x = stk[top];
                    T[cnt].push_back(x);
                    T[x].push_back(cnt);
                    val[cnt]++;
                }
                T[cnt].push_back(u);
                T[u].push_back(cnt);
                val[cnt]++;
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}
int main() {
    cnt = n;
    for (int i = 1; i <= n; i++) if (!dfn[i]) {
        tarjan(i);
        --top;
    }
}
```

### 2.5 点分树

需要注意，点分树上的路径与原来的树完全没有关系。

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

const int mod = 998244353;
const int N = 2e5 + 5;

int n, q, val[N], rt, siz[N], SUM, mx[N];
int faz[N];
```

```cpp
vector<int> T[N];
bool vis[N];

// 假设树剖已经写好

void getrt(int u, int fa) {
    siz[u] = 1;
    mx[u] = 0;
    for (int v : t.G[u]) if (v != fa && !vis[v]) {
        getrt(v, u);
        siz[u] += siz[v];
        mx[u] = max(mx[u], siz[v]);
    }
    mx[u] = max(mx[u], SUM - siz[u]);
    if (mx[u] < mx[rt]) rt = u;
}
void Dfs(int u, int fa) {
    vis[u] = 1;
    for (int v : t.G[u]) if (v != fa && !vis[v]) {
        rt = 0;
        SUM = siz[v];
        getrt(v, rt);
        T[u].push_back(rt);
        faz[rt] = u;
        Dfs(rt, rt);
    }
}

// 假设动态开点权值线段树已经写好

void dfs(int u, const int now) { // 预处理
    add(rt1[now], 0, n, dist(faz[now], u), val[u]);
    add(rt0[now], 0, n, dist(u, now), val[u]);
    for (int v : T[u]) dfs(v, now);
}
void upd(int u, int d) {
    int x = u;
    while (true) {
        add(rt1[u], 0, n, dist(x, faz[u]), d);
        add(rt0[u], 0, n, dist(x, u), d);
        if (u == root) break;
        u = faz[u];
    }
}
```

```cpp
int qry(int u, int k) {
    int ans = query(rt0[u], 0, n, 0, k), x = u;
    while (u != root) {
        if (k - dist(faz[u], x) >= 0) ans += query(rt0[faz[u]], 0, n, 0, k -
           dist(faz[u], x)) - query(rt1[u], 0, n, 0, k - dist(faz[u], x));
        u = faz[u];
    }
    return ans;
}
```

## 2.6 二分图

**最大匹配**

```cpp
int mch[maxn], vis[maxn];
std::vector<int> e[maxn];
bool dfs(const int u, const int tag) {
    for (auto v : e[u]) {
        if (vis[v] == tag) continue;
        vis[v] = tag;
        if (!mch[v] || dfs(mch[v], tag)) return mch[v] = u, 1;
    }
    return 0;
}
int main() {
    int ans = 0;
    for (int i = 1; i <= n; ++i) if (dfs(i, i)) ++ans;
}
```

## 2.7 有向图最小路径覆盖问题

```cpp
int n, m;
bitset<N> f[N];
int vis[N], mch[N];

bool dfs(int u, int dfc) {
    for (int v = 1; v <= n; v++) if (v != u && vis[v] != dfc && f[u][v]) {
        vis[v] = dfc;
        if (!mch[v] || dfs(mch[v], dfc)) return mch[v] = u, 1;
    }
    return 0;
}

void solve() {
```

```cpp
    memset(vis, 0, sizeof vis);
    memset(mch, 0, sizeof mch);
    for (int i = 1; i <= n; i++) f[i].reset();
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        f[u].set(v);
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) if (f[i][k]) f[i] |= f[k];
    }
    int res = n;
    for (int i = 1; i <= n; i++) res -= dfs(i, i);
    printf("%d\n", res);
}
```

## 2.8 网络流

### 2.8.1 Dinic 最大流

注意每次清空数组的范围是 $s$ 到 $t$.

```cpp
int head[N], cur[N], ecnt, d[N];
struct Edge {
    int nxt, v, flow, cap;
}e[];
void add_edge(int u, int v, int flow, int cap) {
    e[ecnt] = {head[u], v, flow, cap}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0}; head[v] = ecnt++;
}
bool bfs() {
    memset(vis, 0, sizeof vis);
    std::queue<int> q;
    q.push(s);
    vis[s] = 1;
    d[s] = 0;
    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 (vis[v] || e[i].flow >= e[i].cap) continue;
            d[v] = d[u] + 1;
            vis[v] = 1;
            q.push(v);
```

```
        }
    }
    return vis[t];
}
int dfs(int u, int a) {
    if (u == t || !a) return a;
    int flow = 0, f;
    for (int& i = cur[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (d[u] + 1 == d[v] && (f = dfs(v, std::min(a, e[i].cap - e[i].flow))
            ) > 0) {
            e[i].flow += f;
            e[i ^ 1].flow -= f;
            flow += f;
            a -= f;
            if (!a) break;
        }
    }
    return flow;
}
```

#### 2.8.2 最小费用最大流

```
const int inf = 1e9;
int head[N], cur[N], ecnt, dis[N], s, t, n, m, mincost;
bool vis[N];
struct Edge {
    int nxt, v, flow, cap, w;
}e[100002];
void add_edge(int u, int v, int flow, int cap, int w) {
    e[ecnt] = {head[u], v, flow, cap, w}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0, -w}; head[v] = ecnt++;
}
bool spfa(int s, int t) {
    std::fill(vis + s, vis + t + 1, 0);
    std::fill(dis + s, dis + t + 1, inf);
    std::queue<int> q;
    q.push(s);
    dis[s] = 0;
    vis[s] = 1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
```

```
        for (int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if (e[i].flow < e[i].cap && dis[u] + e[i].w < dis[v]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) vis[v] = 1, q.push(v);
            }
        }
    }
    return dis[t] != inf;
}
int dfs(int u, int a) {
    if (vis[u]) return 0;
    if (u == t || !a) return a;
    vis[u] = 1;
    int flow = 0, f;
    for (int& i = cur[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (dis[u] + e[i].w == dis[v] && (f = dfs(v, std::min(a, e[i].cap - e[
            i].flow))) > 0) {
            e[i].flow += f;
            e[i ^ 1].flow -= f;
            flow += f;
            mincost += e[i].w * f;
            a -= f;
            if (!a) break;
        }
    }
    vis[u] = 0;
    return flow;
}
```

#### 2.8.3 最大闭权子图

正权点向 S 连边，负权点向 T 连边。边权为点权的绝对值。原图的边容量设为 INF 。

则最大收益为 $\sum_{v>0} v - mincost$

在最大闭权子图中的点是残量网络中 S 能到达的点。

### 2.9 树哈希

```
const ull mask = chrono::steady_clock::now().time_since_epoch().count();

ull shift(ull x) {
```

```cpp
    x ^= mask;
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    x ^= mask;
    return x;
}
int n;
ull H[N];
vector<int> G[N];
set<ull> s;

void dfs(int u, int fa) {
    H[u] = 1;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs(v, u);
        H[u] += shift(H[v]);
    }
    s.emplace(H[u]);
}
```

### 2.10 强联通分量

```cpp
int n, dfc, dfn[N], low[N], stk[N], top, idx[N], in_stk[N], scc_cnt;
vector<int> G[N];

void tarjan(int u) {
    low[u] = dfn[u] = ++dfc;
    stk[++top] = u;
    in_stk[u] = 1;
    for (int v : G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in_stk[v]) low[u] = min(dfn[v], low[u]);
    }
    if (low[u] == dfn[u]) {
        int x;
        scc_cnt++;
        do {
            x = stk[top--];
            idx[x] = scc_cnt;
            in_stk[x] = 0;
```

```
        } while (x != u);
    }
}

// 多测清空
dfc = scc_cnt = top = 0;
for (int i = 1; i <= tot; i++) low[i] = dfn[i] = idx[i] = in_stk[i] = 0;
```

## 2.11 割点和桥

```
int dfn[N], low[N], dfs_clock;
bool iscut[N], vis[N];
void dfs(int u, int fa) {
    dfn[u] = low[u] = ++dfs_clock;
    vis[u] = 1;
    int child = 0;
    for (int v : e[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            child++;
            if (low[v] >= dfn[u]) iscut[u] = 1;
        } else if (dfn[u] > dfn[v] && v != fa) low[u] = min(low[u], dfn[v]);
        if (fa == 0 && child == 1) iscut[u] = 0;
    }
}
```

## 2.12 点双联通分量

```
int bccno[N], bcc_cnt, siz_e[N], siz_p[N], dfs_clock, low[N], dfn[N], top;
pair<int, int> stk[N];
void dfs(int u, int fa) {
    low[u] = dfn[u] = ++dfs_clock;
    for(int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if(v == fa) continue;
        if(!dfn[v]) {
            stk[++top] = make_pair(u, v);
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) {
                bcc_cnt++;
```

```
                while(true) {
                    int x = stk[top].first, y = stk[top].second;
                    top--;
                    siz_e[bcc_cnt]++;
                    if(bccno[x] != bcc_cnt) {bccno[x] = bcc_cnt; siz_p[bcc_cnt
                        ]++;}
                    if(bccno[y] != bcc_cnt) {bccno[y] = bcc_cnt; siz_p[bcc_cnt
                        ]++;}
                    if(x == u && y == v) break;
                }
            }
        } else if(dfn[v] < dfn[u]) {stk[++top] = make_pair(u, v); low[u] = min
            (low[u], dfn[v]);}
    }
}
```

## 2.13 边双联通分量

```
const int N = 5000 + 5;
int n, m, stk[N], top, ccno, sc[N];
int dfn[N], dfc, low[N];
int mp[N][N];
int in[N];
int head[N], ecnt;
struct Edge {
    int nxt, v;
} e[N << 2];
void add_edge(int u, int v) {
    e[ecnt] = {head[u], v}; head[u] = ecnt++;
    e[ecnt] = {head[v], u}; head[v] = ecnt++;
}
void dfs(int u, int from) {
    stk[++top] = u;
    low[u] = dfn[u] = ++dfc;
    for (int i = head[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (!dfn[v]) {
            dfs(v, i);
            low[u] = min(low[u], low[v]);
        } else if ((i ^ 1) != from) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        ccno++;
        int x;
```

```
        while (true) {
            x = stk[top--];
            sc[x] = ccno;
            if (x == u) break;
        }
    }
}

void solve() {
    memset(head, -1, sizeof head);
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        add_edge(u, v);
    }
    for (int i = 1; i <= n; i++) if (!dfn[i]) dfs(i, i);
    for (int i = 1; i <= n; i++) {
        for (int k = head[i]; k != -1; k = e[k].nxt) {
            int j = e[k].v;
            if (sc[i] != sc[j]) mp[sc[i]][sc[j]] = 1;
        }
    }

    for (int i = 1; i <= ccno; i++) {
        for (int j = 1; j <= ccno; j++) if (mp[i][j]) in[j]++;
    }
    int cnt = 0;
    for (int i = 1; i <= ccno; i++) if (in[i] == 1) cnt++;
    printf("%d\n", (cnt + 1) / 2);
}
```

### 2.14 2-SAT

$2 * u$ 代表不选择，$2 * u + 1$ 代表选择。

也可以求强连通分量。

如果对于一个 *x* ‘sccno‘比它的反状态 *x*1 的 ‘sccno‘ 要小，那么我们用 *x* 这个状态当做答案，否则用它的反状态当做答案。

```
vector<int> G[N * 2];
bool mark[N * 2];
int stk[N], top;
```

```
void build_G() {
    for (int i = 1; i <= n; i++) {
        int u, v;
        G[2 * u + 1].push_back(2 * v);
        G[2 * v + 1].push_back(2 * u);
    }
}
bool dfs(int u) {
    if (mark[u ^ 1]) return false;
    if (mark[u]) return true;
    mark[u] = 1;
    stk[++top] = u;
    for (int v : G[u]) {
        if (!dfs(v)) return false;
    }
    return true;
}
bool 2_sat() {
    for (int i = 1; i <= n; i++) {
        if (!mark[i * 2] && !mark[i * 2 + 1]) {
            top = 0;
            if (!dfs(2 * i)) {
                while (top) mark[stk[top--]] = 0;
                if (!dfs(2 * i + 1)) return 0;
            }
        }
    }
    return 1;
}
```

# 3 数据结构

## 3.1 Splay

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

#define rank abcdefg
const int mod = 998244353;
const int N = 1e5 + 5;

int tot, fa[N], tr[N][2], sz[N], cnt[N], val[N], rt;

```

```
void maintain(int x) {
    sz[x] = sz[tr[x][0]] + sz[tr[x][1]] + cnt[x];
}
int getdir(int x) {
    return tr[fa[x]][1] == x;
}
void clear(int x) {
    fa[x] = sz[x] = cnt[x] = tr[x][0] = tr[x][1] = val[x] = 0;
}
int create(int v) {
    ++tot;
    val[tot] = v;
    sz[tot] = cnt[tot] = 1;
    return tot;
}
void rotate(int x) {
    if (x == rt) return;
    int y = fa[x], z = fa[y], d = getdir(x);
    tr[y][d] = tr[x][d ^ 1];
    if (tr[x][d ^ 1]) fa[tr[x][d ^ 1]] = y;
    fa[y] = x;
    tr[x][d ^ 1] = y;
    fa[x] = z;
    if (z) tr[z][y == tr[z][1]] = x;
    maintain(y);
    maintain(x);
}
void splay(int x) {
    for (int f = fa[x]; f = fa[x], f; rotate(x)) {
        if (fa[f]) rotate(getdir(f) == getdir(x) ? f : x);
    }
    rt = x;
}
void insert(int v) {
    if (!rt) {
        rt = create(v);
        return;
    }
    int u = rt, f = 0;
    while (true) {
        if (val[u] == v) {
            cnt[u]++;
            maintain(u);
            maintain(f);
```

```
            splay(u);
            return;
        }
        f = u, u = tr[u][v > val[u]];
        if (u == 0) {
            int id;
            fa[id = create(v)] = f;
            tr[f][v > val[f]] = id;
            maintain(f);
            splay(id);
            return;
        }
    }
}

int rank(int v) {
    int rk = 0;
    int u = rt;
    while (u) {
        if (val[u] == v) {
            rk += sz[tr[u][0]];
            splay(u);
            return rk + 1;
        }
        if (v < val[u]) {
            u = tr[u][0];
        } else {
            rk += sz[tr[u][0]] + cnt[u];
            u = tr[u][1];
        }
    }
    return -1;
}

int kth(int x) {
    int u = rt;
    while (u) {
        if (sz[tr[u][0]] + cnt[u] >= x && sz[tr[u][0]] < x) return val[u];
        if (x <= sz[tr[u][0]]) {
            u = tr[u][0];
        } else {
            x -= sz[tr[u][0]] + cnt[u];
            u = tr[u][1];
        }
```

```
    }
    return u ? val[u] : -1;
}
int pre() {
    int u = tr[rt][0];
    if (!u) return val[rt];
    while (true) {
        if (tr[u][1] == 0) return splay(u), val[u];
        u = tr[u][1];
    }
    return 233;
}
int suf() {
    int u = tr[rt][1];
    if (!u) return val[rt];
    while (true) {
        if (tr[u][0] == 0) return splay(u), val[u];
        u = tr[u][0];
    }
    return 233;
}
void del(int v) {
    if (rank(v) == -1) return;
    if (cnt[rt] > 1) {
        cnt[rt]--;
        return;
    }
    if (!tr[rt][1] && !tr[rt][0]) {
        clear(rt), rt = 0;
    } else if (!tr[rt][0]) {
        int x = rt;
        rt = tr[x][1];
        fa[rt] = 0;
        clear(x);
    } else if (!tr[rt][1]) {
        int x = rt;
        rt = tr[x][0];
        fa[rt] = 0;
        clear(x);
    } else {
        int cur = rt, y = tr[cur][1];
        pre();
        tr[rt][1] = y;
        fa[y] = rt;
```

```
        clear(cur);
        maintain(rt);
    }
}

int main() {
    int n, opt, x;

    for (scanf("%d", &n); n; --n) {
        scanf("%d%d", &opt, &x);

        if (opt == 1)
            insert(x);
        else if (opt == 2)
            del(x);
        else if (opt == 3)
            printf("%d\n", rank(x));
        else if (opt == 4)
            printf("%d\n", kth(x));
        else if (opt == 5)
            insert(x), printf("%d\n", pre()), del(x);
        else
            insert(x), printf("%d\n", suf()), del(x);
    }

    return 0;
}
```

### 3.2 李超线段树

```
struct Line {
    ll k, b;
} lin[N];
int lcnt;
int add_line(ll k, ll b) {
    lin[++lcnt] = {k, b};
    return lcnt;
}
struct node {
    int ls, rs, u;
} tr[N << 2];
int tot;
ll calc(int u, ll x) {
    return lin[u].k * x + lin[u].b;
```

```cpp
}
bool cmp(int u, int v, ll x) {
    return calc(u, x) <= calc(v, x); // 如果要求最大值，只需要修改为大于等于
}
void pushdown(int &p, int l, int r, int v) {
    if (!p) p = ++tot;
    if (l == r) return;
    int mid = (l + r) >> 1;
    int &u = tr[p].u, b = cmp(v, u, mid);
    if (b) swap(u, v);
    int bl = cmp(v, u, l), br = cmp(v, u, r);
    if (bl) pushdown(tr[p].ls, l, mid, v);
    if (br) pushdown(tr[p].rs, mid + 1, r, v);
}
void update(int &p, int l, int r, int L, int R, int v) {
    if (l > R || r < L) return;
    if (!p) p = ++tot;
    int mid = (l + r) >> 1;
    if (l >= L && r <= R) return pushdown(p, l, r, v), void();
    update(tr[p].ls, l, mid, L, R, v);
    update(tr[p].rs, mid + 1, r, L, R, v);
}
ll query(int p, int l, int r, ll pos) {
    if (!p) return 1e16;
    ll res = calc(tr[p].u, pos);
    int mid = (l + r) >> 1;
    if (l == r) return res;
    if (pos <= mid) {
        res = min(res, query(tr[p].ls, l, mid, pos));
    } else res = min(res, query(tr[p].rs, mid + 1, r, pos));
    return res;
}

int main() {
    lin[0].b = 1e16;
    return 0;
}
```

### 3.3 Link Cut Tree

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

const int N = 1e5 + 5;
```

```
int n, m, ch[N][2], f[N], s[N], r[N], v[N];

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

bool noroot(int x) {
    return ch[f[x]][1] == x || ch[f[x]][0] == x;
}
void pushup(int x) {
    s[x] = s[lc] ^ s[rc] ^ v[x];
}
void pushr(int x) {
    swap(lc, rc);
    r[x] ^= 1;
}
void pushdown(int x) {
    if (r[x]) {
        if (lc) pushr(lc);
        if (rc) pushr(rc);
        r[x] = 0;
    }
}
void rotate(int x) {
    int y = f[x], z = f[y], k = (ch[y][1] == x), w = ch[x][k ^ 1];
    if (noroot(y)) ch[z][y == ch[z][1]] = x;
    ch[x][k ^ 1] = y;
    ch[y][k] = w;
    f[w] = y;
    f[y] = x;
    f[x] = z;
    pushup(y), pushup(x);
}
void update(int x) {
    if (noroot(x)) update(f[x]);
    pushdown(x);
}
bool get(int x) {
    return ch[f[x]][1] == x;
}
void splay(int x) {
    update(x);
    for (int fa; fa = f[x], noroot(x); rotate(x)) {
        if (noroot(fa)) rotate(get(x) == get(fa) ? fa : x);
    }
```

```
    pushup(x);
}
void access(int x) {
    int p;
    for (p = 0; x; p = x, x = f[x]) {
        splay(x),ch[x][1] = p, pushup(x);
    }
}
void makeroot(int x) {
    access(x); splay(x);
    pushr(x);
}
int findroot(int x) {
    access(x);
    splay(x);
    while (lc) pushdown(x), x = lc;
    splay(x);
    return x;
}
void split(int x, int y) {
    makeroot(x);
    access(y);splay(y);
}
void link(int x, int y) {
    makeroot(x);
    if (findroot(y) != x) f[x] = y;
}
void cut(int x, int y) {
    makeroot(x);
    if (findroot(y) == x && f[y] == x && !ch[y][0]) {
        f[y] = ch[x][1] = 0;
        pushup(x);
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &v[i]);
    while (m--) {
        int opt, x, y;
        scanf("%d %d %d", &opt, &x, &y);
        if (opt == 0) split(x, y), printf("%d\n", s[y]);
        if (opt == 1) link(x, y);
        if (opt == 2) cut(x, y);
```

```
        if (opt == 3) splay(x), v[x] = y, pushup(x);
    }
    return 0;
}
```

### 3.4 兔队线段树

求有多少个严格前缀最大值。

线段树保存每个区间为子问题时右部分的答案 res（可以不需要信息可减），和区间的最大值 mx。

calc 考虑一段区间之前有 x 大的数时，区间此时前缀最大数的树目。

1. $x \geq val[lson]$，$ans = calc(rson)$

2. $x < val[lson]$，$ans = calc(lson) + res[p]$

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

const int N = 1e5 + 5;
#define lson (p << 1)
#define rson ((p << 1) | 1)
#define mid ((l + r) >> 1)
int n, m;
struct node {
    int s, a, b;
} tr[N << 2];
bool cmp(int a, int b, int c, int d) {
    if (d == 0 && b == 0) return 0;
    if (d == 0 && a == 0) return 0;
    if (d == 0) return 1;
    return a * 1ll * d > c * 1ll * b;
}
int calc(int p, int l, int r, int c, int d) {
    if (l == r)
        return cmp(tr[p].a, tr[p].b, c, d);
    if (cmp(tr[lson].a, tr[lson].b, c, d)) {
        return calc(lson, l, mid, c, d) + tr[p].s;
    }
    return calc(rson, mid + 1, r, c, d);
}
void modify(int p, int l, int r, int pos, int v) {
    if (l == r) {
```

```
        tr[p] = {0, v, pos};
        return;
    }
    if (pos <= mid) modify(lson, l, mid, pos, v);
    else modify(rson, mid + 1, r, pos, v);
    if (cmp(tr[lson].a, tr[lson].b, tr[rson].a, tr[rson].b)) {
        tr[p] = tr[lson];
    } else tr[p] = tr[rson];
    tr[p].s = calc(rson, mid + 1, r, tr[lson].a, tr[lson].b);
}

int main() {
    scanf("%d %d", &n, &m);
    while (m--) {
        int x, y;
        scanf("%d %d", &x, &y);
        modify(1, 1, n, x, y);
        printf("%d\n", calc(1, 1, n, 0, 0));
    }
    return 0;
}
```

### 3.5 线段树分治

有一个 $n$ 个节点的图。

在 $k$ 时间内有 $m$ 条边会出现后消失。

要求出每一时间段内这个图是否是二分图。

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

const int N = 4e5 + 5, M = 4e6;
int tot, n, m, t, fa[N], d[N], u[N], v[N];
int stk[N], top, head[N];
bool fl[N];

struct E {
    int nxt, id;
} e[M];

int find(int x) {
    while (fa[x]) x = fa[x];
    return x;
```

```
}
void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x == y) return;
    if (d[x] > d[y]) swap(x, y);
    fa[x] = y;
    stk[++top] = x;
    d[y] += fl[top] = (d[x] == d[y]);
}

void upd(int p, int l, int r, int L, int R, const int &i) {
    if (l == L && r == R) {
        e[++tot] = (E){head[p], i};
        head[p] = tot;
        return ;
    }
    int mid = (l + r) >> 1;
    if (R <= mid) upd(p << 1, l, mid, L, R, i);
    else if (L > mid) upd(p << 1 | 1, mid + 1, r, L, R, i);
    else upd(p << 1, l, mid, L, mid, i), upd(p << 1 | 1, mid + 1, r, mid + 1, R, i);
}
void solve(int p, int l, int r) {
    int lst = top, mid = (l + r) >> 1;;
    for (int i = head[p]; i; i = e[i].nxt) {
        int x = u[e[i].id], y = v[e[i].id];
        if (find(x) == find(y)) {
            for (int i = l; i <= r; i++) puts("No");
            goto che;
        }
        merge(x + n, y), merge(x, y + n);
    }
    if (l == r)
        puts("Yes");
    else {
        solve(p << 1, l, mid);
        solve(p << 1 | 1, mid + 1, r);
    }
    che : for (; top > lst; top--) d[fa[stk[top]]] -= fl[top], fa[stk[top]] = 0;
}


int main() {
```

```
    //freopen("2.in", "r", stdin);
    scanf("%d %d %d", &n, &m, &t);
    for (int i = 1; i <= m; i++) {
        int l, r;
        scanf("%d %d %d %d", &u[i], &v[i], &l, &r);
        if (l == r) continue;
        upd(1, 1, t, l + 1, r, i);
    }
    solve(1, 0, t - 1);
    return 0;
}
```

# 4 字符串

## 4.1 哈希

### 4.1.1 最长回文子串

通过哈希同样可以 $O(n)$ 解决这个问题，具体方法就是记 $R_i$ 表示以 $i$ 作为结尾的最长回文的长度，那么答案就是 $\max_{i=1}^{n} R_i$。考虑到 $R_i \le R_{i-1} + 2$，因此我们只需要暴力从 $R_{i-1} + 2$ 开始递减，直到找到第一个回文即可。记变量 $z$ 表示当前枚举的 $R_i$，初始时为 $0$，则 $z$ 在每次 $i$ 增大的时候都会增大 $2$，之后每次暴力循环都会减少 $1$，故暴力循环最多发生 $2n$ 次，总的时间复杂度为 $O(n)$。

## 4.2 字典树

## 4.3 维护异或和

```
const int N = 526010, MX = 22;
int ch[N * MX][2], tot, rt[N], w[N * MX], xorv[N * MX], val[N];
ll ans;

void pushup(int u) {
    w[u] = xorv[u] = 0;
    if (ch[u][0]) {
        w[u] += w[ch[u][0]];
        xorv[u] ^= (xorv[ch[u][0]] << 1);
    }
    if (ch[u][1]) {
        w[u] += w[ch[u][1]];
        xorv[u] ^= (xorv[ch[u][1]] << 1) | (w[ch[u][1]] & 1);
```

```cpp
    }
    w[u] &= 1;
}
void insert(int &o, ll ux, int dep) {
    if (!o) o = ++tot;
    if (dep > MX) return (void)(w[o]++);
    insert(ch[o][ux & 1], ux >> 1, dep + 1);
    pushup(o);
}
void addall(int o) {
    swap(ch[o][0], ch[o][1]);
    if (ch[o][0]) addall(ch[o][0]);
    pushup(o);
}
int merge(int a, int b) {
    if (!b || !a) return a + b;
    xorv[a] ^= xorv[b];
    w[a] += w[b];
    ch[a][0] = merge(ch[a][0], ch[b][0]);
    ch[a][1] = merge(ch[a][1], ch[b][1]);
    return a;
}

vector<int> G[N];
int read() {
    int w = 0, f = 1; char ch = getchar();
    while (ch > '9' || ch < '0') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        w = w * 10 + ch - 48;
        ch = getchar();
    }
    return w * f;
}

void dfs(int u) {
    for (auto v : G[u]) {
        dfs(v);
        rt[u] = merge(rt[u], rt[v]);
    }
    addall(rt[u]);
    insert(rt[u], val[u], 0);
```

```
    ans += (ll)xorv[rt[u]];
}

int main() {
    int n = read();
    for (int i = 1; i <= n; i++) val[i] = read();
    for (int i = 2; i <= n; i++) G[read()].push_back(i);
    dfs(1);
    printf("%lld\n", ans);
    return 0;
}
```

## 4.4 KMP

```
int n = strlen(s + 1);
for (int i = 2; i <= n; i++) {
    int j = k[i - 1];
    while (j != 0 && s[i] != s[j + 1]) j = k[j];
    if (s[i] == s[j + 1]) k[i] = j + 1;
    else k[i] = 0;
}
```

### 4.4.1 字符串最小周期

设 border 长度为 r

则 $s[i] = s[n - r + i]$

$|T| = n - r$

### 4.4.2 每个前缀的出现次数

1. 统计每个前缀在自身的出现次数

```
vector<int> ans(n + 1);
for (int i = 1; i <= n; i++) ans[k[i]]++;
for (int i = n; i >= 1; i--) ans[k[i]] += ans[i];
for (int i = 1; i <= n; i++) ans[i]++;
```

2. 统计每个前缀在其他串的出现次数

我们应用来自 Knuth-Morris-Pratt 的技巧：构造一个字符串 $s + \# + t$ 并计算其前缀函数。与第一个问题唯一的不同之处在于，我们只关心与字符串 $t$ 相关的前缀函数值，即 $i \geq n + 1$ 的 $\pi[i]$。

有了这些值之后，我们可以同样应用在第一个问题中的算法来解决该问题。

#### 4.4.3 一个字符串中本质不同子串的数目

给定一个长度为 $n$ 的字符串 $s$，我们希望计算其本质不同子串的数目。

我们将迭代的解决该问题。换句话说，在知道了当前的本质不同子串的数目的情况下，我们要找出一种在 $s$ 末尾添加一个字符后重新计算该数目的方法。

令 $k$ 为当前 $s$ 的本质不同子串数量。我们添加一个新的字符 $c$ 至 $s$。显然，会有一些新的子串以字符 $c$ 结尾。我们希望对这些以该字符结尾且我们之前未曾遇到的子串计数。

构造字符串 $t = s + c$ 并将其反转得到字符串 $t^{\sim}$。现在我们的任务变为计算有多少 $t^{\sim}$ 的前缀未在 $t^{\sim}$ 的其余任何地方出现。如果我们计算了 $t^{\sim}$ 的前缀函数最大值 $\pi_{\max}$，那么最长的出现在 $s$ 中的前缀其长度为 $\pi_{\max}$。自然的，所有更短的前缀也出现了。

因此，当添加了一个新字符后新出现的子串数目为 $|s| + 1 - \pi_{\max}$。

所以对于每个添加的字符，我们可以在 $O(n)$ 的时间内计算新子串的数目，故最终复杂度为 $O(n^2)$。

值得注意的是，我们也可以重新计算在头部添加一个字符，或者从尾或者头移除一个字符时的本质不同子串数目。

### 4.5 AC 自动机

```cpp
namespace AC {
    int ch[N][26], tot, fail[N], e[N];
    void insert(const char *s) {
        int u = 0, n = strlen(s + 1);
        for (int i = 1; i <= n; i++) {
            if (!ch[u][s[i] - 'a']) ch[u][s[i] - 'a'] = ++tot;
            u = ch[u][s[i] - 'a'];
        }
        e[u] += 1;
    }
    void build() {
        queue<int> q;
        for (int i = 0; i <= 25; i++) if (ch[0][i]) q.push(ch[0][i]);
        while (!q.empty()) {
            int now = q.front(); q.pop();
```

```
            for (int i = 0; i < 26; i++) {
                if (ch[now][i]) fail[ch[now][i]] = ch[fail[now]][i], q.push(ch
                    [now][i]);
                else ch[now][i] = ch[fail[now]][i];
            }
        }
    }
    int query(const char *s) {
        int u = 0, n = strlen(s + 1), res = 0;
        for (int i = 1; i <= n; i++){
            u = ch[u][s[i] - 'a'];
            for (int j = u; j && e[j] != -1; j = fail[j]) {
                res += e[j];
                e[j] = -1;
            }
        }
        return res;
    }
}
```

### 4.6 后缀数组

```
const int N = 2e5 + 5;
int sa[N << 1], ork[N << 1], rk[N << 1], cnt[N], id[N << 1], M, n;
char s[N];

int main() {
    scanf("%s", s + 1);
    n = strlen(s + 1);
    for (int i = n + 1; i <= (n << 1); i++) s[i] = s[i - n], M = max(M, (int)s
        [i]);
    n <<= 1;
    for (int i = 1; i <= n; i++) if ((int)(s[i]) > M) M = (int)(s[i]);
    for (int i = 1; i <= n; i++) cnt[rk[i] = s[i]]++;
    for (int i = 0; i <= M; i++) cnt[i] += cnt[i - 1];
    for (int i = n; i; i--) sa[cnt[rk[i]]--] = i;
    for (int w = 1, p; w < n; w <<= 1, M = p) {
        p = 0;
        for (int i = n; i > n - w; i--) id[++p] = i;
        for (int i = 1; i <= n; i++) if (sa[i] > w) id[++p] = sa[i] - w;
        for (int i = 0; i <= M; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) cnt[rk[i]]++;
        for (int i = 1; i <= M; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i; i--) sa[cnt[rk[id[i]]]--] = id[i];
```

```
        p = 0;
        for (int i = 0; i <= n; i++) ork[i] = rk[i];
        for (int i = 1; i <= n; i++) {
            if (ork[sa[i]] == ork[sa[i - 1]] && ork[sa[i] + w] == ork[sa[i -
               1] + w]) rk[sa[i]] = p;
            else rk[sa[i]] = ++p;
        }
        if (p == n) break;
    }
    for (int i = 1, k = 0; i <= n; i++) {
        if (rk[i] == 1) continue;
        if (k) k--;
        while (s[i + k] == s[sa[rk[i] - 1] + k]) k++;
        h[rk[i]] = k;
    }
    return 0;
}
```

## 4.7 Manacher

对于第 $i$ 个字符为对称轴:

1. 如果回文串长为奇数, $d[2*i]/2$ 是半径加上自己的长度

2. 如果长为偶数, $d[2*i-1]/2$ 是半径的长度, 方向向右.

```
int n, d[N * 2];
char s[N];

for (int i = 1; i <= n; i++) t[i * 2] = s[i], t[i * 2 - 1] = '#';
t[n * 2 + 1] = '#';
m = n * 2 + 1;
for (int i = 1, l = 0, r = 0; i <= m; i++) {
    int k = i <= r ? min(d[r - i + l], r - i + 1) : 1;
    while (i + k <= m && i - k >= 1 && t[i + k] == t[i - k]) k++;
    d[i] = k--;
    if (i + k > r) r = i + k, l = i - k;
}
```

## 4.8 Z 函数

$z[i] = lcp(suf_1, suf_i)$

```
 for (int i = 2, l = 0, r = 0; i <= n; i++) {
```

```
    if (r >= i && r - i + 1 > z[i - l + 1]) {
        z[i] = z[i - l + 1];
    } else {
        z[i] = max(0, r - i + 1);
        while (z[i] < n - i + 1 && s[z[i] + 1] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
}
```

# 5 数学

## 5.1 基本算法

```
int fpow(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = a * 1ll * a % mod) if (b & 1) res = res * 1ll * a %
        mod;
    return res;
}
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b) {
        ll d = exgcd(b, a % b, y, x);
        return y -= a / b * x, d;
    } else return x = 1, y = 0, a;
}
int getinv(int v) {
    return fpow(v, mod - 2);
    // ll x, y;
    // exgcd(v, mod, x, y);
    // return (x % mod + mod) % mod;
}
int fac[N], ifac[N];
void init_binom(int n) {
    fac[0] = ifac[0] = 1;
    for (int i = 1; i <= n; i++) fac[i] = fac[i - 1] * 1ll * i % mod;
    ifac[n] = getinv(fac[n]);
    for (int i = n; i > 1; i--) ifac[i - 1] = ifac[i] * 1ll * i % mod;
}
int binom(int a, int b) {
    if (b < 0 || a < 0 || b > a) return 0;
    return fac[a] * 1ll * ifac[b] % mod * ifac[a - b] % mod;
}
int getphi(int x) {
    int res = 1;
```

```
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            x /= i;
            res *= (i - 1);
            while (x % i == 0) {
                x /= i;
                res *= i;
            }
        }
    }
    if (x > 1) res *= (x - 1);
    return res;
}
int prime[N], pcnt;
bool isp[N];
int get_prime(int n) {
    for (int i = 2; i <= n; i++) {
        if (!isp[i]) prime[++pcnt] = i;
        for (int j = 1; j <= pcnt && i * prime[j] <= n; j++) {
            isp[prime[j] * i] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}
pll get_up(ll a, ll b, ll x1, ll x2) {
    //x2>=ax+b>=x1 a >= 0
    if (a == 0) return (b >= x1 && b <= x2) ? (pll){-1e18, 1e18} : (pll){1,
        0};
    ll L, R;
    ll l = (x1 - b) / a - 3;
    for (L = l; L * a + b < x1; L++);
    ll r = (x2 - b) / a + 3;
    for (R = r; R * a + b > x2; R--);
    return {L, R};
}
pll get_dn(ll a, ll b, ll x1, ll x2) {
    //x2>=ax+b>=x1 a <= 0
    if (a == 0) return (b >= x1 && b <= x2) ? (pll){-1e18, 1e18} : (pll){1,
        0};
    ll L, R;
    ll l = (x2 - b) / a - 3;
    for (L = l; L * a + b > x2; L++);
    ll r = (x1 - b) / a + 3;
    for (R = r; R * a + b < x1; R--);
```

```
    return {L, R};
}
```

## 5.2 CRT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 100005;
ll n, m, a;
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b != 0) {
        ll g = exgcd(b, a % b, y, x);
        return y -= a / b * x, g;
    } return x = 1, y = 0, a;
}
ll getinv(ll a, ll mod) {
    ll x, y;
    exgcd(a, mod, x, y);
    x = (x % mod + mod) % mod;
    return x;
}
int get(ll x) {
    return x < 0 ? -1 : 1;
}
ll mul(ll a, ll b, ll mod) {
    ll res = 0;
    if (a == 0 || b == 0) return 0;
    ll f = get(a) * get(b);
    a = abs(a), b = abs(b);
    for (; b; b >>= 1, a = (a + a) % mod) if (b & 1) res = (res + a) % mod;
    res *= f;
    if (res < 0) res += mod;
    return res;
}
// m 互质
// int main() {
//     cin >> n;
//     ll phi = 1;
//     for (int i = 1; i <= n; i++) {
//         cin >> m[i] >> a[i];
//         phi *= m[i];
//     }
//     ll ans = 0;
```

```
//     for (int i = 1; i <= n; i++) {
//         ll p = phi / m[i], q = getinv(p, m[i]);
//         ans += mul(p, mul(q, a[i], phi), phi);
//         ans %= phi;
//     }
//     cout << ans << '\n';
// }
int main() {
    cin >> n;
    cin >> m >> a;
    for (int i = 2; i <= n; i++) {
        ll nm, na;
        cin >> nm >> na;
        ll x, y;
        ll g = exgcd(m, -nm, x, y), d = (na - a) / g, md = abs(nm / g);
        x = mul(x, d, md);
        ll lc = abs(m / g);
        lc *= nm;
        a = (a + mul(m, x, lc)) % lc;
        m = lc;
    }
    cout << a << '\n';
}
```

### 5.3 Lucas

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 2e5;
int fac[N], ifac[N], mod;
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b != 0) {
        ll g = exgcd(b, a % b, y, x);
        return y -= a / b * x, g;
    } return x = 1, y = 0, a;
}
int fpow(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = a * 1ll * a % mod) if (b & 1) res = res * 1ll * a %
        mod;
    return res;
}
ll getinv(ll a, ll mod) {
```

```
    return fpow(a, mod - 2);
    ll x, y;
    exgcd(a, mod, x, y);
    x = (x % mod + mod) % mod;
    return x;
}
void init_binom(int n) {
    fac[0] = ifac[0] = 1;
    for (int i = 1; i <= n; i++) fac[i] = fac[i - 1] * 1ll * i % mod;
    ifac[n] = getinv(fac[n], mod);
    for (int i = n; i > 1; i--) ifac[i - 1] = ifac[i] * 1ll * i % mod;
}
int binom(int a, int b) {
    if (b < 0 || a < 0 || b > a) return 0;
    return fac[a] * 1ll * ifac[b] % mod * ifac[a - b] % mod;
}
int lucas(int a, int b) {
    if (a < mod) return binom(a, b);
    return lucas(a / mod, b / mod) * 1ll * binom(a % mod, b % mod) % mod;
}
int main() {
    int T;
    cin >> T;
    while (T--) {
        int n, m;
        cin >> n >> m >> mod;
        init_binom(mod - 1);
        cout << lucas(n + m, m) << '\n';
    }
    return 0;
}
```

### 5.4 exLucas

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e6;
ll a1, b1, mod;
ll m[N], a[N], pr[N], tot;
ll pre[N];
ll fpow(ll a, ll b, ll p) {
    ll res = 1;
    for (; b; b >>= 1, a = a * a % p) if (b & 1) res = res * a % p;
```

```
    return res;
}
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b != 0) {
        ll g = exgcd(b, a % b, y, x);
        return y -= a / b * x, g;
    } return x = 1, y = 0, a;
}
ll getinv(ll a, ll mod) {
    ll x, y;
    exgcd(a, mod, x, y);
    x = (x % mod + mod) % mod;
    return x;
}
ll F(ll n, int k) {
    if (n == 0) return 1;
    ll res = fpow(pre[m[k]], n / m[k], m[k]), rem = n % m[k];
    res = res * pre[rem] % m[k];
    return F(n / pr[k], k) * res % m[k];
}
int G(ll n, ll p) {
    if (n < p) return 0;
    return G(n / p, p) + n / p;
}
int get(ll x) {
    return x < 0 ? -1 : 1;
}
ll mul(ll a, ll b, ll mod) {
    ll res = 0;
    if (a == 0 || b == 0) return 0;
    ll f = get(a) * get(b);
    a = abs(a), b = abs(b);
    for (; b; b >>= 1, a = (a + a) % mod) if (b & 1) res = (res + a) % mod;
    res *= f;
    if (res < 0) res += mod;
    return res;
}
int main() {
    cin >> a1 >> b1 >> mod;
    ll x = mod;
    for (ll i = 2; i * i <= x; i++) {
        if (x % i) continue;
        pr[++tot] = i;
        m[tot] = 1;
```

```
        while (x % i == 0) x /= i, m[tot] *= i;
    }
    if (x != 1) pr[++tot] = x, m[tot] = x;
    for (int k = 1; k <= tot; k++) {
        pre[0] = 1;
        for (int i = 1; i <= m[k]; i++) pre[i] = pre[i - 1] * (i % pr[k] == 0
            ? 1 : i) % m[k];
        ll res = F(a1, k) * getinv(F(b1, k), m[k]) % m[k] * getinv(F(a1 - b1,
            k), m[k]) % m[k];
        ll d = G(a1, pr[k]) - G(b1, pr[k]) - G(a1 - b1, pr[k]), r = (d < 0 ?
            getinv(fpow(pr[k], -d, m[k]), m[k]) : fpow(pr[k], d, m[k]));
        res = res * r % mod;
        a[k] = res;
    }
    ll ans = 0;
    for (int i = 1; i <= tot; i++) {
        ll p = mod / m[i], q = getinv(p, m[i]);
        ans += mul(p, mul(q, a[i], mod), mod);
        ans %= mod;
    }
    cout << ans << '\n';
    return 0;
}
```

## 5.5 线性基

```
struct LinerBasis {
    int a[20], pos[20];
    void add(int v, int p) {
        for (int i = 19; i >= 0; i--) if ((v >> i) & 1) {
            if (a[i]) {
                if (p > pos[i]) {
                    swap(p, pos[i]);
                    swap(a[i], v);
                }
                v ^= a[i];
            } else {
                a[i] = v;
                pos[i] = p;
                return;
            }
        }
    }
} b[N];
```

```cpp

LinerBasis operator + (LinerBasis a, LinerBasis b) {
    for (int i = 19; i >= 0; i--) {
        if (b.a[i]) a.add(b.a[i], b.pos[i]);
    }
    return a;
}
```

## 5.6 高斯消元

### 5.6.1 解线性方程组

```cpp
void gauss() {
    for (int i = 0; i < n; i++) {
        int id = i;
        for (int j = i + 1; j < n; j++) if (fabs(a[j][i]) > fabs(a[id][i])) id
             = j;
        for (int j = i; j <= n; j++) swap(a[id][j], a[i][j]);
        if (a[i][i] == 0) {
            puts("No Solution");
            return;
        }
        for (int j = 0; j < n; j++) {
            if (j == i) continue;
            double t = a[j][i] / a[i][i];
            for (int k = i; k <= n; k++) a[j][k] -= a[i][k] * t;
        }
    }
    for (int i = 0; i < n; i++) printf("%.2lf\n", a[i][n] / a[i][i]);
}
```

### 5.6.2 求行阶梯矩阵

```cpp
bool gauss() {
    int k = 1;
    for (int i = 1; i <= m; i++) {
        if (k > n) break;
        if (a[k][i] == 0) {
            for (int j = k + 1; j <= n; j++) if (a[j][i] != 0) {
                for (int l = 1; l <= m + 1; l++) swap(a[j][l], a[k][l]);
                break;
            }
        }
        if (a[k][i] == 0) continue;
```

```
            for (int j = k + 1; j <= n; j++) if (a[j][i] == 1) {
                for (int l = i; l <= m + 1; l++) a[j][l] ^= a[k][l];
            }
            k++;
        }
    int flag = 1;
    for (int i = k; i <= n; i++) if (a[i][m + 1] == 1) flag = 0;
    return flag;
}
```

### 5.6.3 解不定方程

```
#define fi first
#define se second
typedef long long ll;
typedef pair<ll, ll> pll;
typedef long double ld;
//std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().
    count());
#define y1 miku
const int mod = 998244353;
const int N = 1e5 + 5;
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b) {
        ll d = exgcd(b, a % b, y, x);
        return y -= a / b * x, d;
    } return x = 1, y = 0, a;
}
pll get_up(ll a, ll b, ll x1, ll x2) {
    //x2>=ax+b>=x1
    if (a == 0) return (b >= x1 && b <= x2) ? (pll){0, min(n, m)} : (pll){1,
        0};
    ll L, R;
    ll l = (x1 - b) / a - 3;
    for (L = l; L * a + b < x1; L++);
    ll r = (x2 - b) / a + 3;
    for (R = r; R * a + b > x2; R--);
    return {L, R};
}
pll get_dn(ll a, ll b, ll x1, ll x2) {
    //x2>=ax+b>=x1
    if (a == 0) return (b >= x1 && b <= x2) ? (pll){0, min(n, m)} : (pll){1,
        0};
    ll L, R;
```

```cpp
    ll l = (x2 - b) / a - 3;
    for (L = l; L * a + b > x2; L++);
    ll r = (x1 - b) / a + 3;
    for (R = r; R * a + b < x1; R--);
    return {L, R};
}
//ax+b+c=0 [x1,x2] [y1,y2]
ll solve(ll a, ll b, ll c, ll x1, ll x2, ll y1, ll y2) {
    if (a == 0 && b == 0) return (c == 0) * (y2 - y1 + 1) * (x2 - x1 + 1);
    ll x, y, d = exgcd(a, b, x, y);
    if (c % d != 0) return 0;
    x *= c / d, y *= c / d;
    ll sx = b / d, sy = -a / d;
    auto A = (sx > 0 ? get_up(sx, x, x1, x2) : get_dn(sx, x, x1, x2));
    auto B = (sy > 0 ? get_up(sy, y, y1, y2) : get_dn(sy, y, y1, y2));
    A.fi = max(A.fi, B.fi), A.se = min(A.se, B.se);
    return max(0ll, A.se - A.fi + 1);
}
```

## 5.7 矩阵树定理

一、无向无环图

$A$ 为邻接矩阵，$A[i][j] = i \to j$ 的边数

$D$ 为度数矩阵，$D[i][i] = \sum_{j=1}^{n} A[i][j] = i$ 的度数，其他位置为 0

基尔霍夫矩阵 $K = D - A$, 令 $K' = K$ 的去掉第 $k$ 行第 $k$ 列 ($k$ 任意) 的 $n-1$ 阶主子式

$det(K') =$ 该无向图生成树个数

特别地，完全图生成树个数是 $n^{n-2}$

二、加权

求所有生成树边权的乘积之和，需要把邻接矩阵中边的条数改为为边权和

度数矩阵改为 $D[i][i] = \sum_{j=1}^{n} A[i][j]$

三、有向图

对于有根外向树，需要把度数矩阵改为入度和，$D[i][i] = \sum_{j=1}^{n} A[j][i]$

对于有根内向树，需要把度数矩阵改为出度和，$D[i][i]=\sum_{j=1}^{n}A[i][j]$

类似地，求所有有向生成树边权的乘积之和，需要把邻接矩阵改为入/出边边权和

四、变形：边权和的和

求所有生成树边权和的和，给原先边权为 $w$ 的边赋值为一次多项式 $wx+1$，多项式乘法对 $x^2$ 取模，$\prod(w_ix+1)$ 的一次项系数即为 $w_i$ 之和

```
struct P {
    ll x,y;//x是一次项系数，y是常数项
    P (ll x=0,ll y=0):x(x),y(y){}
    friend P operator + (const P &u, const P &v) {
        return P(add(u.x, v.x), add(u.y, v.y));
    }
    friend P operator - (const P &u, const P &v) {
        return P(add(u.x, mod - v.x), add(u.y, mod - v.y));
    }
    friend P operator * (const P &u, const P &v) {
        return P(add(mul(u.x, v.y), mul(u.y, v.x)), mul(u.y, v.y));
    }
    friend P operator / (const P &u, const P &v) {
        ll inv=qpow(v.y, mod-2);
        return P(add(mul(u.x, v.y), mod - mul(u.y, v.x)) * inv % mod * inv %
           mod, mul(u.y, inv));
    }
};
P g[N][N];
ll gauss(P g[N][N],int n){
    P res(0,1);
    for(int i=1;i<=n;++i) {
        int pos=-1;
        for(int j=i;j<=n;++j){
            if(g[j][i].y){
                pos=j;break;
            }
        }
        if(pos==-1)return 0;
        swap(g[i],g[pos]);
        if(pos!=i)res=res*P(0,mod-1);
        res=res*g[i][i];
        P inv=P(0,1)/g[i][i];
        for(int j=i+1;j<=n;++j){
            P t=g[j][i]*inv;
            for(int k=n;k>=i;--k)g[j][k]=g[j][k]-t*g[i][k];
```

```
        }
    }
    return res.x;
}
```

## 5.8 多项式

### 5.8.1 FFT

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

typedef complex<double> cp;
const int N = 4e6 + 5;
const double pi = acos(-1.0);
int n, m, limit = 1, l, rev[N];
cp a[N], b[N];

void FFT(cp *a, int n, int inv) {
    for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        cp wn(cos(pi / k), inv * sin(pi / k));
        for (int i = 0; i < n; i = i + k + k) {
            cp w(1, 0);
            for (int j = 0; j < k; j++, w *= wn) {
                cp x = a[i + j], y = w * a[i + j + k];
                a[i + j] = x + y;
                a[i + j + k] = x - y;
            }
        }
    }
    if (inv < 0) for (int i = 0; i < n; i++) a[i] /= n;
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i <= n; i++) scanf("%lf", &a[i]);
    for (int j = 0; j <= m; j++) scanf("%lf", &b[j]);
    while (n + m >= limit) limit <<= 1, l = l + 1;
    for (int i = 1; i <= limit; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) <<
        (l - 1));
    FFT(a, limit, 1), FFT(b, limit, 1);
    for (int i = 0; i < limit; i++) a[i] *= b[i];
    FFT(a, limit, -1);
```

```
    for (int i = 0; i <= n + m; i++) printf("%d ", (int)(a[i].real() + 0.5));
    return 0;
}
```

### 5.8.2 NTT

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

const int N = 4e6 + 5, mod = 998244353;
int n, m, r[N], lim, a[N], b[N];

int fpow(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = a * 1ll * a % mod) if (b & 1) res = res * 1ll * a %
        mod;
    return res;
}
void ntt(int *x, int lim, int op) {
    int i, j, k, m, gn, g, tmp;
    for (int i = 0; i < lim; i++) if (r[i] < i) swap(x[i], x[r[i]]);
    for (m = 2; m <= lim; m <<= 1) {
        k = m >> 1;
        gn = fpow(3, (mod - 1) / m);
        for (i = 0; i < lim; i += m) {
            g = 1;
            for (j = 0; j < k; j++, g = g * 1ll * gn % mod) {
                tmp = x[i + j + k] * 1ll * g % mod;
                x[i + j + k] = (x[i + j] - tmp + mod) % mod;
                x[i + j] = (x[i + j] + tmp) % mod;
            }
        }
    }
    if (op == -1) {
        reverse(x + 1, x + lim);
        int inv = fpow(lim, mod - 2);
        for (int i = 0; i < lim; i++) x[i] = x[i] * 1ll * inv % mod;
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 0; i <= m; i++) scanf("%d", &b[i]);
```

```
    lim = 1;
    while (lim < (n + m) << 1) lim <<= 1;
    for (int i = 0; i < lim; i++) r[i] = (i & 1) * (lim >> 1) + (r[i >> 1] >>
        1);
    ntt(a, lim, 1), ntt(b, lim, 1);
    for (int i = 0; i < lim; i++) a[i] = a[i] * 1ll * b[i] % mod;
    ntt(a, lim, -1);
    for (int i = 0; i <= n + m; i++) printf("%d ", a[i]);
    return puts(""), 0;
}
```

## 5.9 组合数学

### 5.9.1 小球放盒

第二类斯特林数（斯特林子集数）$\begin{Bmatrix} n \\ k \end{Bmatrix}$，也可记做 $S(n,k)$，表示将 $n$ 个两两不同的元素，划分为 $k$ 个互不区分的非空子集的方案数。

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k\begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} n \\ 0 \end{Bmatrix} = [n=0]$。

假设小球个数为 $n$, 盒子个数为 $m$

1. 小球无标号，盒子有标号，不允许空盒。

即求解方程 $\sum\limits_{i=1}^{m} x_i = n$ 解的个数

即 $\binom{n-1}{m-1}$

2. 小球无标号，盒子有标号，允许空盒。

令 $y_i = x_i + 1$

即求解方程 $\sum\limits_{i=1}^{m} y_i = n$ 解的个数

即 $\binom{n+m-1}{m-1}$

3. 小球有标号，盒子有标号，允许空盒。

即 $m^n$

4. 小球有标号，盒子有标号，不允许空盒。

$m! \times \begin{Bmatrix} n \\ m \end{Bmatrix}$

5. 小球有标号，盒子无标号，不允许空盒。

$\begin{Bmatrix} n \\ m \end{Bmatrix}$

6. 小球有标号，盒子无标号，允许空盒。

$\sum_{i=1}^{m} \begin{Bmatrix} n \\ i \end{Bmatrix}$

7. 小球无标号，盒子无标号，允许空盒。

设 $f[i][j]$ 表示 i 个球放入 j 个盒子的方案数。

1. $i = 0$ 或者 $j = 1$, 方案数为 1

2. $i < j$, $f[i][j] = f[i][i]$

3. $i \geq j$, $f[i][j] = f[i-j][j] + f[i][j-1]$

8. 小球无标号，盒子无标号，不允许空盒。

用 7 的结论，提前在每个盒子放 1 个球。

方案数就是 $f[n-m][m]$

# 6 MISC

## 6.1 完全平方数判断

```
typedef unsigned long long ull;
int sqrt1(ull x) {
    ull y = sqrt(x);
    return y * y == x;
}
constexpr ull calc_table(int k) {
    ull table = 0;
    for (int i = 0; i < 64; i++)
        table |= 1ull << (i * i % (1 << k));
    return table;
}
int sqrt4(ull x) {
    constexpr int k = 6;
```

```
    constexpr auto table = calc_table(k);
    ull y = x % (1 << k);
    if ((table >> y) & 1) return sqrt1(x);
    return 0;
}
```

### 6.2 维护区间 GCD 值

```
int main() {
    for (int i = 1; i <= n; i++) {
        v.push_back({i, a[i]});
        for (int j = (int)(v.size()) - 2; j >= 0; j--) {
            v[j].second = gcd(v[j].second, a[i]);
            if (v[j].second == v[j + 1].second) v.erase(v.begin() + j + 1);
        }
        mp[v[(int)(v.size()) - 1].second] += i - v[(int)(v.size()) - 1].first + 1;
        for (int j = (int)(v.size()) - 2; j >= 0; j--) {
            mp[v[j].second] += v[j + 1].first - v[j].first;
        }
    }
}
```

### 6.3 FastIO

```
//https://github.com/huanghaox1212/FastIO/
struct control{
    int ct,val;
    control(int Ct,int Val=-1):ct(Ct),val(Val){}
    inline control operator()(int Val){
        return control(ct,Val);
    }
}_endl(0),_prs(1),_setprecision(2);
struct FastIO{
    #define IOSIZE 1000000
    char in[IOSIZE],*p,*pp,out[IOSIZE],*q,*qq,ch[20],*t,b,K,prs;
    FastIO():p(in),pp(in),q(out),qq(out+IOSIZE),t(ch),b(1),K(6){}
    ~FastIO(){fwrite(out,1,q-out,stdout);}
    inline char getch(){
        return p==pp&&(pp=(p=in)+fread(in,1,IOSIZE,stdin),p==pp)?b=0,EOF:*p++;
    }
    inline void putch(char x){
        q==qq&&(fwrite(out,1,q-out,stdout),q=out),*q++=x;
```

```
    }
    inline void puts(const char str[]){fwrite(out,1,q-out,stdout),fwrite(str,1,strlen(str),stdout),q=out;}
    inline void getline(string& s){
        s="";
        for( char ch;(ch=getch())!='\n'&&b;)s+=ch;
    }
    #define indef(T) inline FastIO& operator>>(T& x){\
        x=0; char f=0,ch;\
        while(!isdigit(ch=getch())&&b)f|=ch=='-';\
        while(isdigit(ch))x=(x<<1)+(x<<3)+(ch^48),ch=getch();\
        return x=f?-x:x,*this;\
    }
    indef(int)
    indef(long long)
    inline FastIO& operator>>(char& ch){return ch=getch(),*this;}
    inline FastIO& operator>>(string& s){
        s=""; char ch;
        while(isspace(ch=getch())&&b);
        while(!isspace(ch)&&b)s+=ch,ch=getch();
        return *this;
    }
    inline FastIO& operator>>(double& x){
        x=0; char f=0,ch;
        double d=0.1;
        while(!isdigit(ch=getch())&&b)f|=(ch=='-');
        while(isdigit(ch))x=x*10+(ch^48),ch=getch();
        if(ch=='.')while(isdigit(ch=getch()))x+=d*(ch^48),d*=0.1;
        return x=f?-x:x,*this;
    }
    #define outdef(_T) inline FastIO& operator<<(_T x){\
        !x&&(putch('0'),0),x<0&&(putch('-'),x=-x);\
        while(x)*t++=x%10+48,x/=10;\
        while(t!=ch)*q++=*--t;\
        return *this;\
    }
    outdef(int)
    outdef(long long)
    inline FastIO& operator<<(char ch){return putch(ch),*this;}
    inline FastIO& operator<<(const char str[]){return puts(str),*this;}
    inline FastIO& operator<<(const string& s){return puts(s.c_str()),*this;}
    inline FastIO& operator<<(double x){
         int k=0;
        this->operator<<(int(x));
```

```
        putch('.');
        x-=int(x);
        prs&&(x+=5*pow(10,-K-1));
        while(k<K)putch(int(x*=10)^48),x-=int(x),++k;
        return *this;
    }
    inline FastIO& operator<<(const control& cl){
        switch(cl.ct){
            case 0:putch('\n');break;
            case 1:prs=cl.val;break;
            case 2:K=cl.val;break;
        }
    }
    inline operator bool(){return b;}
}io;
```