河南大學

Henan University

Dissonant Interval

Contestant

曹家宁 罗施达 孙小文

Jianing Cao **Shida Luo** **Xiaowen Sun**

目录

1 图论 **1**

1.1 图的连通性 . . . 1
1.1.1 拓扑排序 . . . 1
1.1.2 Tarjan 割点 . . . 1
1.1.3 Tarjan 割边 . . . 2
1.1.4 Tarjan 强连通分量 . . . 2
1.1.5 Tarjan 点双连通分量 . . . 3
1.1.6 Tarjan 边双连通分量 . . . 3
1.2 最小生成树 . . . 4
1.2.1 Kruskal . . . 4
1.2.2 Prim . . . 5
1.2.3 树的重心 . . . 5
1.2.4 欧拉回路 . . . 6
1.3 流和匹配 . . . 7
1.3.1 二分图判定 . . . 7
1.3.2 二分图最大匹配 . . . 8
1.3.3 EdmondsKarp . . . 8
1.3.4 重链剖分 . . . 9
1.3.5 长链剖分 . . . 10

2 字符串 **11**

2.1 字符串哈希 . . . 11
2.2 字符串哈希 plus . . . 12
2.3 KMP . . . 13
2.4 EXKMP . . . 13
2.5 马拉车 . . . 14

3 数学 **14**

3.1 多项式 . . . 14
3.1.1 FFT . . . 14
3.2 数论 . . . 15
3.2.1 区间筛 . . . 15
3.2.2 欧拉筛 . . . 16
3.2.3 MillerRabin . . . 16
3.2.4 PollardRho . . . 17
3.2.5 矩阵 . . . 18
3.3 组合数学 . . . 19
3.3.1 组合数 . . . 19
3.3.2 卢卡斯定理 . . . 19

4 数据结构 **20**

4.1 ST 表 . . . 20
4.2 并查集 . . . 20
4.3 可撤销并查集 . . . 21
4.4 带权并查集 . . . 22
4.5 智慧集 . . . 22
4.6 字典树 . . . 23
4.7 左偏树 . . . 24
4.8 Splay . . . 25
4.9 树状数组 . . . 27
4.9.1 树状数组 . . . 27
4.9.2 树状数组 2 . . . 27
4.9.3 欧拉序 . . . 28
4.9.4 波纹疾走树 . . . 29
4.10 线段树 . . . 31
4.10.1 线段树 simple . . . 31
4.10.2 线段树 . . . 32
4.10.3 动态开点线段树 . . . 34
4.10.4 线段树优化建图 . . . 36
4.10.5 主席树 . . . 37
4.10.6 标记永久化主席树 . . . 38

5 计算几何 **40**
5.1 凸包 40

6 杂项 **42**
6.1 康托展开 42
6.2 逆康托展开 42
6.3 高精度 43
6.4 高维前缀和 45
6.5 命令行 45

7 编译参数 **45**

8 随机素数 **45**

9 常用组合数学公式 **46**

10 常数表 **46**

1 图论

1.1 图的连通性

1.1.1 拓扑排序

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

//拓扑排序
//https://www.luogu.com.cn/problem/B3644
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
    std::vector<std::vector<int>> v(n + 1); //存图
    std::vector<int> d(n + 1); //统计入度数量
    for(int i = 1; i <= n; ++i) { //建图
        int x;
        while((std::cin >> x) && x != 0) {
            v[i].push_back(x);
            d[x]++;
        }
    }
    std::queue<int> q;
    for(int i = 1; i <= n; ++i) {
        if(d[i] == 0) {
            q.push(i); //将入度为0的放入队列
        }
    }
    while(!q.empty()) {
        int id = q.front();
        q.pop();
        std::cout << id << ' ';
        for(auto &nxt  : v[id]) {
            d[nxt]--; //更新入度数
            if(d[nxt] == 0) { //将入度为0的放入队列
                q.push(nxt);
            }
        }
    }
    return 0;
}
```

1.1.2 Tarjan 割点

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求割点
//https://www.luogu.com.cn/problem/P3388
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    std::vector<int> dfn(n + 1), low(n + 1), bel(n + 1), cutPoint(n + 1);
    int cnt = 0, root = 0;
    auto dfs = [&](auto self, int id, int lst) ->void {
        dfn[id] = low[id] = ++cnt;
        int sz = 0; //儿子个数
        for(auto nxt : v[id]) {
            if(!dfn[nxt]) {
                sz++;
                self(self, nxt, id);
                low[id] = std::min(low[id], low[nxt]);
                if(low[nxt] >= dfn[id]) {
                    cutPoint[id] = 1;
                }
            } else if(nxt != lst) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
        if(num <= 1 && id == root) {
            cutPoint[id] = 0;
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            root = i;
            dfs(dfs, i, 0);
        }
    }
    std::cout << std::count(cutPoint.begin() + 1, cutPoint.end(), 1) << '\n';
    for(int i = 1; i <= n; ++i) {
        if(cutPoint[i] == 1) {
            std::cout << i << ' ';
        }
    }
    return 0;
}
```

1.1.3 Tarjan 割边

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求割边
//https://www.luogu.com.cn/problem/P1656
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back({y, i});//记录边id(从1开始)，防止重边
        v[y].push_back({x, i});
    }
    std::vector<int> dfn(n + 1), low(n + 1);
    std::vector<std::pair<int, int>> bridge;
    int cnt = 0;
    auto dfs = [&](auto self, int id, int lid) ->void {
        dfn[id] = low[id] = ++cnt;
        for(auto [nxt, eid] : v[id]) {
            if(!dfn[nxt]) {
                self(self, nxt, eid);
                low[id] = std::min(low[id], low[nxt]);
                if(low[nxt] == dfn[nxt]) { //是割边
                    bridge.push_back({id, nxt});
                }
            } else if(eid != lid) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i, 0);
        }
    }
    std::sort(bridge.begin(), bridge.end());
    for(auto [x, y] : bridge) {
        std::cout << x << ' ' << y << '\n';
    }
    return 0;
}
```

1.1.4 Tarjan 强连通分量

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求强连通分量(scc)
//https://www.luogu.com.cn/problem/B3609
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 0; i < m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
    }
    std::vector<std::vector<int>> scc(n + 1);
    std::vector<int> dfn(n + 1), low(n + 1), ins(n + 1), bel(n + 1);
    std::stack<int> stk;
    int cnt = 0, tot = 0;
    auto dfs = [&](auto self, int id) ->void {
        dfn[id] = low[id] = ++cnt;
        stk.push(id);
        ins[id] = 1;
        for(auto nxt : v[id]) {
            if(!dfn[nxt]) {
                self(self, nxt);
                low[id] = std::min(low[id], low[nxt]);
            } else if(ins[nxt]) {
                low[id] = std::min(low[id], low[nxt]);
            }
        }
        if(dfn[id] == low[id]) {
            ++tot;
            while(true) {
                int num = stk.top();
                stk.pop();
                ins[num] = 0;
                bel[num] = tot;
                scc[tot].push_back(num);
                if(id == num) break;
            }
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i);
        }
    }
```

```
    for(int i = 1; i <= tot; ++i) {
        std::sort(scc[i].begin(), scc[i].end());
    }
    std::sort(scc.begin() + 1, scc.begin() + tot + 1);
    std::cout << tot << '\n';
    for(int i = 1; i <= tot; ++i) {
        for(int j = 0; j < scc[i].size(); ++j) {
            std::cout << scc[i][j] << " \n"[j == scc[i].size() - 1];
        }
    }
    return 0;
}
```

1.1.5 Tarjan 点双连通分量

```
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求点双连通分量
//https://www.luogu.com.cn/problem/P8435
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    std::vector<std::vector<int>> vcc(n + 1);
    std::vector<int> dfn(n + 1), low(n + 1);
    std::stack<int> stk;
    int cnt = 0, tot = 0;
    auto dfs = [&](auto self, int id, int lst) ->void {
        dfn[id] = low[id] = ++cnt;
        stk.push(id);
        int num = 0;
        for(auto nxt : v[id]) {
            if(!dfn[nxt]) {
                num++;
                self(self, nxt, id);
                low[id] = std::min(low[id], low[nxt]);
                if(low[nxt] >= dfn[id]) {
                    ++tot;
                    while(true) {
                        int num = stk.top();
                        stk.pop();
                        vcc[tot].push_back(num);
                        if(num == nxt) break;
                    }
                    vcc[tot].push_back(id);
                }
            } else if(nxt != lst) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
        if(lst == 0 && num == 0) {
            ++tot;
            vcc[tot].push_back(id);
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i, 0);
        }
    }
    std::cout << tot << '\n';
    for(int i = 1; i <= tot; ++i) {
        std::cout << vcc[i].size() << ' ';
        for(int j = 0; j < vcc[i].size(); ++j) {
            std::cout << vcc[i][j] << " \n"[j == vcc[i].size() - 1];
        }
    }
    return 0;
}
```

1.1.6 Tarjan 边双连通分量

用途：求边双连通分量

模板题：洛谷 P8436

```
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back({y, i});
        v[y].push_back({x, i});
    }
```

```
    std::vector<std::vector<int>> ecc(n + 1);
    std::vector<int> dfn(n + 1), low(n + 1);
    std::stack<int> stk;
    int cnt = 0, tot = 0;
    auto dfs = [&](auto self, int id, int lid) ->void {
        dfn[id] = low[id] = ++cnt;
        stk.push(id);
        for(auto [nxt, eid] : v[id]) {
            if(!dfn[nxt]) {
                self(self, nxt, eid);
                low[id] = std::min(low[id], low[nxt]);
            } else if(lid != eid) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
        if(dfn[id] == low[id]) {
            ++tot;
            while(true) {
                int num = stk.top();
                ecc[tot].push_back(num);
                stk.pop();
                if(id == num) break;
            }
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i, 0);
        }
    }
    std::cout << tot << '\n';
    for(int i = 1; i <= tot; ++i) {
        std::cout << ecc[i].size() << ' ';
        for(int j = 0; j < ecc[i].size(); ++j) {
            std::cout << ecc[i][j] << " \n"[j == ecc[i].size() - 1];
        }
    }
    return 0;
}
```

1.2 最小生成树

1.2.1 Kruskal

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

//kruskal算法最小生成树(稀疏图)
//https://www.luogu.com.cn/problem/P3366
struct DSU {
    DSU(int n) : p(n + 1), sz(n + 1, 1) {
        std::iota(p.begin(), p.end(), 0);
    }
    int find(int x) {
        return p[x] == x ? x : p[x] = find(p[x]);
    }
    bool same(int x, int y) {
        return find(x) == find(y);
    }
    int merge(int x, int y) {
        if (same(x, y)) return 0;
        x = find(x), y = find(y);
        if (sz[x] < sz[y]) std::swap(x, y);
        sz[x] += sz[y];
        p[y] = x;
        return x;
    }
    int& size(int x) {
        return sz[find(x)];
    }
    std::vector<int> p, sz;
};

struct edge { //边
    int x, y, w; //点，点，边权
    bool operator<(const edge& o) const {
        return w < o.w;
    }
};

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<edge> v(m);
    DSU dsu(n);
    for(auto &[x, y, w] : v) {
        std::cin >> x >> y >> w;
    }
    std::sort(v.begin(), v.end()); //对边排序
    int ans = 0, tot = 0;
    for(auto [x, y, w] : v) {
        if(!dsu.same(x, y)) {
            dsu.merge(x, y);
            ans += w;
            tot++;
        }
    }
    if(tot != n - 1) {
        std::cout << "orz" << '\n';
```

```
    } else {
        std::cout << ans << '\n';
    }
    return 0;
}
```

1.2.2 Prim

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

//prim算法最小生成树(稠密图)
//https://www.luogu.com.cn/problem/P3366
struct node {
    int id, w;
    bool operator<(const node& o) const {
        return w > o.w;
    }
};

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
    std::vector<int> vis(n + 1);
    for(int i = 0; i < m; ++i) {
        int x, y, w;
        std::cin >> x >> y >> w;
        v[x].push_back({y, w});
        v[y].push_back({x, w});
    }
    std::priority_queue<node> pq; //利用优先队列不断加入最小边
    int ans = 0;
    pq.push({1, 0});
    while(!pq.empty()) {
        auto [id, w] = pq.top();
        pq.pop();
        if(!vis[id]) {
            vis[id] = 1;
            ans += w;
            for(auto [nxt, w] : v[id]) {
                if(!vis[nxt]) {
                    pq.push({nxt, w});
                }
            }
        }
    }
    if(!*std::min_element(vis.begin() + 1, vis.end())) {
        std::cout << "orz" << '\n'; //图不连通
    } else {
        std::cout << ans << '\n';
    }
    return 0;
}
```

1.2.3 树的重心

定义：如果在树中选择某个节点并删除，这棵树将分为若干棵子树，统计子树节点数并记录最大值。取遍树上所有节点，使此最大值取到最小的节点被称为整个树的重心。

```
#include <bits/stdc++.h>
using i64 = long long;

void solve() {
    int n;
    std::cin >> n;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 1; i <= n - 1; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    std::vector<int> sz(n + 1), weight(n + 1);
    int ans = n;
    auto dfs = [&](auto self, int id, int lst) ->void {
        sz[id] = 1;
        for(auto nxt : v[id]) {
            if(nxt == lst) continue;
            self(self, nxt, id);
            weight[id] = std::max(weight[id], sz[nxt]);
            sz[id] += sz[nxt];
        }
        weight[id] = std::max(weight[id], n - sz[id]);
        ans = std::min(ans, weight[id]);
    };
    dfs(dfs, 1, 0);
    for(int i = 1; i <= n; ++i) {
        if(weight[i] == ans) {
            std::cout << i << ' ';
            break;
        }
    }
    std::cout << ans << '\n';
}
//树的重心（重心最多有两个）
//http://bailian.openjudge.cn/practice/1655/
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
```

```
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

1.2.4 欧拉回路

```
#include <bits/stdc++.h>
using namespace std;
// 无向图欧拉回路or通路
struct Euler {
    int id = 0;
    vector<int> edg;           // 存储边
    vector<vector<int>> graph; // 存储图
    // 初始化
    Euler(int n, int m) {
        graph.resize(n + 1);
        edg.resize(m + 1);
    }
    // 添加边
    void add_edge(int u, int v) {
        id++;
        graph[u].push_back(id);
        graph[v].push_back(id);
        edg[id] = u ^ v;
    }
    // 判断是否存在欧拉通路and返回起点
    int is_semiEuler() {
        int n = graph.size() - 1;
        int odd = 0, start = 0;
        for (int i = 1; i <= n; i++) {
            if (graph[i].size() & 1) {
                odd++;
                start = i;
            }
        }
        if (odd == 0) return 1;
        if (odd == 2) return start;
        return 0;
    }
    // 判断是否存在欧拉回路
    bool is_Euler() {
        int n = graph.size() - 1;
        for (int i = 1; i <= n; i++) {
            if (graph[i].size() & 1) {
                return false;
            }
        }
        return true;
    }
    // 求解欧拉回路or通路
    vector<int> Euler_tour(int start) {
        vector<int> tour;
        function<void(int)> dfs = [&](int u) {
            while (!graph[u].empty()) {
                int i = graph[u].back();
                graph[u].pop_back();
                if (edg[i] == -1) continue;
                int v = edg[i] ^ u;
                edg[i] = -1;
                dfs(v);
            }
            tour.push_back(u);
        };
        dfs(start);
        return tour;
    }
};
// 有向图欧拉回路or通路
struct Directed_Euler {
    vector<int> inE, outE;     // 存储入,出度
    vector<vector<int>> graph; // 存储图
    int n, m;
    // 初始化
    Directed_Euler(int n, int m) : n(n), m(m) {
        graph.resize(n + 1);
        inE.resize(n + 1);
        outE.resize(n + 1);
    }
    // 添加边
    void add_edge(int u, int v) {
        graph[u].push_back(v);
        outE[u]++;
        inE[v]++;
    }
    // 判断是否存在欧拉通路and返回起点
    int is_semiEuler() {
        int odd = 0, neodd = 0, start = 0;
        for (int i = 1; i <= n; i++) {
            if (outE[i] - inE[i] == 1) {
                odd++;
                start = i;
            } else if (inE[i] - outE[i] == 1) {
                neodd++;
            } else if (inE[i] != outE[i]) {
```

```
                return 0;
            }
        }
        if (odd == 1 && neodd == 1) {
            return start;
        }
        if (odd == 0 && neodd == 0) {
            return 1;
        }
        return 0;
    }
    // 判断是否存在欧拉回路
    bool is_Euler() {
        int n = graph.size() - 1;
        for (int i = 1; i <= n; i++) {
            if (inE[i] != outE[i])
                return false;
        }
        return true;
    }
    // 求解欧拉回路or通路
    vector<int> Euler_tour(int start) {
        vector<int> tour;
        function<void(int)> dfs = [&](int u) {
            while (!graph[u].empty()) {
                int v = graph[u].back();
                graph[u].pop_back();
                dfs(v);
            }
            tour.push_back(u);
        };
        dfs(start);
        return tour; // 返回的是逆序的欧拉回路or通路
    }
};
```

1.3 流和匹配

1.3.1 二分图判定

时间复杂度： $O(|V|+|E|)$
空间复杂度： $O(|V|)$
模板题： Luogu P1330

```
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    std::vector<int> col(n + 1), vis(n + 1); //染色值1/2，是否标记
    auto dfs = [&](auto self, int id, int val) ->bool { //判定是否是二分图
        col[id] = val;
        vis[id] = 1;
        for(auto nxt : v[id]) {
            if(!col[nxt]) {
                if(!self(self, nxt, val ^ 3)) {
                    return false;
                }
            } else if(col[nxt] == val) {
                return false;
            }
        }
        return true;
    };
    int ans = 0;
    for(int i = 1; i <= n; ++i) {
        if(!vis[i]) {
            col = std::vector<int>(n + 1);
            if(!dfs(dfs, i, 1)) {
                std::cout << "Impossible\n";
                exit(0);
            }
            int A = std::count(col.begin(), col.end(), 1);
            int B = std::count(col.begin(), col.end(), 2);
            ans += std::min(A, B);
        }
    }
    std::cout << ans << '\n';
    return 0;
}
```

1.3.2 二分图最大匹配

时间复杂度： $O(|V_1||V_2|)$
空间复杂度： $O(|E|+|V_1|+|V_2|)$
模板题： Luogu P3386

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

```cpp
using i64 = long long;

struct BipartiteGraph {
    BipartiteGraph(int n, int m)
    : n(n), m(m), g(n + 1), vis(m + 1), mch(m + 1) {};
    void add(int x, int y) {
        g[x].push_back(y);
    }
    bool dfs(int id) {
        for(auto nxt : g[id]) {
            if(!vis[nxt]) {
                vis[nxt] = 1;
                if(!mch[nxt] || dfs(mch[nxt])) {
                    mch[nxt] = id;
                    return true;
                }
            }
        }
        return false;
    }
    int solve() { //求最大匹配
        int res = 0;
        for(int i = 1; i <= n; ++i) {
            std::fill(vis.begin(), vis.end(), false);
            res += dfs(i);
        }
        return res;
    }
    int n, m;
    std::vector<std::vector<int>> g; //存图
    std::vector<bool> vis;  //标记是否搜索过
    std::vector<int> mch;   //mch[i]表示i号点匹配的编号
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m, k;
    std::cin >> n >> m >> k;
    BipartiteGraph bg(n + 1, m + 1);
    for(int i = 1; i <= k; ++i) {
        int x, y;
        std::cin >> x >> y;
        bg.add(x, y);
    }
    std::cout << bg.solve() << '\n';
    return 0;
}
```

1.3.3 EdmondsKarp

时间复杂度： $O(|V||E|^2)$ 实际情况一般远低于此复杂度
空间复杂度： $O(|V|+|E|)$
用途：求最大流
模板题：洛谷 P3376

```cpp
#include <bits/stdc++.h>
using i64 = long long;

template<typename T>
struct MaxFolw {
    struct Edge {
        Edge() = default;
        Edge(int _nxt, T _cap, int _enxt) : nxt(_nxt), cap(_cap), enxt(_enxt) {}
        int nxt, enxt;
        T cap;
    };
    MaxFolw(int n) : head(n, -1), pre(n), mf(n) {}
    void addEdge(int x, int y, T cap) {
        edge.push_back(Edge(y, cap, head[x]));
        head[x] = edge.size() - 1;
        edge.push_back(Edge(x, 0, head[y]));
        head[y] = edge.size() - 1;
    }
    bool bfs(int s, int t) {
        std::fill(mf.begin(), mf.end(), 0);
        mf[s] = INF;
        std::queue<int> q;
        q.push(s);
        while(!q.empty()) {
            int id = q.front();
            q.pop();
            for(int eid = head[id]; ~eid; eid = edge[eid].enxt) {
                auto &[nxt, _, cap] = edge[eid];
                if(mf[nxt] == 0 && cap > 0) {
                    mf[nxt] = std::min(mf[id], cap);
                    pre[nxt] = eid;
                    if(nxt == t) return true;
                    q.push(nxt);
                }
            }
        }
        return false;
    };
    T flow(int s, int t) {
        T flow = 0;
        while(bfs(s, t)) { //找到增广路
            for(int id = t; id != s; id = edge[pre[id] ^ 1].nxt) {
                edge[pre[id]].cap -= mf[t];
```

```
                edge[pre[id] ^ 1].cap += mf[t];
            }
            flow += mf[t];
        }
        return flow;
    }
    std::vector<Edge> edge;
    std::vector<int> head, pre; // pre: id的前驱边
    std::vector<T> mf; //每S~v的流量上限,
    const T INF = INT_MAX;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m, S, T;
    std::cin >> n >> m >> S >> T;
    MaxFolw<i64> mf(n + 1);
    for(int i = 0; i < m; ++i) {
        int x, y, cap;
        std::cin >> x >> y >> cap;
        mf.addEdge(x, y, cap);
    }
    std::cout << mf.flow(S, T) << '\n';
    return 0;
}
```

1.3.4 重链剖分

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

struct HLD {
    HLD(const int &_n) : n(_n), v(_n + 1) {
        fa = dep = son = sz = top = in = out = rin = std::vector<int>(_n + 1);
    }

    void addEdge(const int &x, const int &y) {
        v[x].push_back(y);
        v[y].push_back(x);
    }

    void dfs1(int id, int &t) {
        sz[id] = 1;
        in[id] = t;
        rin[t] = id;
        for(const auto &nxt : v[id]) {
            if(nxt == fa[id]) continue;
            fa[nxt] = id;
            dep[nxt] = dep[id] + 1;
            dfs1(nxt, ++t);
            sz[id] += sz[nxt];
            if(sz[son[id]] < sz[nxt]) {
                son[id] = nxt;
            }
        }
        out[id] = t;
    }
    void dfs2(int id, int t) {
        top[id] = t;
        if(son[id] == 0) return;
        dfs2(son[id], t);
        for(const auto &nxt : v[id]) {
            if(nxt == fa[id] || nxt == son[id]) continue;
            dfs2(nxt, nxt);
        }
    }
    void work(int root = 1) {
        int dfsn = 1;
        dfs1(root, dfsn);
        dfs2(root, root);
    }

    bool isAncestor(int x, int y) {
        return in[x] <= in[y] && out[x] >= out[y];
    }

    int lca(int x, int y) {
        while(top[x] != top[y]) {
            if(dep[top[x]] < dep[top[y]]) {
                std::swap(x, y);
            }
            x = fa[top[x]];
        }
        return (dep[x] < dep[y] ? x : y);
    }

    int dis(int x, int y) {
        return dep[x] + dep[y] - dep[lca(x, y)];
    }

    int kth(int id, int k) {
        if(k > dep[id]) return 0;
        while(dep[id] - dep[top[id]] + 1 <= k) {
            k -= (dep[id] - dep[top[id]] + 1);
            id = fa[top[id]];
        }
        return rin[in[id] - k];
    }
```

```

    std::vector<std::vector<int>> v;
    std::vector<int> fa, dep, son, sz, top, in, out, rin;
    int n;
};

//树链剖分求LCA
//https://www.luogu.com.cn/problem/P3379
int main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(nullptr);
    int n, m, s;
    std::cin >> n >> m >> s;
    HLD tree(n);
    for(int i = 0; i < n - 1; ++i) {
        int x, y;
        std::cin >> x >> y;
        tree.addEdge(x, y);
    }
    tree.work(s);
    for(int i = 0; i < m; ++i) {
        int x, y;
        std::cin >> x >> y;
        std::cout << tree.lca(x, y) << '\n';
    }
    return 0;
}
```

1.3.5 长链剖分

```
#include <bits/stdc++.h>
using i64 = long long;

struct LLD {
    LLD(const int &_n) : n(_n) {
        son = len = dep = top = in = out = lg = std::vector<int>(n + 1);
        v = up = down = std::vector<std::vector<int>>(n + 1);
        m = std::bit_width(std::bit_ceil((unsigned)n));
        fa = std::vector(m + 1, std::vector<int>(n + 1));
        for(int i = 2; i <= n; ++i) {
            lg[i] = lg[i >> 1] + 1;
        }
    }

    void addEdge(const int &x, const int &y) {
        v[x].push_back(y);
        v[y].push_back(x);
    }

    void dfs1(int id, int &t) {
        for(int i = 0; i < m; ++i) {
            fa[i + 1][id] = fa[i][fa[i][id]];
        }
        in[id] = t;
        len[id] = 1;
        for(const auto &nxt : v[id]) {
            if(nxt == fa[0][id]) continue;
            fa[0][nxt] = id;
            dep[nxt] = dep[id] + 1;
            dfs1(nxt, ++t);
            if(len[nxt] + 1 > len[id]) {
                len[id] = len[nxt] + 1;
                son[id] = nxt;
            }
        }
        out[id] = t;
    }

    void dfs2(int id, int t) {
        top[id] = t;
        if(son[id] == 0) return;
        dfs2(son[id], t);
        for(const auto &nxt : v[id]) {
            if(nxt == fa[0][id] || nxt == son[id]) continue;
            dfs2(nxt, nxt);
        }
    }

    void work(int root = 1) {
        int dfsn = 1;
        dfs1(root, dfsn);
        dfs2(root, root);
        for(int i = 1; i <= n; ++i) {
            if(top[i] != i) continue;
            for(int j = 1, now = i; j <= len[i] && now; ++j, now = fa[0][now]) {
                up[i].push_back(now);
            }
            for(int j = 1, now = i; j <= len[i] && now; ++j, now = son[now]) {
                down[i].push_back(now);
            }
        }
    }

    bool isAncestor(int x, int y) { //x是y的祖先
        return in[x] <= in[y] && out[x] >= out[y];
    }

    int lca(int x, int y) {
```

```
        while(top[x] != top[y]) {
            if(dep[top[x]] < dep[top[y]]) {
                std::swap(x, y);
            }
            x = fa[0][top[x]];
        }
        return (dep[x] < dep[y] ? x : y);
    }

    int dis(int x, int y) {
        return dep[x] + dep[y] - dep[lca(x, y)];
    }

    int kth(int id, int k) {
        if(k == 0) return id;
        int t = lg[k];
        k -= (1 << t);
        id = fa[t][id];
        int p = top[id];
        if(dep[id] - dep[p] >= k) {
            id = down[p][(dep[id] - dep[p]) - k];
        } else {
            id = up[p][k - (dep[id] - dep[p])];
        }
        return id;
    }

    std::vector<std::vector<int>> v, up, down, fa;
    std::vector<int> son, len, dep, top, in, out, lg;
    int n, m;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m, s;
    std::cin >> n >> m >> s;
    LLD tree(n);
    for(int i = 1; i <= n - 1; ++i) {
        int x, y;
        std::cin >> x >> y;
        tree.addEdge(x, y);
    }
    tree.work(s);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        std::cout << tree.lca(x, y) << '\n';
    }
    return 0;
}
```

2 字符串

2.1 字符串哈希

```
#include <bits/stdc++.h>
using i64 = long long;

const int NUM = 2, MAXLEN = 60000;//哈希次数，字符串最大长度
const std::vector<i64> base = {31, 37, 233};
const std::vector<i64> mod = {2013265921, 1004535809, 2147483647};
std::vector<std::array<i64, NUM>> fac(MAXLEN + 1);
struct Hash {
    Hash() {}
    Hash(const std::string &s) : n(s.size()), hs(s.size() + 1) {//0-index
        for(int j = 0; j < NUM; ++j) {
            for(int i = 1; i <= n; ++i) {
                hs[i][j] = (hs[i - 1][j] * base[j] + s[i - 1]) % mod[j];
            }
        }
    }
    std::array<i64, NUM> range(int l, int r) {//1-index
        std::array<i64, NUM> res;
        for(int i = 0; i < NUM; ++i) {
            res[i] = (hs[r][i] - hs[l - 1][i] * fac[r - l + 1][i] % mod[i] + mod[
    i]) % mod[i];
        }
        return res;
    }
    int n;
    std::vector<std::array<i64, NUM>> hs;
};

void HashInit() {
    for(int j = 0; j < NUM; ++j) {
        fac[0][j] = 1;
        for(int i = 1; i <= MAXLEN; ++i) {
            fac[i][j] = fac[i - 1][j] * base[j] % mod[j];
        }
    }
}

//字符串hash
//https://www.luogu.com.cn/problem/P3370
int main() {
    std::ios::sync_with_stdio(false);
```

```
    std::cin.tie(nullptr);
    HashInit();//预处理
    int n;
    std::cin >> n;
    std::set<std::array<i64, NUM>> st;
    for(int i = 0; i < n; ++i) {
        std::string s;
        std::cin >> s;
        Hash hs(s);
        st.insert(hs.range(1, s.size()));
    }
    std::cout << st.size() << '\n';
    return 0;
}
```

2.2 字符串哈希 plus

```
#include <bits/stdc++.h>
using i64 = long long;
using i128 = __int128;

struct StringHash {
    struct Hash {
        i64 hash;
        int n;
        Hash() = default;
        Hash(const i64 &hash, const int &n) : hash(hash), n(n) {}
        Hash operator+(const Hash &rhs) {
            return Hash(StringHash::add(rhs.hash, StringHash::mul(hash,
    StringHash::p[rhs.n])), n + rhs.n);
        }
        friend constexpr std::strong_ordering operator<=>(const Hash &lhs, const
    Hash &rhs) {
            return lhs.hash == rhs.hash ? lhs.n <=> rhs.n : lhs.hash <=> rhs.hash
    ;
        }
    };
    constexpr static i64 base = 114514;
    constexpr static i64 mod = (1ll << 61) - 1;
    inline static std::vector<i64> p{1};
    inline static int n = 0;
    std::vector<i64> h;
    StringHash() = default;
    StringHash(const std::string &s) {
        int n = s.size();
        h.resize(n + 1);
        init(2 * n);
        for (int i = 1; i <= n; i++) {
            h[i] = add(s[i - 1], mul(h[i - 1], base));
        }
    }
    Hash getHash(const int &l, const int &r) {
        return Hash(sub(h[r + 1], mul(h[l], p[r - l + 1])), r - l + 1);
    }
private:
    void init(const int &m) {
        if (n > m) return;
        p.resize(m + 1);
        for (int i = n + 1; i <= m; i++) {
            p[i] = mul(p[i - 1], base);
        }
        n = m;
    }
    inline static i64 mul(const i64 &a, const i64 &b) {
        i128 c = (i128)a * b;
        return add(c >> 61, c & mod);
    }
    inline static i64 add(const i64 &a, const i64 &b) {
        return (a + b >= mod ? a + b - mod : a + b);
    }
    inline static i64 sub(const i64 &a, const i64 &b) {
        return (a - b < 0 ? a - b + mod : a - b);
    }
};

int main() {

    return 0;
}
```

2.3 KMP

```
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::string s, p;
    std::cin >> s >> p;
    int n = s.size(), m = p.size();
    s = '#' + s, p = '#' + p;
    std::vector<int> kmp(m + 1);
    for(int i = 2, j = 0; i <= m; ++i) { //求kmp数组
        while(j > 0 && p[i] != p[j + 1]) {
            j = kmp[j];
        }
        if(p[j + 1] == p[i]) {
```

```
            j++;
        }
        kmp[i] = j;
    }
    for(int i = 1, j = 0; i <= n; ++i) {
        while(j > 0 && s[i] != p[j + 1]) {
            j = kmp[j];
        }
        if(s[i] == p[j + 1]) {
            j++;
        }
        if(j == m) {
            std::cout << i - j + 1 << '\n';
            j = kmp[j];
        }
    }
    for(int i = 1; i <= m; ++i) {
        std::cout << kmp[i] << " \n"[i == m];
    }
    return 0;
}
```

2.4 EXKMP

```
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::string a, b;
    std::cin >> a >> b;
    int n = a.size(), m = b.size();
    a = '#' + a, b = '#' + b;
    std::vector<int> z(m + 1), p(n + 1);
    z[1] = m;
    for(int i = 2, l = 0, r = 0; i <= m; ++i) {
        if(i <= r) {
            z[i] = std::min(z[i - l + 1], r - i + 1);
        }
        while(i + z[i] <= m && b[i + z[i]] == b[1 + z[i]]) {
            z[i]++;
        }
        if(i + z[i] - 1 > r) {
            l = i, r = i + z[i] - 1;
        }
    }
    for(int i = 1, l = 0, r = 0; i <= n; ++i) {
        if(i <= r) {
            p[i] = std::min(z[i - l + 1], r - i + 1);
        }
        while(1 + p[i] <= m && i + p[i] <= n && b[1 + p[i]] == a[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] - 1 > r) {
            l = i, r = i + p[i] - 1;
        }
    }
    i64 ans1 = 0, ans2 = 0;
    for(int i = 1; i <= m; ++i) {
        ans1 ^= 1LL * i * (z[i] + 1);
    }
    for(int i = 1; i <= n; ++i) {
        ans2 ^= 1LL * i * (p[i] + 1);
    }
    std::cout << ans1 << '\n' << ans2 << '\n';
    return 0;
}
```

2.5 马拉车

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

//马拉车(manacher)
//https://www.luogu.com.cn/problem/P3805

// 以第i个数为轴的最大回文 v[2 * i + 1]
// 以第i个数和i+1个数中间为轴的最大回文 v[2 * i + 2]
// 以[L, R] 区间中轴的最大回文为v[L + R + 1]
std::vector<int> manacher(const std::string& s) {
    int n = 2 * s.length() + 1;
    std::string t(n, '#');//处理字符串
    for(int i = 0; i < s.length(); ++i) {
        t[2 * i + 1] = s[i];
    }
    std::vector<int> v(n);//记录回文半径  [l, r] <=> [mid - v[mid], mid - v[mid]]
    for(int i = 0, mid = 0; i < n; ++i) { // mid为回文中心
        if(i <= mid + v[mid]) {
            v[i] = std::min(v[2 * mid - i], mid + v[mid] - i);// (t + i) / 2 =
     mid <=> t = 2 * mid - i;
        }
        while(t[i - v[i] - 1] == t[i + v[i] + 1] && 0 <= i - v[i] - 1 && i + v[i]
      + 1 < n) {
            ++v[i];
        }
        if(i + v[i] > mid + v[mid]) {
            mid = i;
        }
```

```
    }
    return v;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::string s;
    std::cin >> s;
    std::vector<int> v = manacher(s);
    int ans = 0;
    for(int i = 0; i < v.size(); ++i) {
        ans = std::max(ans, v[i]);//求最长回文子串
        std::cout << v[i] << " \n"[i == v.size() - 1];
    }
    std::cout << ans << '\n';
    return 0;
}
```

3 数学

3.1 多项式

3.1.1 FFT

模板题：Luogu P3803 **模板题：**ABC 392G

```
#include <bits/stdc++.h>
using i64 = long long;

constexpr double PI = std::numbers::pi_v<double>;

void FFT(std::vector<std::complex<double>> &A, int opt = 1) {
    int n = A.size();
    std::vector<int> p(n);
    for(int i = 0; i < n; ++i) {
        p[i] = p[i / 2] / 2 + (n / 2) * (i & 1);
    }
    for(int i = 0; i < n; ++i) {
        if(i < p[i]) {
            std::swap(A[i], A[p[i]]);
        }
    }
    for(int len = 2; len <= n; len <<= 1) {
        std::complex<double> w1 = {cos(2 * PI / len), sin(2 * PI / len) * opt};
        for(int i = 0; i < n; i += len) {
            std::complex<double> wk = {1, 0};
            for(int j = 0; j < len / 2; ++j) {
                std::complex<double> x = A[i + j];
                std::complex<double> y = A[i + j + len / 2] * wk;
                A[i + j] = x + y;
                A[i + j + len / 2] = x - y;
                wk *= w1;
            }
        }
    }
}

template<typename T>
std::vector<T> multiply(const std::vector<T> &A, const std::vector<T> &B) {
    int n = std::bit_ceil(A.size() + B.size() - 1);
    std::vector<std::complex<double>> v(n);
    for(int i = 0; i < A.size(); ++i) {
        v[i].real(A[i]);
    }
    for(int i = 0; i < B.size(); ++i) {
        v[i].imag(B[i]);
    }
    v.resize(n);
    FFT(v);
    for(int i = 0; i < n; ++i) {
        v[i] *= v[i];
    }
    FFT(v, -1);
    std::vector<T> res(A.size() + B.size() - 1);
    for(int i = 0; i < res.size(); ++i) {
        res[i] = (T)round(v[i].imag() / 2 / n);
    }
    return res;
}

template<typename T>
std::vector<T> convolution(const std::vector<T> &A, std::vector<T> kernel) {
    std::reverse(kernel.begin(), kernel.end());
    auto res = multiply(A, kernel);
    return std::vector(res.begin() + kernel.size() - 1, res.begin() + A.size());
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<int> a(n + 1), b(m + 1);
    for(int i = 0; i <= n; ++i) {
        std::cin >> a[i];
    }
    for(int i = 0; i <= m; ++i) {
```

```
        std::cin >> b[i];
    }
    auto c = multiply(a, b);
    for(int i = 0; i < c.size(); ++i) {
        std::cout << c[i] << " \n"[i + 1 == c.size()];
    }
    return 0;
}
```

3.2 数论

3.2.1 区间筛

```
#include <bits/stdc++.h>
using i64 = long long;

constexpr int MAXN = 2E5;
std::vector<int> prime;
std::vector<bool> nonPrime(MAXN + 1);
void findPrime(int n) {
    nonPrime[0] = nonPrime[1] = 1;
    for(int i = 2; i <= n; ++i) {
        if(nonPrime[i] == false) {
            prime.push_back(i);
        }
        for(int j = 0; i * prime[j] <= n; ++j) {
            nonPrime[i * prime[j]] = true;
            if(i % prime[j] == 0) break;
        }
    }
}

//区间筛, 筛区间[L, R]的质数
//https://www.luogu.com.cn/problem/UVA10140
int main() {
    i64 L, R;
    findPrime(MAXN);
    while(std::cin >> L >> R) {

        std::vector<i64> res;
        std::vector<bool> nonp(R - L + 1);
        for(auto x : prime) {
            if(x > R) break;
            for(int j = std::max((L + x - 1) / x, 2LL); 1LL * j * x <= R; ++j) {
                nonp[j * x - L] = 1;
            }
        }
        for(int i = 0; i <= R - L; ++i) {
            if(nonp[i] == 0 && i + L >= 2) {
                res.push_back(i + L);
            }
        }

        i64 mn = INT_MAX, mx = INT_MIN;
        int mnidx = -1, mxidx = -1;
        for(int i = 1; i < res.size(); ++i) {
            if(res[i] - res[i - 1] < mn) {
                mn = res[i] - res[i - 1];
                mnidx = i;
            }
            if(res[i] - res[i - 1] > mx) {
                mx = res[i] - res[i - 1];
                mxidx = i;
            }
        }
        if(res.size() <= 1) {
            std::cout << "There are no adjacent primes.\n";
        } else {
            std::cout << res[mnidx - 1] << ',' << res[mnidx] << " are closest, "
                      << res[mxidx - 1] << ',' << res[mxidx] << " are most
    distant.\n";
        }
    }
    return 0;
}
```

3.2.2 欧拉筛

```
#include <bits/stdc++.h>
using i64 = long long;
constexpr int MAXN = 1E8;
std::vector<int> prime;
std::vector<bool> nonPrime(MAXN + 1);

void findPrime(int n) { //[0, n]之间素数
    nonPrime[0] = nonPrime[1] = 1;
    for(int i = 2; i <= n; ++i) {
        if(nonPrime[i] == false) {
            prime.push_back(i);
        }
        for(int j = 0; i * prime[j] <= n; ++j) {
            nonPrime[i * prime[j]] = true;
            if(i % prime[j] == 0) break;
        }
    }
}

// 线性莫比乌斯函数
```

```cpp
// const int MAXN = 1E6;

// std::array<int, MAXN + 1> mu;
// std::array<bool, MAXN + 1> nonPrime;
// std::vector<int> prime;

// void init(int n = MAXN) {
//     mu[1] = 1;
//     nonPrime[0] = nonPrime[1] = true;
//     for(int i = 2; i < n; ++i) {
//          if(nonPrime[i] == false) {
//             prime.push_back(i);
//             mu[i] = -1;
//         }
//         for(int j = 0; i * prime[j] <= n; ++j) {
//             nonPrime[i * prime[j]] = true;
//             if(i % prime[j] == 0) break;
//             mu[i * prime[j]] = -mu[i];
//         }
//     }
// };

//线性筛
//https://www.luogu.com.cn/problem/P3383
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q;
    std::cin >> n >> q;
    findPrime(n);
    while(q--) {
        int idx;
        std::cin >> idx;
        std::cout << prime[idx - 1] << '\n';
    }
    return 0;
}
```

3.2.3 MillerRabin

```cpp
#include <bits/stdc++.h>
using i64 = long long;

i64 qpow(i64 a, i64 b, i64 p) {
    i64 res = 1;
    while(b) {
        if(b & 1) {
            res = (__int128)res * a % p;
        }
        a = (__int128)a * a % p;
        b >>= 1;
    }
    return res;
}

bool Minller(i64 n) {
    if(n == 2) return true;
    if(n <= 1 || n % 2 == 0) return false;
    i64 u = n - 1, k = 0;
    while(u % 2 == 0) u /= 2, ++k;
    static std::vector<i64> base = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};
    for(auto x : base) {
        i64 res = qpow(x, u, n);
        if(res == 0 || res == 1 || res == n - 1) continue;
        for(int i = 1; i <= k; ++i) {
            res = (__int128)res * res % n;
            if(res == n - 1) break;
            if(i == k) return false;
        }
    }
    return true;
}

void solve() {
    i64 x;
    std::cin >> x;
    std::cout << (Minller(x) ? "YES" : "NO") << '\n';
}

//Miller_rabin素数测验
//https://www.luogu.com.cn/problem/SP288
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

3.2.4 PollardRho

```cpp
#include <bits/stdc++.h>
using i64 = long long;

```

```cpp
i64 qpow(i64 a, i64 b, i64 p) {
    i64 res = 1;
    while(b) {
        if(b & 1) {
            res = (__int128)res * a % p;
        }
        a = (__int128)a * a % p;
        b >>= 1;
    }
    return res;
}

//Miller_rabin判断质数
bool Miller(i64 n) {
    if(n <= 1 || n % 2 == 0) return (n == 2);
    i64 u = n - 1, k = 0;
    while(u % 2 == 0) u /= 2, ++k;
    static std::vector<i64> base = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};
    for(auto x : base) {
        i64 res = qpow(x, u, n);
        if(res == 0 || res == 1 || res == n - 1) continue;
        for(int i = 1; i <= k; ++i) {
            res = (__int128)res * res % n;
            if(res == n - 1) break;
            if(i == k) return false;
        }
    }
    return true;
}

//Pollard_rho找因子
i64 Pollard_rho(i64 n) {
    assert(n >= 2);
    if(n == 4) return 2;
    static std::mt19937_64 rnd  (std::chrono::steady_clock::now().
    time_since_epoch().count());
    std::uniform_int_distribution<int64_t> rangeRand(1, n - 1);
    i64 c = rangeRand(rnd);
    auto f = [&](i64 x) {
        return ((__int128)x * x + c) % n;
    };
    i64 x = f(0), y = f(x);
    while(x != y) {
        i64 gd = std::gcd(std::abs(x - y), n);
        if(gd != 1) return gd;
        x = f(x), y = f(f(y));
    }
    return n;
}

void solve() {
    i64 x;
    std::cin >> x;
    i64 res = 0;
    auto max_factor = [&](auto self, i64 x) ->void {
        if(x <= res || x < 2) return;
        if(Miller(x)) {
            res = std::max(res, x);
            return;
        }
        i64 p = x;
        while(p == x) {
            p = Pollard_rho(x);
        }
        while(x % p == 0) {
            x /= p;
        }
        self(self, x), self(self, p);
    };
    max_factor(max_factor, x);
    if(res == x) {
        std::cout << "Prime\n";
    } else {
        std::cout << res << '\n';
    }
}

//Pollard_rho快速求大数因子
//https://www.luogu.com.cn/problem/P4718
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

3.2.5 矩阵

```cpp
#include <bits/stdc++.h>
using i64 = long long;

template<typename T>
struct Matrix {
```

```
    Matrix() : n(0), m(0) {};
    Matrix(int _n, int _m) : n(_n), m(_m), mt(n, std::vector<T>(m)){}
    Matrix(const std::vector<std::vector<T>> &v) : Matrix(v.size(), v[0].size())
    {
        for(int i = 0; i < n; ++i) {
            assert(v[i].size() == m);
            for(int j = 0; j < m; ++j) {
                mt[i][j] = v[i][j];
            }
        }
    }
    Matrix<T> operator*(const Matrix<T> &o) {
        assert(m == o.n);
        Matrix<T> res(n, o.m);
        for(int i = 0; i < n; ++i) {
            for(int j = 0; j < o.m; ++j) {
                for(int k = 0; k < m; ++k) {
                    res.mt[i][j] = res.mt[i][j] + mt[i][k] * o.mt[k][j];
                }
            }
        }
        return res;
    }
    Matrix<T> operator*=(const Matrix<T> &o) {
        return *this = *this * o;
    }
    Matrix<T> operator+(const Matrix<T> &o) {
        assert(n == o.n && m == o.m);
        Matrix<T> res(n, m);
        for(int i = 0; i < n; ++i) {
            for(int j = 0; j < m; ++j) {
                res.mt[i][j] = mt[i][j] + o.mt[i][j];
            }
        }
        return res;
    }
    Matrix<T> operator-(const Matrix<T> &o) {
        assert(n == o.n && m == o.m);
        Matrix<T> res(n, m);
        for(int i = 0; i < n; ++i) {
            for(int j = 0; j < m; ++j) {
                res.mt[i][j] = mt[i][j] - o.mt[i][j];
            }
        }
        return res;
    }
    Matrix<T> operator=(const Matrix<T> &o) {
        n = o.n, m = o.m;
        mt = o.mt;
        return *this;
    }
    static Matrix<T> eye(int n) {
        Matrix<T> res(n, n);
        for(int i = 0; i < n; ++i) {
            res.mt[i][i] = 1;
        }
        return res;
    }
    static Matrix<T> qpow(Matrix<T> a, i64 b) {
        Matrix<T> res(Matrix::eye(a.n));
        while(b != 0) {
            if(b & 1) {
                res = res * a;
            }
            a = a * a;
            b >>= 1;
        }
        return res;
    }
    friend std::ostream& operator<<(std::ostream& os, const Matrix<T>& o) {
        for(int i = 0; i < o.n; ++i) {
            for(int j = 0; j < o.m; ++j) {
                os << o.mt[i][j] << " \n"[j + 1 == o.m];
            }
        }
        return os;
    }
    int n, m;
    std::vector<std::vector<T>> mt;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    Matrix<int> res({{1, 2}, {2, 3}});
    Matrix<int> b(res);
    std::cout << Matrix<int>::qpow(res, 3);
    return 0;
}
```

3.3 组合数学

3.3.1 组合数

```
constexpr i64 P = 998244353;
constexpr i64 MAXN = 3000;

std::array<i64, MAXN + 1> fac, inv;
```

```cpp
i64 qpow(i64 a, i64 b) {
    i64 res = 1;
    while(b) {
        if(b & 1) {
            res = res * a % P;
        }
        b >>= 1;
        a = a * a % P;
    }
    return res;
}

void init(int n = MAXN) {
    fac[0] = 1;
    for(int i = 1; i <= n; ++i) {
        fac[i] = fac[i - 1] * i % P;
    }
    inv[n] = qpow(fac[n], P - 2);
    for(int i = n; i >= 1; --i) {
        inv[i - 1] = inv[i] * i % P;
    }
}

//n中选m个
i64 comb(i64 n, i64 m) {
    if(n < m || n <= 0 || m <= 0) return 0;
    return fac[n] * inv[m] % P * inv[n - m] % P;
}
```

3.3.2 卢卡斯定理

$C_n^m \pmod p = C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n \bmod p}^{m \bmod p}$

```cpp
#include <bits/stdc++.h>
using i64 = long long;

i64 qpow(i64 a, i64 b, i64 p) {
    i64 res = 1;
    while(b) {
        if(b & 1) {
            res = res * a % p;
        }
        a = a * a % p;
        b >>= 1;
    }
    return res;
}

void solve() {
    int n, m, p;
    std::cin >> n >> m >> p;
    std::vector<i64> fac(p + 1, 1);
    for(int i = 2; i <= p; ++i) {
        fac[i] = fac[i - 1] * i % p;
    }
    auto comb = [&fac, &p](i64 n, i64 m) ->i64 {
        return fac[n] * qpow(fac[m], p - 2, p) % p * qpow(fac[n - m], p - 2, p) %
         p;
    };
    auto lucas = [&fac, &p, &comb](auto self, i64 n, i64 m) ->i64 {
        if(m == 0) return 1;
        return self(self, n / p, m / p) * comb(n % p, m % p) % p;
    };
    std::cout << lucas(lucas, n + m, m) << '\n';
}

//lucas定理，求大数组合数
//https://www.luogu.com.cn/problem/P3807
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

4 数据结构

4.1 ST 表

时间复杂度：

- 初始化：$O(n \log(n))$
- 查询：$O(1)$

空间复杂度：$O(nlog(n))$

用途：RMQ 问题，不支持修改

模板题：Luogu P3865

```cpp
#include <bits/stdc++.h>
using i64 = long long;

template <typename T, typename Func = std::function<T(const T&, const T&)>>
struct ST {
    ST(const std::vector<T> &v, Func func = [](const T& a, const T& b) {
        return std::max(a, b);
    }) : func(std::move(func)) {
```

```
        int k = std::__lg(v.size());
        st = std::vector<std::vector<T>>(k + 1, std::vector<T>(v.size()));
        st[0] = v;
        for(int i = 0; i < k; ++i) {
            for(int j = 0; j + (1 << (i + 1)) - 1 < v.size(); ++j) {
                st[i + 1][j] = this->func(st[i][j], st[i][j + (1 << i)]);
            }
        }
    }
    T range(int l, int r) {
        int t = std::__lg(r - l + 1);
        return func(st[t][l], st[t][r + 1 - (1 << t)]);
    }
    std::vector<std::vector<T>> st;
    Func func;
};

//ST表(sparseTable)
//https://www.luogu.com.cn/problem/P3865
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q;
    std::cin >> n >> q;
    std::vector<int> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> v[i];
    }
    ST<int> st(v);
    while(q--) {
        int l, r;
        std::cin >> l >> r;
        std::cout << st.range(l, r) << '\n';
    }
    return 0;
}
```

4.2 并查集

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

//并查集(disjoint set union)
//https://www.luogu.com.cn/problem/P3367
struct DSU {
    DSU(int n) : p(n + 1), sz(n + 1, 1) {
        std::iota(p.begin(), p.end(), 0);
    }
    int find(int x) {
        return p[x] == x ? x : p[x] = find(p[x]);
    }
    bool same(int x, int y) {
        return find(x) == find(y);
    }
    int merge(int x, int y) {
        if (same(x, y)) return 0;
        x = find(x), y = find(y);
        if (sz[x] < sz[y]) std::swap(x, y);
        sz[x] += sz[y];
        p[y] = x;
        return x;
    }
    int& size(int x) {
        return sz[find(x)];
    }
    std::vector<int> p, sz;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    DSU dsu(n);
    for(int i = 0; i < m; ++i) {
        int z, x, y;
        std::cin >> z >> x >> y;
        if(z == 1) {
            dsu.merge(x, y);
        } else if(z == 2) {
            std::cout << (dsu.same(x, y) ? 'Y' : 'N') << '\n';
        }
    }
    return 0;
}
```

4.3 可撤销并查集

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

struct RDSU {
    RDSU(int n) : p(n + 1), sz(n + 1, 1) {
        std::iota(p.begin(), p.end(), 0);
    }
    int find(int x) {
        while(p[x] != x) x = p[x];
        return x;
    }
    bool same(int x, int y) {
```

```
        return find(x) == find(y);
    }
    int merge(int x, int y) {
        if (same(x, y)) return 0;
        x = find(x), y = find(y);
        if (sz[x] < sz[y]) std::swap(x, y);
        hsz.push({sz[x], sz[x]});
        hfa.push({p[y], p[y]});
        sz[x] += sz[y];
        p[y] = x;
        return x;
    }
    int& size(int x) {
        return sz[find(x)];
    }
    size_t now() {
        return hsz.size();
    }
    void version(int ver) {
        rollback(now() - ver);
    }
    void rollback(int t = 1) {
        for(int i = 1; i <= t && !hfa.empty(); ++i) {
            hfa.top().first = hfa.top().second;
            hsz.top().first = hsz.top().second;
            hfa.pop(), hsz.pop();
        }
    }
    std::vector<int> p, sz;
    std::stack<std::pair<int&, int>> hsz, hfa;
};

//https://www.starrycoding.com/problem/9
int main() {
   std::ios::sync_with_stdio(false);
   std::cin.tie(nullptr);
   int n, q;
   std::cin >> n >> q;
   RDSU rdsu(n);
   for(int i = 1; i <= q; ++i) {
       int opt;
       std::cin >> opt;
       if(opt == 1) {
           int x, y;
           std::cin >> x >> y;
           rdsu.merge(x, y);
       } else if(opt == 2) {
           rdsu.rollback();
       } else if(opt == 3) {
           int x, y;
           std::cin >> x >> y;
           std::cout << (rdsu.same(x, y) ? "YES" : "NO") << '\n';
       }
   }
   return 0;
}
```

4.4 带权并查集

模板题： Luogu P1196

```
#include <bits/stdc++.h>
using i64 = long long;

struct DSUT {
    DSUT(int n) : fa(n + 1), f(n + 1), sz(n + 1, 1) {
        std::iota(fa.begin(), fa.end(), 0);
    }
    int find(int id) {
        if(id == fa[id]) return id;
        int root = find(fa[id]);
        f[id] += f[fa[id]];
        return fa[id] = root;
    }
    void merge(int x, int y) { //要依据题意修改
        int fx = find(x), fy = find(y);
        fa[fx] = fy;
        f[fx] += sz[fy];
        sz[fy] += sz[fx];
    }
    bool query(int x, int y) {
        return find(x) == find(y);
    }
    void set(int pos, int val) {
        f[pos] = val;
    }
    int get(int pos) {
        return f[pos];
    }
    std::vector<int> fa, f, sz;//父节点，到父节点的权值，集合大小
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
    DSUT dsut(n);
    for(int i = 1; i <= n; ++i) {
```

```
        char opt;
        int x, y;
        std::cin >> opt >> x >> y;
        if(opt == 'M') {
            dsut.merge(x, y);
        } else if(opt == 'C') {
            if(!dsut.query(x, y)) {
                std::cout << -1 << '\n';
            } else {
                std::cout << std::abs(dsut.get(x) - dsut.get(y)) - 1 << '\n';
            }
        }
    }
    return 0;
}
```

4.5 智慧集

时间复杂度：

- 插入：$O(\log(n))$
- 删除：$O(\log(n))$
- 第 k 小：$O(1)$ 前提：每次操作 k 变化不大

空间复杂度：$O(n)$

用途：双指针中位数

模板题：2023ICPC-Jinan-Regional K. Rainbow Subarray

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename T>
struct SmartSet {
    std::multiset<T> small, large;
    T smallSum, largeSum;
    SmartSet() : small(), large(), smallSum(), largeSum() {}
    void insert(T val) {
        if(small.empty() || val > *small.rbegin()) {
            large.insert(val);
            largeSum += val;
        } else {
            small.insert(val);
            smallSum += val;
        }
    }
    void erase(T val) {
        if(small.contains(val)) {
            smallSum -= val;
            small.extract(val);
        } else if(large.contains(val)) {
            largeSum -= val;
            large.extract(val);
        }
    }
    void balance(int k) {
        k = std::max(0, std::min(k, size()));
        while(small.size() > k) {
            T val = *small.rbegin();
            smallSum -= val;
            largeSum += val;
            large.insert(val);
            small.extract(val);
        }
        while(small.size() < k) {
            T val = *large.begin();
            smallSum += val;
            largeSum -= val;
            small.insert(val);
            large.extract(val);
        }
    }
    int size() {
        return small.size() + large.size();
    }
    int smallSize(int k) {
        balance(k);
        return small.size();
    }
    int largeSize(int k) {
        balance(k);
        return large.size();
    }
    T kth(int k) {
        balance(k);
        return *small.rbegin();
    }
    T getSmallSum(int k) {
        balance(k);
        return smallSum;
    }
    T getLargeSum(int k) {
        balance(k);
        return largeSum;
    }
};

void solve() {
    i64 n, k;
    std::cin >> n >> k;
```

```
    std::vector<i64> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> v[i];
        v[i] = v[i] - i + n;
    }
    SmartSet<i64> sst;
    int ans = 1;
    for(int i = 1, j = 1; j <= n; ++j)  {
        sst.insert(v[j]);
        while(true) {
            int len = (j - i + 1);
            int mid = (len + 1) / 2;
            i64 target = sst.kth(mid);
            i64 res = 1LL * target * (sst.smallSize(mid) - sst.largeSize(mid)) + sst.getLargeSum(mid) - sst.getSmallSum(mid) ;
            if(res > k) {
                sst.erase(v[i]);
                ++i;
            } else {
                ans = std::max(ans, j - i + 1);
                break;
            }

        }
    }
    std::cout << ans << '\n';
}


int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
}
```

4.6 字典树

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

struct Trie {
    Trie() : v(1) {};
    void insert(const std::string &s) {
        int cur = 0;
        for(const auto &val : s) {
            if(v[cur][val - '0'] == 0) {
                v[cur][val - '0'] = ++idx;
            }
            cur = v[cur][val - '0'];
            if(v.size() <= cur) {
                v.resize(cur + 1);
                tot.resize(cur + 1);
            }
            tot[cur]++;
        }
    }
    int find(const std::string &s) {
        int cur = 0;
        for(const auto &val : s) {
            if(v.size() <= cur || v[cur][val - '0'] == 0) {
                return 0;
            }
            cur = v[cur][val - '0'];
        }
        return tot[cur];
    }
    constexpr static int N = 80;
    int idx = 0;
    std::vector<int> tot;
    std::vector<std::array<int, N>> v;
};

void solve() {
    int n, q;
    std::cin >> n >> q;
    Trie t;
    for(int i = 1; i <= n; ++i) {
        std::string s;
        std::cin >> s;
        t.insert(s);
    }
    for(int i = 1; i <= q; ++i) {
        std::string s;
        std::cin >> s;
        std::cout << t.find(s) << '\n';
    }
}


int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T;
    std::cin >> T;
    while(T--) {
```

```
        solve();
    }

    return 0;
}
```

4.7 左偏树

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename T, typename _Compare = std::less<T>>
struct LeftistHeap {
    struct Node {
        Node(const T &_info) : info(_info){}
        T info;
        int dis = -1;
        Node *ls = nullptr;
        Node *rs = nullptr;
    };
    LeftistHeap() = default;
    LeftistHeap(Comp _cmp) : cmp(std::move(_cmp)) {}
    void merge(LeftistHeap &o) {
        _size += o.size();
        root = merge(root, o.root);
    }
    int dis(Node* &node) {
        return node == nullptr ? -1 : node->dis;
    }
    Node* merge(Node* &x, Node* &y) {
        if(x == nullptr) return y;
        if(y == nullptr) return x;
        if(cmp(x->info, y->info)) std::swap(x, y);
        x->rs = merge(x->rs, y);
        if(dis(x->ls) < dis(x->rs)) {
            std::swap(x->ls, x->rs);
        }
        x->dis = dis(x->rs) + 1;
        return x;
    }
    void push(const T &info) {
        root = merge(root, new Node(info));
        ++_size;
    }
    void pop() {
        Node* temp = root;
        root = merge(root->ls, root->rs);
        delete temp;
        --_size;
    }
    T top() {
        assert(root != nullptr);
        return root->info;
    }
    size_t size() {
        return _size;
    }
    bool empty() {
        return _size == 0;
    }
    _Compare cmp;
    Node* root = nullptr;
    size_t _size = 0;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);


    return 0;
}
```

4.8 Splay

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

class SplayTree {
public:
    SplayTree() {
        tr.push_back(Node());
        insert(INF);
        insert(-INF);
    }
    void insert(int t) { //插入值为t的数
        int id = root, fa = 0;
        while(id && tr[id].val != t) {
            fa = id;
            id = tr[id].nxt[t > tr[id].val];
        }
        if(id) {
            tr[id].cnt++;
        } else {
            id = ++size;
            tr[fa].nxt[t > tr[fa].val] = id;
            tr.push_back(Node(fa, t));
        }
        splay(id);
```

```
    }
    int get_pre(int t) { //查找t的前驱节点
        find(t);
        int id = root;
        if(tr[id].val < t) return id;
        id = tr[id].nxt[0];
        while(tr[id].nxt[1]) {
            id = tr[id].nxt[1];
        }
        splay(id);
        return id;
    }
    int get_suc(int t) { //查找t的后继节点
        find(t);
        int id = root;
        if(tr[id].val > t) return id;
        id = tr[id].nxt[1];
        while(tr[id].nxt[0]) {
            id = tr[id].nxt[0];
        }
        splay(id);
        return id;
    }
    void find(int t) {  //查找值为t的节点，并将该节点转到根
        int id = root;
        while(tr[id].nxt[t > tr[id].val] && t != tr[id].val) {
            id = tr[id].nxt[t > tr[id].val];
        }
        splay(id);
    }
    void erase(int t) { //删除值为t的，只删除1个
        int pre = get_pre(t);
        int suc = get_suc(t);
        splay(pre);
        splay(suc, pre);
        int tid = tr[suc].nxt[0];//目标节点
        if(tr[tid].cnt > 1) {
            tr[tid].cnt--;
            splay(tid);          //向上更新其他节点
        } else {
            tr[suc].nxt[0] = 0;
            splay(suc);          //向上更新其他节点
        }
    }
    int get_root() {
        return root;
    }
    int get_rank(int t) { //查一个数t的排名
        insert(t);
        int res = tr[tr[root].nxt[0]].size;
        erase(t);
        return res;
    }
    int get_kth(int t) {   //查找第k个节点编号
        t++;               //有哨兵，所以++
        int id = root;
        while(true) {
            pushdown(id); //向下传递懒标记
            const auto &[x, y] = tr[id].nxt;
            if(tr[x].size + tr[id].cnt < t) {
                t -= tr[x].size + tr[id].cnt;
                id = y;
            } else {
                if(tr[x].size >= t) {
                    id = tr[id].nxt[0];
                } else {
                    return id;
                }
            }
        }
    }
    int get_val(int t) { //查找排名为t的数的数值
        int id = get_kth(t);
        splay(id);
        return tr[id].val;
    }
    void reverse(int l, int r) { //反转区间[l, r]
        l = get_kth(l - 1), r = get_kth(r + 1);
        splay(l, 0), splay(r, l);
        tr[tr[r].nxt[0]].tag ^= 1;
    }
    void output(int id) { //中序遍历
        pushdown(id);
        const auto &[x, y] = tr[id].nxt;
        if(x != 0) output(x);
        if(std::abs(tr[id].val) != INF) {
            std::cout << tr[id].val << ' ';
        }
        if(y) output(y);
    }
    int val(int id) {
        return tr[id].val;
    }
private:
    class Node {
    public:
        Node() {
            nxt = {0, 0};
```

```
            lst = val = size = cnt = tag = 0;
        }
        Node(int _lst, int _val) : lst(_lst), val(_val) {
            nxt = {0, 0};
            tag = 0;
            size = cnt = 1;
        }
        std::array<int, 2> nxt; //左右节点[0左, 1右]
        int lst;                //父亲
        int val;                //权值
        int cnt;                //权值数
        int size;               //子树大小
        int tag;                //懒标记[1翻, 0不翻]
    };
    void rotate(int id) {
        int pid = tr[id].lst, gid = tr[pid].lst;//父节点, 爷节点
        int k = (tr[pid].nxt[1] == id);         //判断id是pid的左节点还是右节点
        tr[pid].nxt[k] = tr[id].nxt[k ^ 1];     //将父节点的k号子节点设置为id的k^1号子节点
        tr[tr[id].nxt[k ^ 1]].lst = pid;        //id的k^1号子节点的父节点设为pid
        tr[id].nxt[k ^ 1] = pid;                //id的k^1号子节点设置为pid
        tr[pid].lst = id;                       //pid的父节点设置为id
        tr[id].lst = gid;                       //id的父节点设置为gid
        tr[gid].nxt[tr[gid].nxt[1] == pid] = id;//gid的子节点设为id
        pushup(pid);                            //更新pid
        pushup(id);                             //更新id
    }
    void splay(int id, int t = 0) {//将id旋转到为t的子节点, 为0时id为根
        while(tr[id].lst != t) {
            int pid = tr[id].lst, gid = tr[pid].lst;
            if(gid != t) { //非根做双旋
                if((tr[pid].nxt[0] == id) == (tr[gid].nxt[0] == pid)) { //直线式转中
                    rotate(pid);
                } else {   //折线式转中
                    rotate(id);
                }
            }
            rotate(id);
        }
        if(t == 0) root = id;
    }
    void pushup(int id) {
        const auto &[x, y] = tr[id].nxt;
        tr[id].size = tr[x].size + tr[y].size + tr[id].cnt;
    }
    void pushdown(int id) {
        if(tr[id].tag) {
            auto &[x, y] = tr[id].nxt;
            std::swap(x, y);
            tr[x].tag ^= 1;
            tr[y].tag ^= 1;
            tr[id].tag = 0;
        }
    }
    std::vector<Node> tr;
    int root = 0; //根节点编号
    int size = 0; //节点个数
    const int INF = INT_MAX;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    SplayTree tr;
    for(int i = 1; i <= n; ++i) {
        tr.insert(i);
    }
    for(int i = 1; i <= m; ++i) {
        int l, r;
        std::cin >> l >> r;
        tr.reverse(l, r);
    }
    tr.output(tr.get_root());
    return 0;
}
```

4.9 树状数组

4.9.1 树状数组

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

//树状数组(Fenwick)
//https://www.luogu.com.cn/problem/P3374
template<typename T>
struct Fenwick {
    Fenwick(int n) : v(n + 1) {};
    void update(int x, T dx) {
        for(int i = x; i < v.size(); i += (i & -i)) {
            v[i] += dx;
        }
    }
    T query(int x) {
        T res{};
        for(int i = x; i > 0; i -= (i & -i)) {
```

```
            res += v[i];
        }
        return res;
    }
    T range(int l, int r) {
        return query(r) - query(l - 1);
    }
    T kth(T k) {
        int pos = 0;
        int logn = std::bit_width(v.size() - 1);
        for(int i = 1 << (logn - 1); i > 0; i >>= 1) {
            if(pos + i < v.size() && v[pos + i] < k) {
                k -= v[pos + i];
                pos += i;
            }
        }
        return pos + 1;
    }
    std::vector<T> v;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    Fenwick<int> tr(n);
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        tr.update(i, x);
    }
    for(int i = 0; i < m; ++i) {
        int o, x, y;
        std::cin >> o >> x >> y;
        if(o == 1) {
            tr.update(x, y);
        } else if (o == 2) {
            std::cout << tr.range(x, y) << '\n';
        }
    }
    return 0;
};
```

4.9.2 树状数组 2

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename T>
struct Fenwick {
    Fenwick(int n) : vec(n + 1), add(n + 1) {}
    void rangeUpdate(int l, int r, T dx) {
        update(l, dx);
        update(r + 1, -dx);
    }
    T rangeQuery(int l, int r) {
        return query(r) - query(l - 1);
    }
    void update(int pos, T dx) {
        for(int i = pos; i < vec.size(); i += (i & -i)) {
            vec[i] += dx;
            add[i] += (pos - 1) * dx;
        }
    }
    T query(int pos) {
        T res{};
        for(int i = pos; i >= 1; i -= (i & -i)) {
            res += pos * vec[i] - add[i];
        }
        return res;
    }
    std::vector<T> vec, add;
};

//树状数组，区间修改，区间查询
//https://www.luogu.com.cn/problem/P3372
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    Fenwick<i64> tr(n);
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        tr.rangeUpdate(i, i, x);
    }
    for(int i = 1; i <= m; ++i) {
        int opt;
        std::cin >> opt;
        if(opt == 1) {
            int l, r, dx;
            std::cin >> l >> r >> dx;
            tr.rangeUpdate(l, r, dx);
        } else if(opt == 2) {
            int l, r;
            std::cin >> l >> r;
```

```
            std::cout << tr.rangeQuery(l, r) << '\n';
        }
    }
    return 0;
}
```

4.9.3 欧拉序

```
#include <bits/stdc++.h>
using namespace std;
typedef long long i64;

struct SparseTable {
    SparseTable() = default;
    vector<int> v;
    vector<vector<int>> st;
    void init(vector<int> &v_) {
        v = v_;
        int k = __lg(v.size());
        st = vector<vector<int>>(k + 1, vector<int>(v.size()));
        iota(st[0].begin(), st[0].end(), 0);
        for (int i = 0; i < k; ++i) {
            for (int j = 0; j + (1 << (i + 1)) - 1 < v.size(); ++j) {
                if (v[st[i][j]] < v[st[i][j + (1 << i)]])
                    st[i + 1][j] = st[i][j];
                else
                    st[i + 1][j] = st[i][j + (1 << i)];
            }
        }
    }
    int query_id(int l, int r) {
        int t = __lg(r - l + 1);
        if (v[st[t][l]] < v[st[t][r + 1 - (1 << t)]])
            return st[t][l];
        else
            return st[t][r + 1 - (1 << t)];
    }
};

struct Euler_tours {
    int n, cnt = 0;
    vector<vector<int>> graph;
    vector<int> et_dep, id, et;
    SparseTable st; //节点个数，图
    Euler_tours(int n) : n(n), graph(n + 1), id(n + 1), et_dep(2 * n), et(2 * n)
    {}

    void add_edg(int u, int v) {
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    void dfs(int u, int fa, int dep) {
        et_dep[++cnt] = dep;
        et[cnt] = u;
        id[u] = cnt;
        for (auto v : graph[u]) {
            if (v != fa) {
                dfs(v, u, dep + 1);
                et_dep[++cnt] = dep;
                et[cnt] = u;
            }
        }
        return;
    }

    void init(int root = 1) {
        dfs(root, 0, 1);
        st.init(et_dep);
    }

    int lca(int u, int v) {
        int idu = id[u];
        int idv = id[v];
        if (idu > idv) {
            swap(idu, idv);
        }
        int idlca = st.query_id(idu, idv);
        return et[idlca];
    }
};

void solve() {
    int n, q, root;
    cin >> n >> q >> root;
    Euler_tours et(n);
    for (i64 i = 1; i < n; i++) {
        i64 u, v;
        cin >> u >> v;
        et.add_edg(u, v);
    }
    et.init(root);
    while (q--) {
        i64 u, v;
        cin >> u >> v;
        cout << et.lca(u, v) << "\n";
    }
    return;
```

```
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    i64 T = 1;
    // cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}
```

4.9.4 波纹疾走树

```
#include <bits/stdc++.h>
using i64 = long long;

struct BitRank {
    // block 管理一行一行的bit
    std::vector<unsigned long long> block;
    std::vector<unsigned int> count;
    BitRank() {}
    // 位向量长度
    void resize(const unsigned int num) {
        block.resize(((num + 1) >> 6) + 1, 0);
        count.resize(block.size(), 0);
    }
    // 设置i位bit
    void set(const unsigned int i, const unsigned long long val) {
        block[i >> 6] |= (val << (i & 63));
    }
    void build() {
        for (unsigned int i = 1; i < block.size(); i++) {
            count[i] = count[i - 1] + __builtin_popcountll(block[i - 1]);
        }
    }
    // [0, i) 1的个数
    unsigned int rank1(const unsigned int i) const {
        return count[i >> 6] + __builtin_popcountll(block[i >> 6] & ((1ULL << (i & 63)) - 1ULL));
    }
    // [i, j) 1的个数
    unsigned int rank1(const unsigned int i, const unsigned int j) const {
        return rank1(j) - rank1(i);
    }
    // [0, i) 0的个数
    unsigned int rank0(const unsigned int i) const {
        return i - rank1(i);
    }
    // [i, j) 0的个数
    unsigned int rank0(const unsigned int i, const unsigned int j) const {
        return rank0(j) - rank0(i);
    }
};


class WaveletMatrix {
private:
    unsigned int height;
    std::vector<BitRank> B;
    std::vector<int> pos;
public:
    WaveletMatrix() {}
    WaveletMatrix(std::vector<int> vec) : WaveletMatrix(vec, *std::max_element(vec.begin(), vec.end()) + 1) {}
    // sigma: 字母表大小(字符串的话), 数字序列的话是数的种类
    WaveletMatrix(std::vector<int> vec, const unsigned int sigma) {
        height = (sigma == 1) ? 1 : (64 - __builtin_clzll(sigma - 1));
        B.resize(height), pos.resize(height);
        for (unsigned int i = 0; i < height; ++i) {
            B[i].resize(vec.size());
            for (unsigned int j = 0; j < vec.size(); ++j) {
                B[i].set(j, get(vec[j], height - i - 1));
            }
            B[i].build();
            auto it = stable_partition(vec.begin(), vec.end(), [&](int c) {
                return !get(c, height - i - 1);
            });
            pos[i] = it - vec.begin();
        }
    }

    int get(const int val, const int i) {
        return (val >> i) & 1;
    }

    // [l, r] 中val出现的频率
    int rank(const int l, const int r, const int val) {
        return rank(r, val) - rank(l - 1, val);
    }

    // [0, i] 中val出现的频率
    int rank(int i, int val) {
        ++i;
        int p = 0;
        for (unsigned int j = 0; j < height; ++j) {
            if (get(val, height - j - 1)) {
                p = pos[j] + B[j].rank1(p);
```

```
                i = pos[j] + B[j].rank1(i);
            } else {
                p = B[j].rank0(p);
                i = B[j].rank0(i);
            }
        }
      return i - p;
    }

    // [l, r] 中k小
    int kth(int l, int r, int k) {
        ++r;
        int res = 0;
        for (unsigned int i = 0; i < height; ++i) {
            const int j = B[i].rank0(l, r);
            if (j >= k) {
                l = B[i].rank0(l);
                r = B[i].rank0(r);
            } else {
                l = pos[i] + B[i].rank1(l);
                r = pos[i] + B[i].rank1(r);
                k -= j;
                res |= (1 << (height - i - 1));
            }
      }
      return res;
    }

    // [l,r] 在[a, b] 值域的数字个数
    int rangeFreq(const int l, const int r, const int a, const int b) {
        return rangeFreq(l, r + 1, a, b + 1, 0, 1 << height, 0);
    }
    int rangeFreq(const int i, const int j, const int a, const int b, const int l
    , const int r, const int x) {
        if (i == j || r <= a || b <= l) return 0;
        const int mid = (l + r) >> 1;
        if (a <= l && r <= b) {
            return j - i;
        } else {
            const int left = rangeFreq(B[x].rank0(i), B[x].rank0(j), a, b, l, mid
    , x + 1);
            const int right = rangeFreq(pos[x] + B[x].rank1(i), pos[x] + B[x].
    rank1(j), a, b, mid, r, x + 1);
            return left + right;
        }
    }

    // [l,r] 在[a,b] 值域内存在的最小值是什么，不存在返回-1，只支持非负整数
    int rangeMin(int l, int r, int a, int b) {
        return rangeMin(l, r + 1, a, b + 1, 0, 1 << height, 0, 0);
    }
    int rangeMin(const int i, const int j, const int a, const int b, const int l,
     const int r, const int x, const int val) {
        if (i == j || r <= a || b <= l) return -1;
        if (r - l == 1) return val;
        const int mid = (l + r) >> 1;
        const int res = rangeMin(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x +
    1, val);
        if (res < 0) {
            return rangeMin(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(j), a, b,
     mid, r, x + 1, val + (1 << (height - x - 1)));
        } else {
            return res;
        }
    }
};

//波纹疾走树(区间第k小， 区间val出现的频率，区间在值域出现的次数和最小值)
//https://www.luogu.com.cn/problem/P3834
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    int n, q;
    std::cin >> n >> q;
    std::vector<int> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> v[i];
    }
    WaveletMatrix wlm(v);
    for(int i = 1; i <= q; ++i) {
        int l, r, k;
        std::cin >> l >> r >> k;
        std::cout << wlm.kth(l, r, k) << '\n';
    }
    return 0;
}
```

4.10 线段树

4.10.1 线段树 simple

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename Info>
struct SegmentTree {
#define ls (id<<1)
#define rs (id<<1|1)
```

```
    SegmentTree(int n) : n(n), info(n << 2) {}
    SegmentTree(const std::vector<Info> &init) : SegmentTree((int)init.size() - 1) {
        auto build = [&](auto self, int id, int l, int r) ->void {
            if(l == r) {
                info[id] = init[l];
                return;
            }
            int mid = (l + r) / 2;
            self(self, ls, l, mid);
            self(self, rs, mid + 1, r);
            pushup(id);
        };
        build(build, 1, 1, n);
    }
    void pushup(int id) {
        info[id] = info[ls] + info[rs];
    }
    void update(int pos, const Info &val) {
        update(1, 1, n, pos, val);
    }
    Info query(int pos) {
        return rangeQuery(pos, pos);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 1, n, l, r);
    }
    void update(int id, int l, int r, int pos, const Info &val) {
        if(l == r) {
            info[id] = val;
            return;
        }
        int mid = (l + r) / 2;
        if(pos <= mid) {
            update(ls, l, mid, pos, val);
        } else {
            update(rs, mid + 1, r, pos, val);
        }
        pushup(id);
    }
    Info rangeQuery(int id, int l, int r, int x, int y) {
        if(x <= l && r <= y) {
            return info[id];
        }
        int mid = (l + r) / 2;
        Info res;
        if(x <= mid) {
            res = res + rangeQuery(ls, l, mid, x, y);
        }
        if(y > mid) {
            res = res + rangeQuery(rs, mid + 1, r, x, y);
        }
        return res;
    }
#undef ls
#undef rs
    const int n;
    std::vector<Info> info;
};

constexpr int INF = 2E9;

struct Info {
    Info() = default;
    Info(int x, int idx) {
        lmn = rmx = x - idx;
        lmx = rmn = x + idx;
    }
    int lmn = INF;
    int rmn = -INF;
    int lmx = INF;
    int rmx = -INF;
    int ans = 0;
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.lmx = std::max(x.lmx, y.lmx);
    res.rmx = std::max(x.rmx, y.rmx);
    res.lmn = std::min(x.lmn, y.lmn);
    res.rmn = std::min(x.rmn, y.rmn);
    res.ans = std::max({x.ans, y.ans, x.lmx - y.rmn, y.rmx - x.lmn});
    return res;
}

void solve() {
    int n, q;
    std::cin >> n >> q;
    std::vector<Info> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        v[i] = Info(x, i);
    }
    SegmentTree<Info> tr(v);
    std::cout << tr.rangeQuery(1, n).ans << '\n';
    for(int i = 1; i <= q; ++i) {
        int idx, x;
```

```
        std::cin >> idx >> x;
        tr.update(idx, Info(x, idx));
        std::cout << tr.rangeQuery(1, n).ans << '\n';
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

4.10.2 线段树

```
#include <bits/stdc++.h>
using i64 = long long;

//线段树，区间修改，区间查询
//https://www.luogu.com.cn/problem/P3372
template<typename Info, typename Tag>
struct SegmentTree {
#define ls (id<<1)
#define rs (id<<1|1)
    SegmentTree(const std::vector<Info> &init) : SegmentTree((int)init.size() - 1) {
        auto build = [&](auto self, int id, int l, int r) ->void {
            if(l == r) {
                info[id] = init[l];
                return;
            }
            int mid = (l + r) / 2;
            self(self, ls, l, mid);
            self(self, rs, mid + 1, r);
            pushup(id);
        };
        build(build, 1, 1, n);
    }
    void apply(int id, const Tag &dx) {
        info[id].apply(dx);
        tag[id].apply(dx);
    }
    void pushup(int id) {
        info[id] = info[ls] + info[rs];
    }
    void pushdown(int id) {
        apply(ls, tag[id]);
        apply(rs, tag[id]);
        tag[id] = Tag();
    }
    void rangeUpdate(int l, int r, const Tag &dx) {
        rangeUpdate(1, 1, n, l, r, dx);
    }
    void update(int t, const Tag &dx) {
        rangeUpdate(t, t, dx);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 1, n, l, r);
    }
    Info query(int t) {
        return rangeQuery(t, t);
    }
    void rangeUpdate(int id, int l, int r, int x, int y, const Tag &dx) {
        if(x <= l && r <= y) {
            apply(id, dx);
            return;
        }
        int mid = (l + r) / 2;
        pushdown(id);
        if(x <= mid) {
            rangeUpdate(ls, l, mid, x, y, dx);
        }
        if(y > mid) {
            rangeUpdate(rs, mid + 1, r, x, y, dx);
        }
        pushup(id);
    }
    Info rangeQuery(int id, int l, int r, int x, int y) {
        if(x <= l && r <= y) {
            return info[id];
        }
        int mid = (l + r) / 2;
        pushdown(id);
        Info res;
        if(x <= mid) {
            res = res + rangeQuery(ls, l, mid, x, y);
        }
        if(y > mid) {
            res = res + rangeQuery(rs, mid + 1, r, x, y);
        }
        return res;
    }
#undef ls
#undef rs
```

```
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
};

constexpr i64 INF = 1E18;

struct Tag {
    i64 add = 0;
    void apply(const Tag &dx) {
        add += dx.add;
    }
};

struct Info {
    i64 mn = INF;
    i64 mx = -INF;
    i64 sum = 0;
    i64 len = 1;
    void apply(const Tag &dx) {
        mn += dx.add;
        mx += dx.add;
        sum += len * dx.add;
    }
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.mn = std::min(x.mn, y.mn);
    res.mx = std::max(x.mx, y.mx);
    res.sum = x.sum + y.sum;
    res.len = x.len + y.len;
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<Info> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        v[i] = {x, x, x, 1};
    }
    SegmentTree<Info, Tag> tr(v);
    // SegmentTree<Info, Tag> tr(n);
    // for(int i = 1; i <= n; ++i) {
    //     int x;
    //     std::cin >> x;
    //     tr.update(i, Tag(x));
    // }
    while(m--) {
        int opt, x, y;
        std::cin >> opt >> x >> y;
        if(opt == 1) {
            int k;
            std::cin >> k;
            tr.rangeUpdate(x, y, Tag(k));
        } else if(opt == 2) {
            std::cout << tr.rangeQuery(x, y).sum << '\n';
        }
    }
    return 0;
}
```

4.10.3 动态开点线段树

```
#include <bits/stdc++.h>
using i64 = long long;
constexpr int MAXN = 2E5;

template<typename Info, typename T = i64>
struct SegmentTree {
    struct Node {
        Node* l = nullptr;
        Node* r = nullptr;
        Info info;
    };
    SegmentTree(T n) : L(0), R(n) {}
    SegmentTree(T L, T R) : L(L), R(R) {}
    void pushup(Node* id) {
        id->info = (id->l == nullptr ? Info() : id->l->info)
                 + (id->r == nullptr ? Info() : id->r->info);
    }
    void update(T pos, const Info &val) {
        update(root, L, R, pos, val);
    }
    Info query(T pos) {
        return rangeQuery(pos, pos);
    }
    Info rangeQuery(T l, T r) {
        return rangeQuery(root, L, R, l, r);
    }
    void update(Node* &id, T l, T r, T pos, const Info &val) {
        if(id == nullptr) id = new Node();
        if(l == r) {
```

```
            id->info = val;
            return;
        }
        T mid = (l + r - 1) / 2;
        if(pos <= mid) {
            update(id->l, l, mid, pos, val);
        } else {
            update(id->r, mid + 1, r, pos, val);
        }
        pushup(id);
    }
    Info rangeQuery(Node* &id, T l, T r, T x, T y) {
        if(y < l || x > r || id == nullptr) return Info();
        if(x <= l && r <= y) {
            return id->info;
        }
        T mid = (l + r - 1) / 2;
        return rangeQuery(id->l, l, mid, x, y)
             + rangeQuery(id->r, mid + 1, r, x, y);
    }

    void merge(SegmentTree<Info, T> seg) {
        root = merge(root, seg.root, L, R);
    }
    Node* merge(Node* &xid, Node* &yid, T l, T r) {
        if(xid == nullptr) return yid;
        if(yid == nullptr) return xid;
        if(l == r) {
            xid->info = (xid->info ^ yid->info);
            return xid;
        }
        T mid = (l + r - 1) / 2;
        xid->l = merge(xid->l, yid->l, l, mid);
        xid->r = merge(xid->r, yid->r, mid + 1, r);
        pushup(xid);
        return xid;
    }

    SegmentTree<Info, T> split(T L, T R) { //分裂出[L, R]的部分
        SegmentTree<Info, T> seg = split(L - 1);
        SegmentTree<Info, T> rem = seg.split(R);
        merge(rem);
        return seg;
    }
    SegmentTree<Info, T> split(T k) { //分裂出(k, ∞]的部分
        SegmentTree<Info, T> seg(L, R);
        seg.root = split(root, L, R, k);
        return seg;
    }
    Node* split(Node* &id, T l, T r, T k) {
        if(id == nullptr || l == r || k >= r) return nullptr;
        Node* nid = new Node();
        if(k < l) {
            std::swap(nid, id);
            return nid;
        }
        T mid = (l + r - 1) / 2;
        if(k > mid) {
            nid->r = split(id->r, mid + 1, r, k);
        } else {
            nid->l = split(id->l, l, mid, k);
            std::swap(nid->r, id->r);
        }
        pushup(id);
        pushup(nid);
        return nid;
    }
    T queryk(T k) { //非通用函数
        return queryk(root, L, R, k);
    }
    T queryk(Node *id, T l, T r, T k) {
        if(id == nullptr) return -1;
        if(id->info.sum < k) return -1;
        if(l == r) return l;
        int mid = (l + r - 1) / 2;
        if(id->l != nullptr && id->l->info.sum >= k) {
            return queryk(id->l, l, mid, k);
        } else if(id->r != nullptr) {
            return queryk(id->r, mid + 1, r, k - (id->l == nullptr ? 0 : id->l->
    info.sum));
        }
        return -1;
    }
    T L, R;
    Node* root = nullptr;
};

struct Info {
    Info() = default;
    Info(i64 _val) {
        sum = _val;
    }
    i64 sum = 0;
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.sum = x.sum + y.sum;
```

```
    return res;
}

Info operator^(const Info &x, const Info &y) {
    Info res;
    res.sum = x.sum + y.sum;
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m, idx = 1;
    std::cin >> n >> m;
    std::vector<SegmentTree<Info>> segs(n + 1, SegmentTree<Info>(MAXN));
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        segs[idx].update(i, x);
    }
    for(int i = 0; i < m; ++i) {
        int opt;
        std::cin >> opt;
        if(opt == 0) {
            int p, x, y;
            std::cin >> p >> x >> y;
            segs[++idx] = segs[p].split(x, y);
        } else if(opt == 1) {
            int p, t;
            std::cin >> p >> t;
            segs[p].merge(segs[t]);
        } else if(opt == 2) {
            int p, x, q;
            std::cin >> p >> x >> q;
            segs[p].update(q, segs[p].query(q).sum + x);
        } else if(opt == 3) {
            int p, x, y;
            std::cin >> p >> x >> y;
            std::cout << segs[p].rangeQuery(x, y).sum << '\n';
        } else if(opt == 4) {
            int p, k;
            std::cin >> p >> k;
            std::cout << segs[p].queryk(k) << '\n';
        }
    }
    return 0;
}
```

4.10.4 线段树优化建图

```
#include <bits/stdc++.h>
using i64 = long long;

struct STOG {
#define ls (id<<1)
#define rs (id<<1|1)
    STOG(int n) : n(n), in(n << 2), out(n << 2), v(n * 7) {
        int tot = n;
        auto build = [&](auto self, int id, int l, int r) ->void {
            if(l == r) {
                in[id] = out[id] = l;
                return;
            }
            int mid = (l + r) / 2;
            self(self, ls, l, mid);
            self(self, rs, mid + 1, r);
            in[id] = ++tot;
            out[id] = ++tot;
            update(in[id], in[ls], 0);
            update(in[id], in[rs], 0);
            update(out[ls], out[id], 0);
            update(out[rs], out[id], 0);
        };
        build(build, 1, 1, n);
    }
    void update(int x, int y, int w) { //连一条从x 到 y的边，边权为w
        v[x].emplace_back(y, w);
    }
    //model == 0 时，从pos 到 [x, y]连边，边权为w
    //model == 1 时，从[x, y] 到 pos连边，边权为w
    void rangeUpdate(int pos, int x, int y,int w, int model) {
        rangeUpdate(1, 1, n, pos, x, y, w, model);
    }
    void rangeUpdate(int id, int l, int r, int pos, int x, int y, int w, auto
    model) {
        if(x <= l && r <= y) {
            if(model == 0) {h
                update(pos, in[id], w);
            } else {
                update(out[id], pos, w);
            }
            return;
        }
        int mid = (l + r) / 2;
        if(x <= mid) {
            rangeUpdate(ls, l, mid, pos, x, y, w, model);
        }
        if(y > mid) {
```

```
            rangeUpdate(rs, mid + 1, r, pos, x, y, w, model);
        }
    }
#undef ls
#undef rs
    int n;
    std::vector<int> in, out;
    std::vector<std::vector<std::pair<int, int>>> v;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q, s;
    std::cin >> n >> q >> s;
    STOG tr(n);
    for(int i = 1; i <= q; ++i) {
        int opt;
        std::cin >> opt;
        if(opt == 1) {
            int pos, x, w;
            std::cin >> pos >> x >> w;
            tr.update(pos, x, w);
        } else if(opt == 2) {
            int pos, x, y, w;
            std::cin >> pos >> x >> y >> w;
            tr.rangeUpdate(pos, x, y, w, 0);
        } else if(opt == 3) {
            int pos, x, y, w;
            std::cin >> pos >> x >> y >> w;
            tr.rangeUpdate(pos, x, y, w, 1);
        }
    }
    auto &graph = tr.v;
    int m = tr.v.size() - 1;
    std::vector<i64> dp(m + 1, LLONG_MAX);
    std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>,
     std::greater<>> pq;
    pq.emplace(0LL, s);
    while(!pq.empty()) {
        auto [w, id] = pq.top();
        pq.pop();
        if(w >= dp[id]) continue;
        dp[id] = w;
        for(const auto &[nxt, dx] : graph[id]) {
            i64 ww = w + dx;
            if(ww < dp[nxt]) {
                pq.emplace(ww, nxt);
            }
        }
    }
    for(int i = 1; i <= n; ++i) {
        std::cout << (dp[i] == LLONG_MAX ? -1 : dp[i]) << " \n"[i == n];
    }
    return 0;
}
```

4.10.5 主席树

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename Info, typename Tag>
struct PersistentTree {
    struct Node {
        int l = 0, r = 0;
        Info info;
        Tag tag;
    };
#define ls(x) (node[x].l)
#define rs(x) (node[x].r)
    PersistentTree(int n) : PersistentTree(std::vector<Info>(n + 1)) {}
    PersistentTree(const std::vector<Info> &init) : n((int)init.size() - 1) {
        node.reserve(n << 3);
        auto build = [&](auto self, int l, int r) ->int {
            node.push_back(Node());
            int id = node.size() - 1;
            if(l == r) {
                node[id].info = init[l];
            } else {
                int mid = (l + r) / 2;
                ls(id) = self(self, l, mid);
                rs(id) = self(self, mid + 1, r);
                node[id].info = node[ls(id)].info + node[rs(id)].info;
            }
            return id;
        };
        root.push_back(build(build, 1, n));
    };
    int update(int version, int pos, const Info &val) {
        root.push_back(update(root[version], 1, n, pos, val));
        return root.size() - 1;
    }
    int update(int version, int pos, const Tag &dx) {
        root.push_back(update(root[version], 1, n, pos, dx));
        return root.size() - 1;
    }
    Info query(int version, int pos) {
```

```
        return rangeQuery(version, pos, pos);
    }
    Info rangeQuery(int version, int l, int r) {
        return rangeQuery(root[version], 1, n, l, r);
    }
    int update(int lst, int l, int r, const int &pos, const Info &val) {
        node.push_back(node[lst]);
        int id = node.size() - 1;
        if(l == r) {
            node[id].info = val;
        } else {
            int mid = (l + r) / 2;
            if(pos <= mid) {
                ls(id) = update(ls(lst), l, mid, pos, val);
            } else if(pos > mid) {
                rs(id) = update(rs(lst), mid + 1, r, pos, val);
            }
            node[id].info = node[ls(id)].info + node[rs(id)].info;
        }
        return id;
    }
    int update(int lst, int l, int r, const int &pos, const Tag &dx) {
        node.push_back(node[lst]);
        int id = node.size() - 1;
        if(l == r) {
            node[id].info.apply(dx);
        } else {
            int mid = (l + r) / 2;
            if(pos <= mid) {
                ls(id) = update(ls(lst), l, mid, pos, dx);
            } else if(pos > mid) {
                rs(id) = update(rs(lst), mid + 1, r, pos, dx);
            }
            node[id].info = node[ls(id)].info + node[rs(id)].info;
        }
        return id;
    }
    Info rangeQuery(int id, int l, int r, const int &x, const int &y) {
        if(x <= l && r <= y) {
            return node[id].info;
        }
        int mid = (l + r) / 2;
        Info res;
        if(x <= mid) {
            res = res + rangeQuery(ls(id), l, mid, x, y);
        }
        if(y > mid) {
            res = res + rangeQuery(rs(id), mid + 1, r, x, y);
        }
        return res;
    }
    int kth(int versionl, int versionr, int k) {
        return kth(root[versionl], root[versionr], 1, n, k);
    }
    int kth(int idx, int idy, int l, int r, int k) { //静态区间第k小，不支持修改
        if(l >= r) return l;
        int mid = (l + r) / 2;
        int dx = node[ls(idy)].info.sum - node[ls(idx)].info.sum;
        if(dx >= k) {
            return kth(ls(idx), ls(idy), l, mid, k);
        } else {
            return kth(rs(idx), rs(idy), mid + 1, r, k - dx);
        }
    }
#undef ls
#undef rs
    const int n;
    std::vector<Node> node;
    std::vector<int> root;
};

struct Tag {
    Tag(int dx = 0) : add(dx) {}
    int add = 0;
    void apply(const Tag &dx) {
        add += dx.add;
    }
};

struct Info {
    int sum = 0;
    void apply(const Tag &dx) {
        sum += dx.add;
    }
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.sum = x.sum + y.sum;
    return res;
}
//主席树(单点修改，历史版本区间查询，静态区间第k小)
//https://www.luogu.com.cn/problem/P3834
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q;
    std::cin >> n >> q;
```

```
    std::vector<int> v(n + 1), tmp(n + 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> v[i];
        tmp[i] = v[i];
    }
    std::sort(tmp.begin() + 1, tmp.end());
    tmp.erase(std::unique(tmp.begin() + 1, tmp.end()), tmp.end());
    int m = tmp.size() - 1;
    PersistentTree<Info, Tag> tr(std::vector<Info>(m + 1));
    std::vector<int> version(n + 1);
    version[0] = tr.root.size() - 1;
    for(int i = 1; i <= n; ++i) {
        int pos = std::lower_bound(tmp.begin() + 1, tmp.end(), v[i]) - tmp.begin
    ();
        version[i] = tr.update(version[i - 1], pos, Tag(1));
    }
    for(int i = 1; i <= q; ++i) {
        int l, r, k;
        std::cin >> l >> r >> k;
        int pos = tr.kth(version[l - 1], version[r], k);
        std::cout << tmp[pos] << '\n';
    }
    return 0;
}
```

4.10.6 标记永久化主席树

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename Info, typename Tag>
struct PersistentTree {
    struct Node {
        int l = 0, r = 0;
        Info info;
        Tag tag;
    };
#define ls(x) (node[id].l)
#define rs(x) (node[id].r)
    PersistentTree(int n) : n(n) {}
    PersistentTree(const std::vector<Info> &init) : PersistentTree((int)init.size
    () - 1) {
        node.reserve(n << 3);
        auto build = [&](auto self, int l, int r) ->int {
            node.push_back(Node());
            int id = node.size() - 1;
            if(l == r) {
                node[id].info = init[l];
            } else {
                int mid = (l + r) / 2;
                ls(id) = self(self, l, mid);
                rs(id) = self(self, mid + 1, r);
                node[id].info = node[ls(id)].info + node[rs(id)].info;
            }
            return id;
        };
        root.push_back(build(build, 1, n));
    };
    int update(int version, int t, const Tag &dx) {
        return rangeUpdate(version, t, t, dx);
    }
    Info query(int version, int t) {
        return rangeQuery(version, t, t);
    }
    int rangeUpdate(int version, int l, int r, const Tag &dx) {
        root.push_back(rangeUpdate(root[version], 1, n, l, r, dx));
        return root.size() - 1;
    }
    Info rangeQuery(int version, int l, int r) {
        return rangeQuery(root[version], 1, n, l, r);
    }
    int rangeUpdate(int lst, int l, int r, const int &x, const int &y, const Tag
    &dx) {
        node.push_back(node[lst]);
        int id = node.size() - 1;
        node[id].info.apply(std::min(r, y) - std::max(l, x) + 1, dx);
        if(x <= l && r <= y) {
            node[id].tag.apply(dx);
        } else {
            int mid = (l + r) / 2;
            if(x <= mid) {
                ls(id) = rangeUpdate(ls(lst), l, mid, x, y, dx);
            }
            if(y > mid) {
                rs(id) = rangeUpdate(rs(lst), mid + 1, r, x, y, dx);
            }
        }
        return id;
    }
    Info rangeQuery(int id, int l, int r, const int &x, const int &y) {
        if(x <= l && r <= y) {
            return node[id].info;
        }
        int mid = (l + r) / 2;
        Info res;
        if(x <= mid) {
            res = res + rangeQuery(ls(id), l, mid, x, y);
        }
```

```
        if(y > mid) {
            res = res + rangeQuery(rs(id), mid + 1, r, x, y);
        }
        res.apply(std::min(r, y) - std::max(l, x) + 1, node[id].tag);
        return res;
    }
#undef ls
#undef rs
    const int n;
    std::vector<Node> node;
    std::vector<int> root;
};

struct Tag {
    Tag(int dx = 0) : add(dx) {}
    int add = 0;
    void apply(const Tag &dx) {
        add += dx.add;
    }
};

struct Info {
    int sum = 0;
    void apply(int len, const Tag &dx) {
        sum += 1LL * len * dx.add;
    }
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.sum = x.sum + y.sum;
    return res;
}

//可持久化线段树(区间修改，区间历史查询)
//https://www.luogu.com.cn/problem/P3919
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q;
    std::cin >> n >> q;
    std::vector<Info> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> v[i].sum;
    }
    PersistentTree<Info, Tag> tr(v);
    std::vector<int> version(q + 1);
    for(int i = 1; i <= q; ++i) {
        int ver, opt, pos;
        std::cin >> ver >> opt >> pos;
        if(opt == 1) {
            int x;
            std::cin >> x;
            int lst = tr.query(version[ver], pos).sum;
            version[i] = tr.update(version[ver], pos, Tag(x - lst));
        } else if(opt == 2) {
            std::cout << tr.query(version[ver], pos).sum << '\n';
            version[i] = version[ver];
        }
    }
    return 0;
}
```

5 计算几何

5.1 凸包

```
#include <bits/stdc++.h>
using i64 = long long;

constexpr long double EPS = 1E-10;

template<typename T>
int sgn(T a){
    return (a > EPS) - (a < -EPS);
}

template<typename T>
struct Vector { //向量
    T x = 0, y = 0;
    Vector(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
    template<typename U> operator Vector<U>() {return Vector<U>(U(x), U(y));}
    Vector operator+(const Vector<T> &o) const {return {x + o.x, y + o.y};}
    Vector operator-(const Vector<T> &o) const {return {x - o.x, y - o.y};}
    Vector operator*(T f) const {return {x * f, y * f};}
    Vector operator/(T f) const {return {x / f, y / f};}
    friend std::istream &operator>>(std::istream &is, Vector &p) {
        return is >> p.x >> p.y;
    }
    friend std::ostream &operator<<(std::ostream &os, Vector p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

template<typename T>
struct Point { //点
    T x = 0, y = 0;
```

```
    Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
    template<typename U> operator Point<U>() {return Point<U>(U(x), U(y));}
    Point operator+(const Vector<T> &o) const {return {x + o.x, y + o.y};}
    Vector<T> operator-(const Point<T> &o) const {return {x - o.x, y - o.y};}
    Point operator-() const {return {-x, -y};}
    Point operator*(T f) const {return {x * f, y * f};}
    Point operator/(T f) const {return {x / f, y / f};}
    friend Point operator*(T f, Point p) {return {p * f};}
    bool operator==(const Point &o) const {
        return sgn(x - o.x) == 0 && sgn(y - o.y) == 0;
    }
    constexpr std::strong_ordering operator<=>(const Point &o) const {
        if(sgn(x - o.x) == 0) {
            return sgn(y - o.y) <=> 0;
        } else {
            return sgn(x - o.x) <=> 0;
        }
    }
    friend std::istream &operator>>(std::istream &is, Point &p) {
        return is >> p.x >> p.y;
    }
    friend std::ostream &operator<<(std::ostream &os, Point p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

template<typename T>
struct Line { //直线
    Point<T> s, t;
    Line() = default;
    Line(Point<T> _s, Point<T> _t) : s(_s), t(_t) {}
};

template<typename T>
struct Seg { //线段
    Point<T> s, t;
    Seg() = default;
    Seg(Point<T> _s, Point<T> _t) : s(_s), t(_t) {}
};

template<typename T, typename U>
struct Circle { //圆
    Point<T> o;
    U r;
};

template<typename T>
T dot(const Vector<T> &a, const Vector<T> &b) { //向量a和向量b的点积
    return a.x * b.x + a.y * b.y;
}

template<typename T>
T dot(const Point<T> &a, const Point<T> &b, const Point<T> &c) { //向量a->b和向量a->c的点积
    return dot(b - a, c - a);
}

template<typename T>
T cross(const Vector<T> &a, const Vector<T> &b) { //向量a和向量b的叉积
    return a.x * b.y - a.y * b.x;
}

template<typename T>
T cross(const Point<T> &a, const Point<T> &b, const Point<T> &c) { //向量a->b和向量a->c的叉积
    return cross(b - a, c - a);
}

template<typename T>
T len2(const Vector<T> &a) { //向量a的模长的平方
    return a.x * a.x + a.y * a.y;
}

template<typename T>
long double len(const Vector<T> &a) { //向量a的模长
    return sqrtl(len2(a));
}

template<typename T>
Vector<T> standardize(const Vector <T> &a) { //向量a的单位向量
    return a / len(a);
}

template<typename T>
long double angle(Vector<T> a, Vector<T> b) {//求两向量夹角[0, pi]
    return fabs(atan2l(cross(a, b), dot(a, b)));
}

template<typename T>
T dis2(const Point<T> &a, const Point<T> &b) { //点a和点b距离的平方（防精度损失）
    return (b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y);
}

template<typename T>
long double dis(const Point<T> &a, const Point<T> &b) { //点a到点b距离
    return sqrtl(dis2(a, b));
}

```

```cpp
template<typename T>
long double angle(const Vector<T> &a, const Vector<T> &b) { //向量a和向量b的夹角弧度
    return acosl(dot(a, b) / len(a) / len(b));
}

template<typename T>
Vector<T> rotate(const Vector <T> &a) { //向量a逆时针旋转pi/2
    return {-a.y, a.x};
}

template<typename T>
Vector<T> rotate(const Vector <T> &a, long double rad) { //向量a逆时针旋转rad弧度
    return {a.x * cosl(rad) - a.y * sinl(rad), a.x * sinl(rad) + a.y * cosl(rad)};
}

template<typename T>
Point<T> rotate(const Point<T> &a, const Point<T> &b, long double rad) { //点b绕点a逆时针旋转rad弧度
    return {
        (b.x - a.x) * cosl(rad) - (b.y - a.y) * sinl(rad) + a.x,
        (b.x - a.x) * sinl(rad) + (b.y - a.y) * cosl(rad) + a.y
    };
}

template<typename T>
bool intersect(const Seg<T> &a, const Seg<T> &b) { //线段a和线段b不严格相交，可以包含端点相交
    return sgn(cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t)) <= 0
        && sgn(cross(b.s, b.t, a.s) * cross(b.s, b.t, a.t)) <= 0;
}

template<typename T>
bool intersectStrictly(const Seg<T> &a, const Seg<T> &b) { //线段a和线段b严格相交，不包含端点相交
    return sgn(cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t)) < 0
        && sgn(cross(b.s, b.t, a.s) * cross(b.s, b.t, a.t)) < 0;
}

template<typename T>
auto getNode(const T &a, const T &b) { //求线段/直线a与线段/直线b的交点（需要先判断是否相交）
    T dx = cross(a.s, b.s, b.t) / cross(a.t - a.s, b.t - b.s);
    return a.s + (a.t - a.s) * dx;
}

template<typename T>
bool onSeg(const Point<T> &p, const Seg<T> &s) {
    if(sgn(cross(s.s, s.t, p)) != 0) return false;
    return std::min(s.s, s.t) <= p && p <= std::max(s.s, s.t);
}

template<typename T, typename U>
int pointOnCircle(const Point<T> &p, const Circle<T, U> &c) {
    return sgn(c.r * c.r - dis2(p, c.o));
}

template<typename T>
std::vector<Point<T>> andrew(std::vector<Point<T>> v) { //Andrew求凸包
    std::sort(v.begin(), v.end());
    std::vector<Point<T>> stk;
    for(int i = 0; i < v.size(); ++i) {
        while(stk.size() > 1 && sgn(cross(stk[stk.size() - 2], stk.back(), v[i])) <= 0) {
            stk.pop_back();
        }
        stk.push_back(v[i]);
    }
    int t = stk.size();
    for(int i = (int)v.size() - 2; i >= 0; --i) {
        while(stk.size() > t && sgn(cross(stk[stk.size() - 2], stk.back(), v[i])) <= 0) {
            stk.pop_back();
        }
        stk.push_back(v[i]);
    }
    stk.pop_back();
    return stk;
};

template<typename T>
std::vector<Point<T>> graham(std::vector<Point<T>> v) { //Graham求凸包
    Point<T> base = *min_element(v.begin(), v.end());
    std::ranges::sort(v, [&](auto p1, auto p2) ->bool {
        if(sgn(cross(base, p1, p2)) == 0) return p1 < p2;
        return sgn(cross(base, p1, p2)) == -1;
    });
    std::vector<Point<T>> stk;
    for(int i = 0; i < v.size(); ++i) {
        while(stk.size() > 1 && sgn(cross(stk[stk.size() - 2], stk.back(), v[i])) >= 0) {
            stk.pop_back();
        }
        stk.push_back(v[i]);
    }
    return stk;
}
```

```

template<typename T>
T diameter2(const std::vector<Point<T>> &v) { //旋转卡壳求凸包直径的平方
    int n = v.size();
    T res = 0;
    for(int i = 0, j = 1; i < n; ++i) {
        while(sgn(cross(v[i], v[(i + 1) % n], v[j]) - cross(v[i], v[(i + 1) % n],
     v[(j + 1) % n])) <= 0) {
            j = (j + 1) % n;
        }
        res = std::max({res, dis2(v[i], v[j]), dis2(v[(i + 1) % n], v[j])});
    }
    return res;
}

template<typename T>
long double diameter(const std::vector<Point<T>> &v) { //旋转卡壳求凸包直径
    return sqrtl(diameter2(v));
}

template<typename T>
long double mindistance(std::vector<Point<T>> v) { //平面最近点对O(nlog(n))
    std::sort(v.begin(), v.end());
    long double d = 1E18;
    auto cmp = [](const Point<T>& a, const Point<T>& b) {
        return a.y < b.y;
    };
    std::multiset<Point<T>, decltype(cmp)> st;
    for(int i = 0, j = 0; i < v.size(); ++i) {
        while(j < i && sgn(v[j].x - v[i].x + d) <= 0) {
            st.erase(v[j]);
            ++j;
        }
        for(auto it = st.lower_bound({v[i].x, v[i].y - T(d)}); it != st.end() &&
     sgn(it->y - v[i].y - T(d + 1)) <= 0; ++it) {
            d = std::min(d, dis(v[i], *it));
        }
        st.insert(v[i]);
    }
    return d;
}

template<typename T>
long double grith(const std::vector<Point<T>> &v) { //求凸包周长
    long double ans = 0;
    for(int i = 0; i < v.size(); ++i) {
        ans += dis(v[i], v[(i + 1) % v.size()]);
    }
    return ans;
}

}


void solve() {

    Point<int> p(1, 1);
    p = p / 2.0;
    std::cout << p << '\n';
    // int n, m;
    // std::cin >> n;
    // std::vector<Point<long double>> A(n);

    // for(int i = 0; i < n; ++i) {
    //     std::cin >> A[i];
    // }
    // std::cin >> m;
    // std::vector<Point<long double>> B(m);
    // for(int i = 0; i < m; ++i) {
    //     std::cin >> B[i];
    // }
    // long double ans = grith(A) + 2.0L * diameter(B) * std::numbers::pi;   //A
     周长 + 2 * B直径 * PI
    // std::cout << std::fixed << std::setprecision(15) << ans << '\n';
}

int main(){
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

6 杂项

6.1 康托展开

```
#include <bits/stdc++.h>
using i64 = long long;
constexpr i64 P = 998244353;

template<typename T>
class Fenwick {
public:
    Fenwick(int n) : v(std::vector<T>(n + 1)) {};
```

```
    void update(int x, T dx) {
        for(int i = x; i < v.size(); i += (i & -i)) {
            v[i] += dx;
        }
    }
    T query(int x) {
        T res{};
        for(int i = x; i > 0; i -= (i & -i)) {
            res += v[i];
        }
        return res;
    }
    T range(int l, int r) {
        return query(r) - query(l - 1);
    }
private:
    std::vector<T> v;
};

//康托展开(求排列的排名)
//https://www.luogu.com.cn/problem/P5367
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
    Fenwick<int> tr(n);
    std::vector<int> p(n + 1);
    std::vector<i64> fac(n + 1, 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> p[i];
        tr.update(p[i], 1);
        fac[i] = fac[i - 1] * i % P;
    }
    i64 ans = 1;
    for(int i = 1; i <= n; ++i) {
        ans = (ans + fac[n - i] * tr.query(p[i] - 1)) % P;
        tr.update(p[i], -1);
    }
    std::cout << ans << '\n';
    return 0;
}
```

6.2 逆康托展开

```
#include <bits/stdc++.h>
using i64 = long long;

template<typename T>
class Fenwick {
public:
    Fenwick(int n) : v(std::vector<T>(n + 1)) {};
    void update(int x, T dx) {
        for(int i = x; i < v.size(); i += (i & -i)) {
            v[i] += dx;
        }
    }
    T query(int x) {
        T res{};
        for(int i = x; i > 0; i -= (i & -i)) {
            res += v[i];
        }
        return res;
    }
    T range(int l, int r) {
        return query(r) - query(l - 1);
    }
private:
    std::vector<T> v;
};

//逆康托展开
//https://acm.hdu.edu.cn/showproblem.php?pid=1027
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    while(std::cin >> n >> m) {
        Fenwick<int> tr(n);
        std::vector<i64> fac(n + 1, 1);
        for(int i = 1; i <= n; ++i) {
            if(fac[i - 1] > m) {
                fac[i] = fac[i - 1];
            } else {
                fac[i] = fac[i - 1] * i;
            }
            tr.update(i, 1);
        }
        m--;
        for(int i = 1; i <= n; ++i) {
            int k = m / fac[n - i];
            int l = k + 1, r = n, res = 1;
            while(l <= r) {
                int mid = (l + r) / 2;
                if(tr.query(mid - 1) <= k) {
                    res = mid;
                    l = mid + 1;
                } else {
```

```
                    r = mid - 1;
                }
            }
            tr.update(res, -1);
            m = m % fac[n - i];
            std::cout << res << " \n"[i == n];
        }
    }
    return 0;
}
```

6.3 高精度

```
#include <bits/stdc++.h>
using i64 = long long;

// using i128= __int128;
// std::istream&operator>>(std::istream &is,i128 &n){
//  std::string s;is>>s;
//  n=0;
//  for(char i:s) n=n*10+i-'0';
//  return is;
// }
// std::ostream &operator<<(std::ostream &os,i128 n){
//  std::string s;
//  while(n){
//      s+='0'+n%10;
//      n/=10;
//  }
//  std::reverse(s.begin(),s.end());
//  return os<<s;
// }

struct Bigint {
    std::string a;
    int sign;
    Bigint() {}
    Bigint(std::string b) {
        (*this) = b;
    }
    int size() {
        return a.size();
    }
    Bigint normalize(int newSign) { //removes leading 0, fixes sign (base)
        for(int i = a.size() - 1; i > 0 && a[i] == '0'; --i) {
            a.erase(a.begin() + i);
        }
        sign = (a.size() == 1 && a[0] == '0') ? 1 : newSign;
        return (*this);
    }
    void operator=(std::string b) {
        a = b[0] == '-' ? b.substr(1) : b;
        reverse(a.begin(), a.end());
        this->normalize(b[0] == '-' ? -1 : 1);
    }
    bool operator<(const Bigint &b) const {
        if(sign != b.sign) {
            return sign < b.sign;
        }
        if(a.size() != b.a.size()) {
            return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
        }
        for(int i = a.size() - 1; i >= 0; --i) {
            if(a[i] != b.a[i]) {
                return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
            }
        }
        return false;
    }
    bool operator==(const Bigint &b) const {
        return (a == b.a && sign == b.sign);
    }
    bool operator!=(const Bigint &b) const {
        return !operator==(b);
    }
    Bigint operator+(Bigint b) {
        if(sign != b.sign) {
            return (*this) -  (-b); //don't modify here
        }
        Bigint c;
        for(int i = 0, carry = 0; i < a.size() || i < b.size() || carry; ++i) {
            carry += (i < a.size() ? a[i] - 48 : 0) + (i < b.a.size() ? b.a[i] -
 48 : 0);
            c.a += (carry % 10 + 48);
            carry /= 10;
        }
        return c.normalize(sign);
    }
    Bigint operator-() {
        sign *= -1;
        return (*this);
    }
    Bigint operator-(Bigint b) {
        if(sign != b.sign) {
            return (*this) + (-b);
        }
        int s = sign; sign = b.sign = 1;
        if((*this) < b) {
```

```
            return (b - (-(*this))).normalize(-s);
        }
        Bigint c;
        for(int i = 0, borrow = 0; i < a.size(); ++i) {
            borrow = (a[i] - borrow - (i < b.size() ? b.a[i] : 48));
            c.a += (borrow >= 0 ? borrow + 48 : borrow + 58);
            borrow = (borrow >= 0 ? 0 : 1);
        }
        return c.normalize(s);
    }
    Bigint operator*(Bigint b) {
        Bigint c("0");
        for(int i = 0, k = a[i] - 48; i < a.size(); ++i, k = a[i] - 48) {
            while(k--) c = c + b;
            b.a.insert(b.a.begin(), '0');
        }
        return c.normalize(sign * b.sign);
    }
    Bigint operator/(Bigint b) {
        assert(b != Bigint("0"));
        if(b.size() == 1 && b.a[0] == '0') {
            b.a[0] /= (b.a[0] - 48);
        }
        Bigint c("0"), d;
        for(int j = 0; j < a.size(); ++j) {
            d.a += "0";
        }
        int dSign = sign * b.sign; b.sign = 1;
        for(int i = a.size() - 1; i >= 0; --i) {
            c.a.insert( c.a.begin(), '0');
            c = c + a.substr( i, 1 );
            while(!(c < b)) {
                c = c - b, d.a[i]++;
            }
        }
        return d.normalize(dSign);
    }
    Bigint operator%(Bigint b) {
        assert(b != Bigint("0"));
        if(b.size() == 1 && b.a[0] == '0') {
            b.a[0] /= (b.a[0] - 48);
        }
        Bigint c("0");
        b.sign = 1;
        for(int i = a.size() - 1; i >= 0; --i) {
            c.a.insert(c.a.begin(), '0');
            c = c + a.substr(i, 1);
            while(!( c < b )) c = c - b;
        }
        return c.normalize(sign);
    }
    friend std::istream& operator>>(std::istream &is, Bigint &integer) {
        std::string input;
        is >> input;
        integer = input;
        return is;
    }
    friend std::ostream& operator<<(std::ostream& os, const Bigint& integer) {
        if (integer.sign == -1) {
            os << "-";
        }
        for (int i = integer.a.size() - 1; i >= 0; --i) {
            os << integer.a[i];
        }
        return os;
    }
};

int main() {
    Bigint a, b;
    std::cin >> a >> b;
    std::cout << a + b << '\n';
    std::cout << a - b << '\n';
    std::cout << a * b << '\n';
    std::cout << a / b << '\n';
    std::cout << a % b << '\n';
    std::cout << (a == b ? "" : "not ") << "equal\n";
    std::cout << "a is " << (a < b ? "" : "not") << "smaller than b\n";
    std::cout << "the max number is:" << std::max(a, b) << '\n';
    std::cout << "the min number is:" << std::min(a, b) << '\n';
    return 0;
}
```

6.4 高维前缀和

时间复杂度：$O(n2^n)$
空间复杂度：$O(n2^n)$
用途：位集合中，求出某个集合的所有子集值之和以及其他可加性操作
模板题：AtCoder ARC100 C

```
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
```

```
    std::vector<std::pair<int, int>> v(1 << n);
    for(int i = 0; i < (1 << n); ++i) {
        std::cin >> v[i].first;
    }
    for(int i = 0; i < n; ++i) {
        for(int j = 0; j < (1 << n); ++j) {
            if(j >> i & 1) { //条件取反 !(j >> i & 1) 即为高维后缀和
                //f[j] = f[j] + f[j ^ (1 << i)]; 一般情况：求真子集和
                if(v[j ^ (1 << i)].first > v[j].first) {
                    v[j].second = v[j].first;
                    v[j].first = v[j ^ (1 << i)].first;
                } else if(v[j ^ (1 << i)].first > v[j].second) {
                    v[j].second = v[j ^ (1 << i)].first;
                }
            }
        }
    }
    int ans = 0;
    for(int i = 1; i < (1 << n); ++i) {
        ans = std::max(ans, v[i].first + v[i].second);
        std::cout << ans << '\n';
    }
    return 0;
}
```

6.5 命令行

```
********** bat ***********
@echo off
g++ %1.cpp -std=c++20 -O2 -Wall -o %1 -D_GLIBCXX_DEBUG
.\%1 < in.txt > out.txt
@REM type out.txt
*************************

********** sh *********** chmod +x
#!/bin/bash
g++ -std=c++20 -O2 -Wall "$1.cpp" -o "$1" -D_GLIBCXX_DEBUG
./"$1" < in.txt > out.txt
cat out.txt
*************************

********** sh *********** chmod +x
while true; do
    ./gen > 1.in
    ./std < 1.in > std.out
    ./my < 1.in > my.out
    if diff my.out std.out; then
        echo ac
    else
        echo wa
    break
    fi
done
*************************
```

7 编译参数

-D_GLIBCXX_DEBUG : STL debugmode
-fsanitize=address : 内存错误检查
-fsanitize=undefined :UB 检查

8 随机素数

979345007 986854057502126921
935359631 949054338673679153
931936021 989518940305146613
984974633 972090414870546877
984858209 956380060632801307

9 常用组合数学公式

性质 1：

$$C_n^m = C_n^{n-m}$$

性质 2：

$$C_{n+m+1}^m = \sum_{i=0}^m C_{n+i}^i$$

性质 3：

$$C_n^m \cdot C_m^r = C_n^r \cdot C_{n-r}^{m-r}$$

性质 4（二项式定理）：

$$\sum_{i=0}^n (C_n^i \cdot x^i) = (1+x)^n$$

$$\sum_{i=0}^n C_n^i = 2^n$$

性质 5：

$$\sum_{i=0}^n ((-1)^i \cdot C_n^i) = 0$$

性质 6：

$$\mathrm{C}_{\mathrm{n}}^{0}+\mathrm{C}_{\mathrm{n}}^{2}+\cdots=\mathrm{C}_{\mathrm{n}}^{1}+\mathrm{C}_{\mathrm{n}}^{3}+\cdots=2^{\mathrm{n}-1}$$

性质 7：

$$\mathrm{C}_{\mathrm{n}+\mathrm{m}}^{\mathrm{r}}=\sum_{i=0}^{\min(\mathrm{n},\mathrm{m},\mathrm{r})}\left(\mathrm{C}_{\mathrm{n}}^{i}\cdot\mathrm{C}_{\mathrm{m}}^{\mathrm{r}-i}\right)$$

$$\mathrm{C}_{\mathrm{n}+\mathrm{m}}^{\mathrm{n}}=\mathrm{C}_{\mathrm{n}+\mathrm{m}}^{\mathrm{m}}=\sum_{i=0}^{\min(\mathrm{n},\mathrm{m})}\left(\mathrm{C}_{\mathrm{n}}^{i}\cdot\mathrm{C}_{\mathrm{m}}^{i}\right),\quad(\mathrm{r}=\mathrm{n}\mid\mathrm{r}=\mathrm{m})$$

性质 8：

$$\mathrm{m}\cdot\mathrm{C}_{\mathrm{n}}^{\mathrm{m}}=\mathrm{n}\cdot\mathrm{C}_{\mathrm{n}-1}^{\mathrm{m}-1}$$

性质 9：

$$\sum_{i=0}^{\mathrm{n}}\left(\mathrm{C}_{\mathrm{n}}^{i}\cdot i^{2}\right)=\mathrm{n}(\mathrm{n}+1)\cdot2^{\mathrm{n}-2}$$

性质 10：

$$\sum_{i=0}^{\mathrm{n}}\left(\mathrm{C}_{\mathrm{n}}^{i}\right)^{2}=\mathrm{C}_{2\mathrm{n}}^{\mathrm{n}}$$

10 常数表

n	$\log_{10} n$	$n!$	$C(n,n/2)$	LCM(1...n)	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1.00000000	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
P_n	37338_{40}	204226_{50}	966467_{60}	190569292_{100}	$1e9_{114}$

$\max\omega(n)$：小于等于 n 中的数最大质因数个数

$\max d(n)$：小于等于 n 中的数最大因数个数

$\pi(n)$：小于等于 n 中的数最大互质数个数

$n\leq$	10	100	1e3	1e4	1e5	1e6
$\max\omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n\leq$	1e7	1e8	1e9	1e10	1e11	1e12
$\max\omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n\leq$	1e13	1e14	1e15	1e16	1e17	1e18
$\max\omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x)\sim\frac{x}{\log(x)}$					