目录

1 基础算法 **1**

1.1 三分 1

2 图论 **1**

2.1 图的连通性 1

2.1.1 Tarjan 割点 1

2.1.2 Tarjan 割边 1

2.1.3 Tarjan 强连通分量 2

2.1.4 Tarjan 点双连通分量 2

2.1.5 Tarjan 边双连通分量 3

2.2 拓扑排序 3

2.3 最小生成树 3

2.3.1 Kruskal 3

2.3.2 Prim 4

2.4 树的重心 4

2.5 流和匹配 4

2.5.1 EdmondsKarp 4

3 数据结构 **5**

3.1 Splay 5

3.2 ST 表 6

3.3 对顶堆 7

3.4 并查集 7

3.5 树状数组 7

3.5.1 树状数组 7

3.5.2 树状数组 2 8

3.6 波纹疾走树 8

3.7 线段树 9

3.7.1 主席树 9

3.7.2 标记永久化主席树 11

3.7.3 线段树 12

3.7.4 线段树优化建图 13

3.8 重链剖分 13

4 数学 **14**

4.1 数论 14

4.1.1 MillerRabin 14

4.1.2 PollardRho 14

4.1.3 区间筛 15

4.1.4 欧拉筛 15

4.2 组合数学 16

4.2.1 卢卡斯定理 16

5 字符串 **16**

5.1 EXKMP 16

5.2 KMP 16

5.3 字符串哈希 17

5.4 马拉车 17

6 计算几何 **18**

6.1 凸包 18

7 杂项 **19**

7.1 康托展开 19

7.2 逆康托展开 19

7.3 高精度 20

7.4 高维前缀和 21

1 基础算法

1.1 三分

```cpp
#include <bits/stdc++.h>
constexpr double eps = 1E-6;//eps控制精度

//三分（实数范围）凸函数
//https://www.luogu.com.cn/record/160695683
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    double l, r;
    std::cin >> n >> l >> r;
    std::vector<double> v(n + 1);
    for(int i = n; i >= 0; --i) {
        std::cin >> v[i];
    }
    auto check = [&](double t) ->double {
        double ans = 0;
        for(int i = 0; i <= n; ++i) {
            ans += v[i] * std::pow(t, i);
        }
        return ans;
    };
    while(l + eps <= r) {
        double lmid = l + (r - l) / 3;//左三分点
        double rmid = r - (r - l) / 3;//右三分点
        if(check(lmid) < check(rmid)) {
            l = lmid;
        } else {
            r = rmid;
        }
    }
    std::cout << l << '\n';
    return 0;
}
```

2 图论

2.1 图的连通性

2.1.1 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;
}
```

2.1.2 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;
}
```

2.1.3 Tarjan 强连通分量

```
#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;
}
```

2.1.4 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;
}
```

```
}
```

2.1.5 Tarjan 边双连通分量

tarjan 求边双连通分量

Link: https://www.luogu.com.cn/problem/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;
}
```

2.2 拓扑排序

```
#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;
}
```

2.3 最小生成树

2.3.1 Kruskal

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

//kruskal算法最小生成树(稀疏图)
//https://www.luogu.com.cn/problem/P3366
class DSU { //维护并查集
public:
    DSU(int n) { //初始构造
        v.resize(n + 1);
        std::iota(v.begin(), v.end(), 0);
    }
    int find(int x) { //找根
        return (v[x] == x ? x : (v[x] = find(v[x])));
    }
    void uniset(int x, int y) { //合并集合
        v[find(x)] = find(y);
    }
    bool query(int x, int y) { //是否在同一集合
        return find(x) == find(y);
    }
private:
    std::vector<int> v;
};

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.query(x, y)) {
            dsu.uniset(x, y);
            ans += w;
            tot++;
        }
    }
    if(tot != n - 1) {
        std::cout << "orz" << '\n';
    } else {
        std::cout << ans << '\n';
    }
    return 0;
}
```

2.3.2 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;
}
```

2.4 树的重心

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;
}
```

2.5 流和匹配

2.5.1 EdmondsKarp

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

struct Edge {
    Edge() = default;
    Edge(int _nxt, int _cap, int _enxt) : nxt(_nxt), cap(_cap), enxt(_enxt)
      {}
    int nxt, cap, enxt;
};
```

```cpp
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m, S, T;
    std::cin >> n >> m >> S >> T;
    std::vector<Edge> e(2 * m);
    std::vector<int> head(n + 1, -1), pre(n + 1); // pre: id的前驱边
    std::vector<int> mf(n + 1); //每S~v的流量上限,
    for(int i = 0; i < 2 * m; i += 2) {
        int x, y, cap;
        std::cin >> x >> y >> cap;
        e[i] = Edge(y, cap, head[x]);
        head[x] = i;
        e[i ^ 1] = Edge(x, 0, head[y]);
        head[y] = i ^ 1;
    }
    auto bfs = [&]() ->bool{
        std::fill(mf.begin(), mf.end(), 0);
        mf[S] = INT_MAX;
        std::queue<int> q;
        q.push(S);
        while(!q.empty()) {
            int id = q.front();
            q.pop();
            for(int eid = head[id]; ~eid; eid = e[eid].enxt) {
                auto &[nxt, cap, _] = e[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;
    };
    auto EK = [&]() ->i64 {
        i64 flow = 0;
        while(bfs()) { //找到增广路
            for(int id = T; id != S; id = e[pre[id] ^ 1].nxt) {
                e[pre[id]].cap -= mf[T];
                e[pre[id] ^ 1].cap += mf[T];
            }
            flow += mf[T];
        }
        return flow;
    };
    std::cout << EK() << '\n';
    return 0;
}
```

3 数据结构

3.1 Splay

```cpp
#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;
}
```

3.2 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

```
#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;
}
```

3.3 对顶堆

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

//对顶堆，维护第k小/大
template<typename T>
struct DoubleHeap {
    DoubleHeap(int _k) : k(_k) {} //第k小，若要第k大，将下面比较函数反转
    std::priority_queue<T, std::vector<T>, std::less<T>> mpq; //大根堆[1, k - 1]
    std::priority_queue<T, std::vector<T>, std::greater<T>> Mpq; //小根堆[k, sz]
    void insert(T x) {
        mpq.push(x);
        while(mpq.size() >= k) {
            Mpq.push(mpq.top());
            mpq.pop();
        }
    }
    T kth() {
        assert(Mpq.empty() == false);
        return Mpq.top();
    }
    const int k;
};

struct MINT {
    int x;
    bool operator<(const MINT &o) const {
        return x < o.x;
    }
    bool operator>(const MINT &o) const {
        return x > o.x;
    }
};

void solve() {
    int n, k;
    std::cin >> n >> k;
    DoubleHeap<MINT> dpq(k);
    for(int i = 1; i <= n; ++i) {
        int opt;
        std::cin >> opt;
        if(opt == 1) {
            int x;
            std::cin >> x;
            dpq.insert({x});
        } else {
            std::cout << dpq.kth().x << '\n';
        }
    }

}

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

3.4 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) { //初始构造
        v.resize(n + 1);
        std::iota(v.begin(), v.end(), 0);
    }
    int find(int x) { //找根
        return (v[x] == x ? x : (v[x] = find(v[x])));
    }
    void merge(int x, int y) { //合并集合
        v[find(x)] = find(y);
    }
    bool query(int x, int y) { //是否在同一集合
        return find(x) == find(y);
    }
    std::vector<int> v;
};

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.query(x, y) ? 'Y' : 'N') << '\n';
        }
    }
    return 0;
}
```

3.5 树状数组

3.5.1 树状数组

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

```cpp
//树状数组(Fenwick)
//https://www.luogu.com.cn/problem/P3374
template<typename T>
struct Fenwick {
    Fenwick(int n) : v(n + 1) {}; //有参构造
    void update(int x, T dx) { //更新(index, dx)
        for(int i = x; i < v.size(); i += (i & -i)) {
            v[i] += dx;
        }
    }
    T query(int x) { //查询前缀和[0, L]
        T res{};
        for(int i = x; i > 0; i -= (i & -i)) {
            res += v[i];
        }
        return res;
    }
    T range(int l, int r) { //查询区间[L, R]
        return query(r) - query(l - 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;
};
```

3.5.2 树状数组 2

```cpp
#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;
}
```

3.6 波纹疾走树

```cpp
#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;
}
```

3.7 线段树

3.7.1 主席树

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

```cpp
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;
}
```

3.7.2 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;
}
```

3.7.3 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(int n) : n(n), info(n << 2), tag(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 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;
}
```

3.7.4 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) {
                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;
}
```

3.8 重链剖分

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

//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;
    std::vector<std::vector<int>> v(n + 1);
    std::vector<int> fa(n + 1), dep(n + 1), son(n + 1), sz(n + 1), top(n + 1,
       0);
    //父节点，深度，重儿子，子树节点数，所在重链的顶点
    for(int i = 0; i < n - 1; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    auto dfs1 = [&](auto self, int id, int lst) ->void {//求fa, dep, son, sz
      数组
        fa[id] = lst;
        dep[id] = dep[lst] + 1;
```

```
        sz[id] = 1;
        for(auto nxt : v[id]) {
            if(nxt == lst) continue;
            self(self, nxt, id);
            sz[id] += sz[nxt];
            if(sz[son[id]] < sz[nxt]) {
                son[id] = nxt;
            }
        }
    };
    auto dfs2 = [&](auto self, int id, int t) ->void {
        top[id] = t;
        if(son[id] == 0) return;
        self(self, son[id], t);
        for(auto nxt : v[id]) {
            if(nxt != fa[id] && nxt != son[id]) {
                self(self, nxt, nxt);
            }
        }
    };
    auto lca = [&](int x, int y) ->int {
        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);
    };
    dfs1(dfs1, s, 0);
    dfs2(dfs2, s, s);
    for(int i = 0; i < m; ++i) {
        int x, y;
        std::cin >> x >> y;
        std::cout << lca(x, y) << '\n';
    }
    return 0;
}
```

4 数学

4.1 数论

4.1.1 MillerRabin

```
#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;
}
```

4.1.2 PollardRho

```
#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;
}

//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;
}
```

4.1.3 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;
}
```

4.1.4 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;
}
```

4.2 组合数学

4.2.1 卢卡斯定理

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

```
#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;
}
```

5 字符串

5.1 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;
}
```

5.2 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;
}
```

5.3 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;
}
```

5.4 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;
}
```

6 计算几何

6.1 凸包

```
#include <bits/stdc++.h>
using i64 = long long;
constexpr long double EPS = 1E-10;

using T = long double;
struct Point {
    T x = 0, y = 0;
    Point operator+(const Point &o) const {return {x + o.x, y + o.y};}
    Point operator-(const Point &o) const {return {x - o.x, y - o.y};}
    Point operator-() const {return {-x, -y};}
    Point operator*(T fac) const {return {x * fac, y * fac};}
    Point operator/(T fac) const {return {x / fac, y / fac};}
    bool operator<(const Point &o) const {
        return std::tie(x, y) < std::tie(o.x, o.y);
    }
    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 << ")";
    }
};

struct Line {
    Point s, t;
    Line() = default;
    Line(Point _s, Point _t) : s(_s), t(_t) {}
};

int sgn(T a){
    if(fabs(a) < EPS) return 0;
    return a > 0 ? 1 : -1;
}

T dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}
T cross(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}
T cross(const Point &a, const Point &b, const Point &c) {
    return cross(b - a, c - a);
}
T len(const Point &a) {
    return sqrtl(a.x * a.x + a.y * a.y);
}
T angle(const Point &a, const Point &b) {
    return acosl(dot(a, b) / len(a) / len(b));
}
T dis2(const Point &a, const Point &b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
T dis(const Point &a, const Point &b) {
    return sqrtl(dis2(a, b));
}
Point rotate(const Point &a, const Point &b, T theta) {
    return {
        (b.x - a.x) * cosl(theta) - (b.y - a.y) * sinl(theta) + a.x,
        (b.x - a.x) * sinl(theta) + (b.y - a.y) * cosl(theta) + a.y
    };
}

bool intersect(const Line &a, const Line &b) {
    return cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t) <= 0
        && cross(b.s, b.t, a.s) * cross(a.s, b.t, a.t) <= 0;
}
bool intersectStrictly(const Line &a, const Line &b) {
    return cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t) < 0
        && cross(b.s, b.t, a.s) * cross(a.s, b.t, a.t) < 0;
}
Point getNode(const Line &a, const Line &b) {
    T dx = cross(b.s, b.t, a.s) / cross(b.s, b.t, a.t);
    return a.s + (a.t - a.s) * std::abs(dx);
};

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

T diameter(const std::vector<Point> &v) {
    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, dis(v[i], v[j]), dis(v[(i + 1) % n], v[j])});
    }
    return res;
}

T diameter2(const std::vector<Point> &v) {
    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;
}

T grith(const std::vector<Point> &convex) {
    long double ans = 0;
    for(int i = 0; i < convex.size(); ++i) {
        ans += dis(convex[i], convex[(i + 1) % convex.size()]);
    }
    return ans;
}

void solve() {
    int n, m;
    std::cin >> n;
    std::vector<Point> A(n);
    for(int i = 0; i < n; ++i) {
        std::cin >> A[i];
    }
    std::cin >> m;
```

```
    std::vector<Point> B(m);
    for(int i = 0; i < m; ++i) {
        std::cin >> B[i];
    }
    long double ans = grith(A) + 2.0L * sqrtl(diameter2(B)) * acosl(-1.0L);
       //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;
}
```

7 杂项

7.1 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;
}
```

7.2 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;
}
```

```
}
```

7.3 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
        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;
        std::cin >> 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;
}
```

7.4 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;
}
```