算法竞赛模板

组织编写：	22 网络-1 肖建华
内容编写：	22 网络-1 肖建华
最后提交日期：	Wednesday 4th June, 2025

计算机学院

2024 年春季学期

声明

本项目中的代码均由以下框架构成，请仔细阅读!!!

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


int main() {


    return 0;
}
```

摘要

该模板由闽南师范大学 2022 级网络一班肖建华主编，同时欢迎同校师生共同维护，其目的为了方便分享以及打印，大部分模板引用于jiangly 算法模板收集。

如有需要，可自取，

望周知！

目录

数据结构专题模板

1.1 普通并查集

```cpp
struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n){
        init(n);
    }

    void init(int n){
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x){
        while(x != f[x]){
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x,int y){
        return find(x) == find(y);
    }

    bool merge(int x,int y){
        x = find(x);
        y = find(y);
        if(x == y){
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x){
        return siz[find(x)];
    }

    int operator[](const int x) {
        return find(x);
    }
};
```

1.2 带权并查集

```cpp
struct WDSU {
    std::vector<int> dist, f;

    WDSU() {}

    WDSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        dist.assign(n, 0);
    }

    int find(int x) {
        while(x != f[x]) {
            int tmp = f[x];
            f[x] = find(tmp);
            dist[x] += dist[tmp];
        }
        return f[x];
    }

    bool merge(int l, int r, int v) {
        int lf = find(l), rf = find(r);
        if(lf != rf) {
            f[lf] = rf;
            dist[lf] = v + dist[r] - dist[l];
        }
    }

    int query(int l, int r) {
        if(find(l) != find(r)) {
            return -1;
        }
        return dist[l] - dist[r];
    }
    int operator[](const int x) {
        return find(x);
    }
};
```

1.3 可撤销并查集

```
struct UndoDSU {
    std::vector<int> f, siz, rank;
    std::stack<std::pair<int, int>> stk;

    DSU() {}
    DSU(int n){
        init(n);
    }

    void init(int n){
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
        rank.assign(n, 0);
    }

    int find(int x){
        while(x != f[x]){
            x = f[x];
        }
        return x;
    }

    bool same(int x,int y){
        return find(x) == find(y);
    }

    bool merge(int x,int y){
        x = find(x);
        y = find(y);
        if(x == y){
            stk.push({-1, -1});
            return false;
        }
        if(rank[x] > rank[y]) {
            std::swap(x, y);
        }

        f[x] = y;
        siz[y] += siz[x];
        stk.push({x, 0});
        if(rank[x] == rank[y]) {
            rank[y]++;
            stk.top().second = 1;
        }

    }

    void rollback() {
        auto v = stk.top();
        stk.pop();
        if(v.first == -1) {
            return;
        }
        int x = v.first, y = f[x];
        rank[y] -= v.second;
        siz[y] -= siz[x];
        f[x] = x;
    }
```

```cpp

    int size(int x){
        return siz[find(x)];
    }
    int operator[](const int x) {
        return find(x);
    }
};
```

1.4 树状数组

```cpp
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    // first p, query(0, p) >= k
    int select(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i - 1] <= k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

1.5 二维树状数组

```cpp
struct BIT_2D {
    using T = long long;
    int n, m;
    std::vector<std::vector<T>> sum[4];
    BIT_2D(int _n, int _m): n(_n), m(_m) {
        for (int i= 0; i < 4; ++i) {
            sum[i].assign(n+1, std::vector<T>(m+1, 0));
        }
    }
    void add(int x, int y, T val) {
        for (int i = x; i <= n; i += i&-i) {
            for (int j = y; j <= m; j += j&-j) {
                sum[0][i][j] += val;
                sum[1][i][j] += val*x;
                sum[2][i][j] += val*y;
                sum[3][i][j] += val*x*y;
            }
        }
    }
    void range_add(int x1, int y1, int x2, int y2, T x) {
        add(x1, y1, x);
        add(x1, y2 + 1, -x);
        add(x2 + 1, y1, -x);
        add(x2 + 1, y2 + 1, x);
    }
    T ask(int x, int y) {
        T res[4]= {};
        for (int i=x; i>0; i-=i&-i)
            for (int j=y; j>0; j-=j&-j)
                for (int k=0; k<4; ++k)
                    res[k]+=sum[k][i][j];
        return (x + 1) * (y + 1) * res[0]-(y + 1) * res[1]-(x +
            1) * res[2] + res[3];
    }
    T range_ask(int x1, int y1, int x2, int y2) {
        return ask(x2, y2) - ask(x1 - 1, y2)- ask(x2, y1 - 1) +
            ask(x1 - 1, y1 - 1);
    }
};
```

1.6 线段树

1.6.1 无 lazy

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
```

```cpp
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) \
            + \
            rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred)
       {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
                return -1;
            }
            if (l >= x && r <= y && !pred(info[p])) {
                return -1;
            }
            if (r - l == 1) {
                return l;
            }
            int m = (l + r) / 2;
            int res = findLast(2 * p + 1, m, r, x, y, pred);
            if (res == -1) {
                res = findLast(2 * p, l, m, x, y, pred);
            }
            return res;
        }
        template<class F>
        int findLast(int l, int r, F &&pred) {
            return findLast(1, 0, n, l, r, pred);
        }
};
```

1.6.2 有 lazy

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
```

```cpp
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p,
           int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) \
                + \
                rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const
       Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
```

1.7 ST 表

```
// vector<int> a(n + 1);
template<typename T>
class SparseTable {
public:
    SparseTable() = default;

    explicit SparseTable(const std::vector<T>& data)
    {
        Initialize(data);
    }

    void Initialize(const std::vector<T>& data) {
        this->n = data.size() - 1;

        log_table.resize(n + 1);
        log_table[0] = -1;
        for(int i = 1; i <= n; i++) {
            log_table[i] = log_table[i >> 1] + 1;
        }

        st_table.resize(n + 1, std::vector<int>(21));

        for(int i = 1; i <= n; i++) {
            st_table[i][0] = data[i];
        }

        for(int p = 1; p <= log_table[n]; p++) {
            for(int i = 1; i + (1 << p) - 1 <= n; i++) {
                st_table[i][p] = op(st_table[i][p - 1],
                    st_table[i + (1 << (p - 1))][p - 1]);
            }
        }

    }

    T Query(size_t left, size_t right) {
        const int k = log_table[right - left + 1];
        return op(st_table[left][k], st_table[right - (1 << k)
            + 1][k]);
    }

private:
    int n;
    std::vector<int> log_table;
    std::vector<std::vector<T>> st_table;

    T op(const T& lv, const T& rv) const {
        return std::max(lv, rv);
    }
};
```

1.8 线性基

```
struct Basis {
    static const int BIT = 21;
    std::vector<int> basis;

    Basis() {
        basis.assign(BIT + 1, 0);
    }

    bool insert(int num) {
        for(int i = BIT; i >= 0; i--) {
            if(num >> i == 1) {
                if(basis[i] == 0) {
                    basis[i] = num;
                    return true;
                }
                num ^= basis[i];
            }
```

```cpp
        }
        return false;
    }

    int max() {
        int ans = 0;
        for(int i = BIT; i>= 0; i--) {
            ans = std::max(ans, ans ^ basis[i]);
        }
        return ans;
    }

};
```

图论专题模板

2.1 链式前向星建图

```cpp
const int N = 2e6 + 10;
struct Enode {
    int next, to, w;
}edges[N];
// 如果是双向图，那么edges中的N为 N << 1;
int head[N], cnt = 0;

void init() {
    std::fill(head, head + N, -1);
    cnt = 0;
}

void addEdge(int u, int v, int w = 1) {
    edges[cnt].to = v;
    edges[cnt].w = w;
    edges[cnt].next = head[u];
    head[u] = cnt++;
}
```

2.2 Dijkstra

```
struct Dijkstra {   // index-base-0
    using i64 = long long;
    int n;
    std::vector<std::vector<std::pair<int, i64>>> adj;
    std::vector<i64> dis;
    std::vector<bool> vis;
    Dijkstra(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        vis.assign(n, false);
        dis.assign(n, 1e18);
    }

    void add_edge(int u, int v, i64 d) {
        adj[u].push_back({v, d});
        adj[v].push_back({u, d});
    }


    struct node {
        i64 dis;
        int pos;
        bool operator <(const node &x) const {
            return x.dis < dis;
        }
    };

    std::priority_queue<node> q;
    void work(int r) {
        dis[r] = 0;
        q.push((node){0, r});

        while(!q.empty()) {
            node tmp = q.top();
            q.pop();
            int u = tmp.pos;
            i64 d = tmp.dis;

            if (vis[u]) {
                continue;
            }

            vis[u] = true;
            for(auto [v, w] : adj[u]) {
                if(dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    q.push((node){dis[v], v});
                }
            }
        }
    }
};
```

2.3 Spfa

```
template<class T = long long>
struct Spfa {   // index-base-0
    static constexpr T inf = std::numeric_limits<T>::max() / 2;

    int n;
    std::vector<std::vector<std::pair<int, T>>> g;
    std::vector<T> dis;

    Spfa() {};

    Spfa(int n) : n(n), g(n) {}

    void add_Edge(int u, int v, T w) {
        g[u].emplace_back({v, w});
    }

    bool work(int root = 0) {
        std::vector<bool> vis(n, false);
        std::vector<int> cnt(n, 0);
        dis.assign(n, inf);
        dis[root] = 0;
        // 最长路
        // dis.assign(n, -inf);
        std::queue<int> q;
        q.push(root);
        vis[root] = true;
        while(q.size()) {
            auto u = q.front();
            q.pop();
            vis[u] = false;
            for(auto [v, w] : g[u]) {
                if(dis[v] > dis[u] + w) { // 最长路 -> v < u + w
                    dis[v] = dis[u] + w;
                    if((cnt[v] = cnt[u] + 1) >= n) {
                        return false;
                    }
                    if(!vis[v]) {
                        q.push(v);
                        vis[v] = true;
                    }
                }
            }
        }
        return true;
    }
};
```

2.4 Floyd

```
const int inf = 1e9;
void floyd(std::vector<std::vector<int>>& dis) {
    int n = dis.size();
    for(int k=0; k<n; ++k)
        for(int i=0; i<n; ++i)
            for(int j=0; j<n; ++j) {
```

```cpp
                if(i!=j && i!=k && j!=k
                && dis[i][k] < inf && dis[k][j] < inf// 边权有负数必加
                && dis[i][j] > dis[i][k] + dis[k][j])
                    dis[i][j] = dis[i][k]+dis[k][j];
            }
}
// 同时记录方案数
void floyd(std::vector<std::vector<int>>& dis, std::vector<std::vector<Z>>& f) {
    int n = dis.size();
    for(int k=0; k<n; ++k)
        for(int i=0; i<n; ++i)
            for(int j=0; j<n; ++j)
                if(i!=j && i!=k && j!=k) {
                    if(dis[i][j] > dis[i][k] + dis[k][j]) {
                        dis[i][j] = dis[i][k] + dis[k][j];
                        f[i][i] = f[i][k] * f[k][j];
                    }else if(dis[i][j] == dis[i][k] + dis[k][j]) {
                        f[i][j] += f[i][k] * f[k][j];
                    }
                }
}
```

2.5 Kruskal

```cpp
template<class T>
struct KruskalMst {      // index-base-0
    int n;
    std::vector<int> f;
    std::vector<std::tuple<T, int, int>> e;
    KruskalMst(int n) {
        this->n = n;
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
    }
    void addEdge(int u, int v, T w) {
        e.emplace_back(w, u, v);
    }
    int find(int u) {
        return f[u] == u ? u : f[u] = find(f[u]);
    }
    T work() {
        std::sort(e.begin(), e.end());
        T ans = 0;
        int cnt = 0;
        for(auto [w, u, v] : e) {
            u = find(u);
            v = find(v);
            if(u == v) {
                continue;
            }
            f[u] = v;
            ans += w;
            if(++cnt == n - 1) {
                break;
            }
        }
        // assert(cnt==n-1);
        return ans;
    }
};
```

2.6 二分图最大匹配

2.6.1 匈牙利算法

```cpp
struct BipartiteGraph {      // index-base-0
    int n, m;
    std::vector<std::vector<int>> g;
    std::vector<int> vis, link;
    BipartiteGraph(int n,int m)
    :n(n),m(m), g(n), vis(m), link(m) {}

    void addEdge(int u, int v) {// left->right
        g[u].push_back(v);
    }
    bool find(int u) {// 左边u -> 右边v
        for(int v: g[u]) {
            if(vis[v]) {
                continue;
            }
            vis[v] = 1;
            if(link[v] == -1 || find(link[v])) {
                link[v] = u;
                return true;
            }
        }
        return false;
    }
    int maxMatching() {
        int cnt = 0;
        std::fill_n(link.begin(), m, -1);
        for(int i = 0; i < n; ++i) {
            if(!find(i)) {
                continue;
            }
            std::fill_n(vis.begin(), m, 0);
            ++cnt;
        }
        return cnt;
    }
};
```

2.6.2 HK 算法

```
struct HopcroftKarp {        // index-base-0
    static constexpr int INF = 0x3f3f3f3f;
    int n, m, dis;
    std::vector<std::vector<int>> e;
    std::vector<int> matchX, matchY, dx, dy;
    std::vector<bool> used;
    HopcroftKarp(int n, int m)
    :n(n), m(m), e(n), matchX(n), matchY(m), dx(n), dy(m), used
       (m) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
    }
    bool searchP() {
        std::fill(dx.begin(), dx.end(), -1);
        std::fill(dy.begin(), dy.end(), -1);
        dis = INF;
        std::queue<int> q;
        for(int i = 0; i < n; ++i) {
            if(matchX[i] == -1) {
                q.push(i), dx[i] = 0;
            }
        }
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            if(dx[u] > dis) {
                break;
            }
            for(int v : e[u])
                if(dy[v] == -1) {
                    dy[v] = dx[u] + 1;
                    if(matchY[v] == -1) {
                        dis = dy[v];
                    }
                    else {
                        dx[matchY[v]] = dy[v] + 1;
                        q.push(matchY[v]);
                    }
                }
        }
        return dis != INF;
    }
    bool dfs(int u) {
        for(int v : e[u]) {
            if(used[v] || dy[v] != dx[u] + 1) {
                continue;
            }
            used[v] = true;
            if(matchY[v] != -1 && dy[v] == dis) {
                continue;
            }
            if(matchY[v] == -1 || dfs(matchY[v])) {
                matchY[v] = u;
                matchX[u] = v;
                return true;
            }
        }
        return false;
    }
    int maxMatching() {
        int res = 0;
        std::fill(matchX.begin(), matchX.end(), -1);
        std::fill(matchY.begin(), matchY.end(), -1);
        while(searchP()) {
            std::fill(used.begin(), used.end(), false);
            for(int i = 0; i < n; ++i) {
                if(matchX[i] == -1 && dfs(i)) {
                    ++res;
                }
            }
        }
        return res;
    }
};
```

2.7 二分图判定

```
struct JudgeBG {    // index-base-0
    int n;
    std::vector<int> bel;    // bel:1 or 2
    std::vector<std::vector<int>> g;
    JudgeBG(int n): n(n),g(n),bel(n) {}
    void addEdge(int u, int v) {
        g[u].push_back(v);
        g[v].push_back(u);
    }
    bool dfs(int u, int color = 1) {
        if(bel[u]) {
            return bel[u]==color;
        }
        bel[u] = color;
        for(int v : g[u])
            if(!dfs(v, 3-color)) {
                return false;
            }
        return true;
    }
    bool paint() {// 二分图染色
        for(int i = 0; i < n; ++i) {
            if(bel[i]) {
                continue;
            }
            if(!dfs(i)) {
                return false;
            }
        }
        return true;
    }
    // std::vector<int> to;//[0,ln), [0,rn)
    // //  use it to divide the graph if paint()=true
    // BipartiteGraph reLabel() {// left->right
    //     int ln = 0,rn = 0;
    //     to.resize(n);
    //     for(int i = 0; i < n; ++i) {
    //         if(bel[i] == 1) {
    //          to[i] = ln++;
    //      }
    //         else {
    //          to[i] = rn++;
    //      }
    //     }
    //     BipartiteGraph bg(ln, rn);
    //     for(int i = 0; i < n; ++i) {
    //         for(int j : g[i]) {
    //             if(bel[i] == 1) {
    //              bg.addEdge(to[i], to[j]);
    //          }
```

```
    //      }
    //  }
    //      return bg;
    // }
};
```

2.8 Kruskal 重构树

```
namespace krt {      // index-1
    constexpr int MAXN = 2e5 + 10, MAXH = 21;
    constexpr bool ASC = true;  // 边权从小到大排序
    int father[MAXN], head[MAXN], next[MAXN], to[MAXN];
    int nodekey[MAXN], dep[MAXN], stjump[MAXN][MAXH];
    int /*dfn[MAXN], */seg[MAXN], in[MAXN], out[MAXN];
    int cntu, cntg, cntd;
    int find(int x) {
        return x == father[x] ? x : father[x] = find(father[x])
            ;
    }

    void addEdge(int u, int v) {
        next[++cntg] = head[u];
        to[cntg] = v;
        head[u] = cntg;
    }

    int lca(int x, int y) {
        if(dep[x] < dep[y]) {
            std::swap(x, y);
        }
        for(int p = MAXH - 1; p >= 0; p--) {
            if(dep[stjump[x][p]] >= dep[y]) {
                x = stjump[x][p];
            }
        }
        if(x == y) {
            return x;
        }
        for(int p = MAXH - 1; p >= 0; p--) {
            if(stjump[x][p] != stjump[y][p]) {
                x = stjump[x][p];
                y = stjump[y][p];
            }
        }
        return stjump[x][0];
    }

    void build(int n, std::vector<std::tuple<int, int, int>> e)
        {
        for(int i = 1; i <= n; i++) {
            father[i] = i;
        }
        if constexpr (ASC) {
            std::sort(e.begin(), e.end());
        } else {
            std::sort(e.begin(), e.end(), std::greater());
        }
        cntu = n;
        for(auto& [w, u, v] : e) {
            int fx = find(u);
            int fy = find(v);
            if(fx != fy) {
                father[fx] = father[fy] = ++cntu;
                father[cntu] = cntu;
                nodekey[cntu] = w;
                addEdge(cntu, fx);
                addEdge(cntu, fy);
            }
        }
        std::function<void(int, int)> dfs1 = [&](int u, int fa)
            -> void {
            dep[u] = dep[fa] + 1;
            in[u] = ++cntd;
            // dfn[u] = ++cntd;
            seg[cntd] = u;
            stjump[u][0] = fa;
            for(int i = 1; i < MAXH; i++) {
                stjump[u][i] = stjump[stjump[u][i - 1]][i - 1];
```

```cpp
            }
            for(int e = head[u]; e > 0; e = next[e]) {
                dfs1(to[e], u);
            }
            out[u] = cntd;
        };
        for(int i = 1; i <= cntu; i++) {
            if(i == father[i]) {
                dfs1(i, 0);
            }
        }
    }
    // 在lim的限制下最高能跳到哪一个节点
    int jumpUp(int u, int lim) {
        for(int i = MAXH - 1; i >= 0; --i) {
            if constexpr (ASC) {
                if(stjump[i][u] and nodekey[stjump[i][u]]<=lim) {
                    u = stjump[i][u];
                }
            } else {
                if(stjump[i][u] and nodekey[stjump[i][u]]>=lim) {
                    u = stjump[i][u];
                }
            }
        }
        return u;
    }
}
```

2.9 SCC 点-双连通分量

```cpp
// index-base-0
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

2.10 EBCC 边-双连通分量

```
std::set<std::pair<int, int>> E;

struct EBCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    EBCC() {}
    EBCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                E.emplace(x, y);
                dfs(y, x);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
                E.emplace(x, y);
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        dfs(0, -1);
        return bel;
    }

    struct Graph {
        int n;
        std::vector<std::pair<int, int>> edges;
        std::vector<int> siz;
        std::vector<int> cnte;
    };
    Graph compress() {
        Graph g;
        g.n = cnt;
        g.siz.resize(cnt);
        g.cnte.resize(cnt);
        for (int i = 0; i < n; i++) {
            g.siz[bel[i]]++;
            for (auto j : adj[i]) {
                if (bel[i] < bel[j]) {
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

2.11 MaxFlow

```
template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };
    int n;
    std::vector<_Edge> e;
```

```cpp
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;
    MaxFlow() {}
    MaxFlow(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }
    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
    }
    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }
    std::vector<bool> minCut() {
        std::vector<bool> c(n);
        for (int i = 0; i < n; i++) {
            c[i] = (h[i] != -1);
        }
        return c;
    }


    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

2.12 Minflow

```
template<class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_),
           cost(cost_) {}
    };
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::make_pair(flow, cost);
    }
    struct Edge {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.cost = e[i].cost;
            x.flow = e[i + 1].cap;
```

```cpp
                a.push_back(x);
            }
            return a;
        }
};
```

2.13 可行流/最大流

```cpp
struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<i64> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<i64>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<i64, int>, std::vector<
           std::pair<i64, int>>, std::greater<std::pair<i64,
           int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            i64 d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] < d) continue;
            for (int i : g[u]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<i64>::max();
    }
    MCFGraph(int n) : n(n), g(n) {}
    //可行流
    void addEdge(int u, int v, int c, int f) {
        if (f < 0) {
            g[u].push_back(e.size());
            e.emplace_back(v, 0, f);
            g[v].push_back(e.size());
            e.emplace_back(u, c, -f);
        } else {
            g[u].push_back(e.size());
            e.emplace_back(v, c, f);
            g[v].push_back(e.size());
            e.emplace_back(u, 0, -f);
        }
    }
    //最大流
    /*void addEdge(int u, int v, int c, int f) {
    g[u].push_back(e.size());
    e.emplace_back(v, c, f);
    g[v].push_back(e.size());
    e.emplace_back(u, 0, -f);
    }*/
    std::pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            int aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug =
                std::min(aug, e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
        }
        return std::make_pair(flow, cost);
    }
};
```

2.14 TwoSat

```
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 *
            n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = std::min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
```

```cpp
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    std::vector<bool> answer() { return ans; }
};
```

树论专题模板

3.1 BlockCutTree

```cpp
struct BlockCutTree {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> dfn, low, stk;
    int cnt, cur;
    std::vector<std::pair<int, int>> edges;

    BlockCutTree() {}
    BlockCutTree(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        stk.clear();
        cnt = cur = 0;
        edges.clear();
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int x) {
        stk.push_back(x);
        dfn[x] = low[x] = cur++;

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
                if (low[y] == dfn[x]) {
                    int v;
                    do {
                        v = stk.back();
                        stk.pop_back();
                        edges.emplace_back(n + cnt, v);
                    } while (v != y);
                    edges.emplace_back(x, n + cnt);
                    cnt++;
                }
            } else {
                low[x] = std::min(low[x], dfn[y]);
            }
        }
    }

    std::pair<int, std::vector<std::pair<int, int>>> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                stk.clear();
                dfs(i);
            }
        }
        return {cnt, edges};
    }
};
```

3.2 树的重心

```
struct CodeOfTree {
    int n, cur;
    std::vector<std::vector<int>> adj;
    std::vector<int> siz, dep, p, in, ord;
    CodeOfTree() = default;

    CodeOfTree(int _n) {
        init(_n);
    }

    void init(int _n) {
        this->n = _n;
        cur = 0;
        adj.assign(n, {});
        siz.resize(n);
        dep.resize(n);
        p.resize(n);
        in.resize(n);
        ord.resize(n);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int u) {
        siz[u] = 1;
        in[u] = cur++;
        ord[in[u]] = u;
        for(auto v : adj[u]) {
```

```cpp
                if(v == p[u]) {
                    continue;
                }
                p[v] = u;
                dep[v] = dep[u] + 1;
                dfs(v);
                siz[u] += siz[v];
            }
        }

        int find(int u) {
            for(auto v : adj[u]) {
                if(v == p[u] || 2 * siz[v] <= n) {
                    continue;
                }
                return find(v);
            }
            return u;
        }

        void work() {
            p[0] = -1;
            dfs(0);
            int rt = find(0);
            dep[rt] = 0;
            p[rt] = -1;
            cur = 0;
            dfs(rt);
        }
};
```

3.3 树的直径

```cpp
struct Diameter {
    int n, start, end;
    std::vector<std::vector<int>> adj;
    std::vector<int> pa;

    Diameter() = default;
    Diameter(int _n) {
        init(_n);
    }

    void init(int _n) {
        this->n = _n;
        start = end = -1;
        adj.assign(n, {});
        pa.resize(n);
        std::iota(pa.begin(), pa.end(), 0);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    std::pair<int, int> dfs(int u, int fa, std::vector<bool>&
       vis) {
        std::pair<int, int> ans{1, u};
        pa[u] = fa;
        vis[u] = true;
        for(auto v : adj[u]) {
            if(v == fa || vis[v]) {
                continue;
            }
            auto pi = dfs(v, u, vis);
            pi.first += 1;
            ans = max(ans, pi);
        }
        return ans;
    }

    std::pair<int, int> work() {
        std::vector<bool> vis(n);
        auto [d1, ss] = dfs(0, -1, vis);
        auto [d2, ee] = dfs(ss, -1, vis);
        start = ss, end = ee;
        return std::make_pair(start, end);
    }

    /* default = end -> start */
    std::vector<int> getPath() {
        std::vector<int> v;
        int cur = end;
        while(cur != -1) {
            v.push_back(cur);
            cur = pa[cur];
        }
        // std::reverse(v.begin(), v.end());
        return v;
    }
};
```

3.4 笛卡尔树

```
template<class T>
struct DescartesTree {

    std::vector<int> lch, rch, stk;

    DescartesTree() {}

    DescartesTree(const std::vector<T>& v) {
        work(v);
    }

    int work(const std::vector<T>& v) {
        int n = v.size();
        lch.resize(n, n);
        rch.resize(n, n);
        for(int i = 0; i < n; i++) {
            while(!stk.empty() && v[i] < v[stk.back()]) {
                rch[stk.back()] = lch[i];
                lch[i] = stk.back();
                stk.pop_back();
            }
            stk.push_back(i);
        }
        while(stk.size() > 1) {
            int x = stk.back();
            stk.pop_back();
            rch[stk.back()] = x;
        }
        return stk.back();
    }
};
```

3.5 HLD

```
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq, son;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        son.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end()
               , parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                son[u] = v;
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    /*
     * 判断u是否是v的祖先
    */
    bool isAncester(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }

    int rootedParent(int u, int v) {
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
        }
        auto it = std::upper_bound(adj[u].begin(), adj[u].end()
           , v, [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }


    /*
     * 返回在以 v 为根时，节点 u 的子树大小。
    */
    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }

    std::veector<std::pair<int, int>> get_path(int u, int v) {
        std::vector<std::pair<int, int>> v1, v2;
        while(top[u] != top[v]) {
            if(dep[top[u]] > dep[top[v]]) {
                v1.push_back({dfn[u], dfn[top[u]]});
                u = parent[top[u]];
            } else {
                v2.push_back({dfn[top[v], dfn[v]]});
                v = parent[top[v]];
            }
        }
        v1.reserve(v1.size() + v2.size() + 1);
        v1.push_back({dfn[u], dfn[v]});
        reverse(v2.begin(), v2.end());
        for(auto v : v2) {
            v1.push_back(v);
        }
        return v1;
    }
};
```

3.6 树上差分

注意！由于这部分内容大部分与树上 *LCA* 高度重合，因此在此作简要说明！
树上 *LCA* 问题可以使用 *HLD* 或者倍增表实现!!!
树上点的编号从 1 开始，并且我们认为 1 的父亲节点是编号 0!!!

3.6.1 点差分

```
// modify
int u, v;
cin >> u >> v;
lca = lca(u, v);
lcafather = stjump[lca][0];
num[u]++;
num[v]++;
num[lca]--;
num[lcafather]--;

// calc
auto dfs = [&](this auto&&self, int u, int fa) -> void {
    for(auto v : adj[u]) {
        if(v != fa) {
            self(v, u);
            num[u] += num[v];
        }
    }
};
dfs(dfs, 1, 0);
```

3.6.2 边差分

```
// modify
```

```cpp
int u, v;
cin >> u >> v;
lca = lca(u, v);
lcafather = stjump[lca][0];
num[u]++;
num[v]++;
num[lca]-=2;

// calc
auto dfs = [&](this auto&&self, int u, int fa) -> void {
    for(auto v : adj[u]) {
        if(v != fa) {
            self(v, u);
            num[u] += num[v];
        }
    }
};
dfs(dfs, 1, 0);
```

3.7 Splay

```cpp
class Splay {
public:
    Splay() {}

    Splay(int n) {
        init(n);
    }

    void init(int n) {
        cnt = head = 0;
        Tree.assign(n + 5, {});
    }

    // add a num to the Tree
    void add(int num) {
        add(head, num);
    }

    // find the rank`s node in the Tree`s inorder
    int find(int rank) {
        return find(head, rank);
    }

    // query the num`s rank in the Tree
    int rank(int num) {
        return rank(head, num);
    }

    // return x-th`s value after sorting
    int index(int x) {
        int i = find(x);
        splay(i, 0);
        return Tree[i].key;
    }

    // return the pre num`s value
    int pre(int num) {
        return pre(head, num);
    }

    // return the post num`s value
    int post(int num) {
        return post(head, num);
    }

    // remove a num from the Tree
    void remove(int num) {
        int kth = rank(num);
        if(kth != rank(num + 1)) {
            int i = find(kth);
            splay(i, 0);
            if(Tree[i].ls == 0) {
                head = Tree[i].rs;
            } else if(Tree[i].rs == 0) {
                head = Tree[i].ls;
            } else {
                int j = find(kth + 1);
                splay(j, i);
                Tree[j].ls = Tree[i].ls;
                Tree[Tree[j].ls].father = j;
                up(j);
                head = j;
            }
            Tree[head].father = 0;
        }
    }

private:

    // Summary all the info about node[i]`s son
    void up(int i) {
        Tree[i].size = Tree[Tree[i].ls].size + Tree[Tree[i].rs
          ].size + 1;
    }

    // check i is the right child of its father
    int lr(int i) {
        return Tree[Tree[i].father].rs == i ? 1 : 0;
    }

    void rotate(int i) {
        int f = Tree[i].father, g = Tree[f].father;
        int soni = lr(i), sonf = lr(f);
        if(soni == 1) {
            Tree[f].rs = Tree[i].ls;
            if(Tree[f].rs != 0) {
                Tree[Tree[f].rs].father = f;
            }
            Tree[i].ls = f;
        } else {
            Tree[f].ls = Tree[i].rs;
            if(Tree[f].ls != 0) {
                Tree[Tree[f].ls].father = f;
            }
            Tree[i].rs = f;
        }
        if(g != 0) {
            if(sonf == 0) {
                Tree[g].ls = i;
            } else {
                Tree[g].rs = i;
            }
        }
        Tree[i].father = g;
        Tree[f].father = i;
        up(f);
        up(i);
    }

    // make node[i] is a child of node[goal]
    void splay(int i, int goal) {
        int f = Tree[i].father, g = Tree[f].father;
        while(f != goal) {
            if(g != goal) {
                if(lr(i) == lr(f)) {
                    rotate(f);
                } else {
                    rotate(i);
                }
            }
            rotate(i);
            f = Tree[i].father;
            g = Tree[f].father;
        }
        if(goal == 0) {
            head = i;
        }
    }

    void add(int i, int num) {
        Tree[++cnt].key = num;
        Tree[cnt].size = 1;
        if(head == 0) {
            head = cnt;
        } else {
            int f = 0, son = 0;
            while(i != 0) {
                f = i;
                if(Tree[i].key <= num) {
                    son = 1;
                    i = Tree[i].rs;
                } else {
                    son = 0;
                    i = Tree[i].ls;
                }
            }
            if(son == 0) {
                Tree[f].ls = cnt;
            } else {
                Tree[f].rs = cnt;
            }
            Tree[cnt].father = f;
            splay(cnt, 0);
        }
    }

    int find(int i, int rank) {
        while(i != 0) {
            if(Tree[Tree[i].ls].size + 1 == rank) {
                return i;
            } else if(Tree[Tree[i].ls].size >= rank) {
                i = Tree[i].ls;
            } else {
                rank -= Tree[Tree[i].ls].size + 1;
                i = Tree[i].rs;
            }
        }
        return 0;
    }

    int rank(int i, int num) {
        int ans = 0, f = 0;
        while(i != 0) {
            f = i;
            if(Tree[i].key >= num) {
                i = Tree[i].ls;
            } else {
                ans += Tree[Tree[i].ls].size + 1;
                i = Tree[i].rs;
            }
        }
        splay(f, 0);
        return ans + 1;
    }

    int pre(int i, int num) {
        int last = head;
        int ans = std::numeric_limits<int>::min();
        while(i != 0) {
            last = i;
            if(Tree[i].key < num) {
                ans = std::max(ans, Tree[i].key);
                i = Tree[i].rs;
            } else {
                i = Tree[i].ls;
            }
        }
        splay(last, 0);
        return ans;
    }

    int post(int i, int num) {
        int last = head;
        int ans = std::numeric_limits<int>::max();
        while(i != 0) {
            last = i;
            if(Tree[i].key > num) {
                ans = std::min(ans, Tree[i].key);
                i = Tree[i].ls;
            } else {
                i = Tree[i].rs;
            }
        }
        splay(last, 0);
        return ans;
    }

    struct Node {
        int ls, rs, size, father, key;
    };

    int cnt, head;
    std::vector<Node> Tree;
};
```

数学专题模板

4.1 Exgcd

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

4.2 Frac

```
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        if (den < 0) {
            num = -num;
            den = -den;
        }
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.num, a.den);
    }
    friend bool operator==(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator!=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator<(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den < rhs.num * lhs.den;
    }
    friend bool operator>(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den > rhs.num * lhs.den;
    }
    friend bool operator<=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den <= rhs.num * lhs.den;
    }
    friend bool operator>=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den >= rhs.num * lhs.den;
    }
    friend std::ostream &operator<<(std::ostream &os, Frac x) {
        T g = std::gcd(x.num, x.den);
        if (x.den == g) {
            return os << x.num / g;
        } else {
            return os << x.num / g << "/" << x.den / g;
        }
    }
};
```

4.3 ModInt

```
template<class T>
constexpr T power(T a, u64 b, T res = 1) {
    for (; b != 0; b /= 2, a *= a) {
        if (b & 1) {
                res *= a;
            }
        }
        return res;
}

template<u32 P>
constexpr u32 mulMod(u32 a, u32 b) {
    return u64(a) * b % P;
}

template<u64 P>
constexpr u64 mulMod(u64 a, u64 b) {
    u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
    res %= P;
    return res;
}

constexpr i64 safeMod(i64 x, i64 m) {
    x %= m;
    if (x < 0) {
        x += m;
    }
    return x;
}

constexpr std::pair<i64, i64> invGcd(i64 a, i64 b) {
    a = safeMod(a, b);
    if (a == 0) {
        return {b, 0};
    }

    i64 s = b, t = a;
    i64 m0 = 0, m1 = 1;

    while (t) {
        i64 u = s / t;
        s -= t * u;
        m0 -= m1 * u;
```

```cpp

        std::swap(s, t);
        std::swap(m0, m1);
    }

    if (m0 < 0) {
        m0 += b / s;
    }

    return {s, m0};
}

template<std::unsigned_integral U, U P>
struct ModIntBase {
public:
    constexpr ModIntBase() : x(0) {}
    template<std::unsigned_integral T>
    constexpr ModIntBase(T x_) : x(x_ % mod()) {}
    template<std::signed_integral T>
    constexpr ModIntBase(T x_) {
        using S = std::make_signed_t<U>;
        S v = x_ % S(mod());
        if (v < 0) {
            v += mod();
        }
        x = v;
    }

    constexpr static U mod() {
        return P;
    }

    constexpr U val() const {
        return x;
    }

    constexpr ModIntBase operator-() const {
        ModIntBase res;
        res.x = (x == 0 ? 0 : mod() - x);
        return res;
    }

    constexpr ModIntBase inv() const {
        return power(*this, mod() - 2);
    }

    constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
        x = mulMod<mod()>(x, rhs.val());
        return *this;
    }
    constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
        x += rhs.val();
        if (x >= mod()) {
            x -= mod();
        }
        return *this;
    }
    constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
        x -= rhs.val();
        if (x >= mod()) {
            x += mod();
        }
        return *this;
    }
    constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
        return *this *= rhs.inv();
    }

    friend constexpr ModIntBase operator*(ModIntBase lhs, const
        ModIntBase &rhs) {
        lhs *= rhs;
        return lhs;
    }
    friend constexpr ModIntBase operator+(ModIntBase lhs, const
        ModIntBase &rhs) {
        lhs += rhs;
        return lhs;
    }
    friend constexpr ModIntBase operator-(ModIntBase lhs, const
        ModIntBase &rhs) {
        lhs -= rhs;
        return lhs;
    }
    friend constexpr ModIntBase operator/(ModIntBase lhs, const
        ModIntBase &rhs) {
        lhs /= rhs;
        return lhs;
    }

    friend constexpr std::istream &operator>>(std::istream &is,
        ModIntBase &a) {
        i64 i;
        is >> i;
        a = i;
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os,
        const ModIntBase &a) {
        return os << a.val();
    }

    friend constexpr std::strong_ordering operator<=>(
       ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() <=> rhs.val();
    }

private:
    U x;
};

template<u32 P>
using ModInt = ModIntBase<u32, P>;
template<u64 P>
using ModInt64 = ModIntBase<u64, P>;

struct Barrett {
public:
    Barrett(u32 m_) : m(m_), im((u64)(-1) / m_ + 1) {}

    constexpr u32 mod() const {
        return m;
    }

    constexpr u32 mul(u32 a, u32 b) const {
        u64 z = a;
        z *= b;

        u64 x = u64((u128(z) * im) >> 64);

        u32 v = u32(z - x * m);
        if (m <= v) {
            v += m;
        }
        return v;
    }

private:
    u32 m;
    u64 im;
};

template<u32 Id>
struct DynModInt {
public:
    constexpr DynModInt() : x(0) {}
    template<std::unsigned_integral T>
    constexpr DynModInt(T x_) : x(x_ % mod()) {}
    template<std::signed_integral T>
    constexpr DynModInt(T x_) {
        int v = x_ % int(mod());
        if (v < 0) {
            v += mod();
        }
        x = v;
    }

```

```cpp
    constexpr static void setMod(u32 m) {
        bt = m;
    }

    static u32 mod() {
        return bt.mod();
    }

    constexpr u32 val() const {
        return x;
    }

    constexpr DynModInt operator-() const {
        DynModInt res;
        res.x = (x == 0 ? 0 : mod() - x);
        return res;
    }

    constexpr DynModInt inv() const {
        auto v = invGcd(x, mod());
        assert(v.first == 1);
        return v.second;
    }

    constexpr DynModInt &operator*=(const DynModInt &rhs) & {
        x = bt.mul(x, rhs.val());
        return *this;
    }
    constexpr DynModInt &operator+=(const DynModInt &rhs) & {
        x += rhs.val();
        if (x >= mod()) {
            x -= mod();
        }
        return *this;
    }
    constexpr DynModInt &operator-=(const DynModInt &rhs) & {
        x -= rhs.val();
        if (x >= mod()) {
            x += mod();
        }
        return *this;
    }
    constexpr DynModInt &operator/=(const DynModInt &rhs) & {
        return *this *= rhs.inv();
    }

    friend constexpr DynModInt operator*(DynModInt lhs, const
       DynModInt &rhs) {
        lhs *= rhs;
        return lhs;
    }
    friend constexpr DynModInt operator+(DynModInt lhs, const
       DynModInt &rhs) {
        lhs += rhs;
        return lhs;
    }
    friend constexpr DynModInt operator-(DynModInt lhs, const
       DynModInt &rhs) {
        lhs -= rhs;
        return lhs;
    }
    friend constexpr DynModInt operator/(DynModInt lhs, const
       DynModInt &rhs) {
        lhs /= rhs;
        return lhs;
    }

    friend constexpr std::istream &operator>>(std::istream &is,
        DynModInt &a) {
        i64 i;
        is >> i;
        a = i;
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os,
        const DynModInt &a) {
        return os << a.val();
    }
```

```cpp

    friend constexpr std::strong_ordering operator<=>(DynModInt
        lhs, DynModInt rhs) {
        return lhs.val() <=> rhs.val();
    }

private:
    u32 x;
    static Barrett bt;
};

constexpr int MOD = 1'000'000'007;
// constexpr int MOD = 998'244'353;

template<u32 Id>
Barrett DynModInt<Id>::bt = MOD;

using Z = ModInt<MOD>;
```

4.4 Sieve

```cpp
// 输入: n - 筛的范围上限
// 输出:
//   primes        - 质数集合
//   min_prime     - 每个数的最小质因子
//   phi          - 欧拉函数值
//   mu           - 莫比乌斯函数值
//   d            - 约数个数
//   cnt_min_p    - 最小质因子的次数
std::vector<int> primes, min_prime, phi, mu, d, cnt_min_p;
std::vector<bool> is_prime;
void euler_sieve(int n) {
    min_prime.resize(n + 1, 0);
    phi.resize(n + 1, 0);
    mu.resize(n + 1, 0);
    d.resize(n + 1, 0);
    cnt_min_p.resize(n + 1, 0);
    is_prime.resize(n + 1, true);
    phi[1] = mu[1] = d[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(is_prime[i]) {
            primes.push_back(i);
            min_prime[i] = i;
            phi[i] = i - 1;
            mu[i] = -1;
            d[i] = 2;
            cnt_min_p[i] = 1;
        }
        for(auto p : primes) {
            int num = i * p;
            if(num > n) {
                break;
            }
            is_prime[num] = false;
            min_prime[num] = p;
            if(i % p == 0) {
                phi[num] = phi[i] * p;
                mu[num] = 0;
                cnt_min_p[num] = cnt_min_p[i] + 1;
                d[num] = d[i] / (cnt_min_p[i] + 1) * (cnt_min_p
                   [i] + 1);
                break;
            } else {
                phi[num] = phi[i] * (p - 1);
                mu[num] = -mu[i];
                cnt_min_p[num] = 1;
                d[num] = d[i] * 2;
            }
        }
    }
}
```

4.5 Comb(结合 ModInt)

```
struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z C(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
```

```cpp
    }
    Z A(int n, int m) {
        if(n < m || m < 0) return 0;
        return fac(n) * invfac(n - m);
    }
} comb;
```

4.6 BigInt

```cpp
constexpr int BASE = 100000000;
constexpr int MAX_LENGTH = 1000;
constexpr int NUM_DIGIT = 8;
i64 mul_mod (i64 x, i64 y, i64 n){
    i64 T = std::floor(std::sqrt(n) + 0.5);
    i64 t = T * T - n;
    i64 a = x / T; i64 b = x % T;
    i64 c = y / T; i64 d = y % T;
    i64 e = a * c / T; i64 f = a * c % T;
    i64 v = ((a * d + b * c) % n + e * t) % n;
    i64 g = v / T; i64 h = v % T;
    i64 ans = (((f + g) * t % n + b * d) % n + h * T) % n;
    while (ans < 0) {
        ans += n;
    }
    return ans;
}
struct BigInt {

    BigInt(const char* str = "0") {
        (*this) = str;
    }

    BigInt operator=(const char* str) {
        int j = std::strlen(str) - 1;
        len = j / NUM_DIGIT + 1;
        for(int i = 0; i <= len; i++) {
            s[i] = 0;
        }
        for(int i = 0; i <= j; i++) {
            int k = (j - i) / NUM_DIGIT + 1;
            s[k] = s[k] * 10 + (str[i] - '0');
        }
        return *this;
    }

    void print() {
        printf("%d", s[len]);
        for(int i = len - 1; i >= 1; i--) {
            printf("%0*d", NUM_DIGIT, s[i]);
        }
    }

    int len;
    int s[MAX_LENGTH];
};

// > return > 0
// < return < 0
// = return = 0
int compare(const BigInt &lhs, const BigInt &rhs) {
    if(lhs.len > rhs.len) {
        return 1;
    }
    if(lhs.len < rhs.len) {
        return -1;
    }
    int cur = lhs.len;
    while((cur > 1) && (lhs.s[cur] == rhs.s[cur])) {
        cur--;
    }
    return lhs.s[cur] - rhs.s[cur];
}

bool operator<(const BigInt &lhs, const BigInt &rhs) {
    return compare(lhs, rhs) < 0;
}

bool operator<=(const BigInt &lhs, const BigInt &rhs) {
    return compare(lhs, rhs) <= 0;
}

bool operator>(const BigInt &lhs, const BigInt &rhs) {
    return compare(lhs, rhs) > 0;
}

bool operator>=(const BigInt &lhs, const BigInt &rhs) {
    return compare(lhs, rhs) >= 0;
}

bool operator==(const BigInt &lhs, const BigInt &rhs) {
    return compare(lhs, rhs) == 0;
}

bool operator!=(const BigInt &lhs, const BigInt &rhs) {
    return compare(lhs, rhs) != 0;
}


BigInt operator+(const BigInt &lhs, const BigInt &rhs) {
    BigInt ret;
    int i;
    for(i = 1; i <= lhs.len || i <= rhs.len || ret.s[i]; i++) {
        if(i <= lhs.len) {
            ret.s[i] += lhs.s[i];
        }
        if(i <= rhs.len) {
            ret.s[i] += rhs.s[i];
        }
        ret.s[i + 1] = ret.s[i] / BASE;
        ret.s[i] %= BASE;
    }
    ret.len = i - 1;
    if(ret.len == 0) {
        ret.len = 1;
    }
    return ret;
}
```

```

// lhs > rhs
BigInt operator-(const BigInt &lhs, const BigInt &rhs) {
    BigInt ret;
    for(int i = 1, j = 0; i <= lhs.len; i++) {
        ret.s[i] = lhs.s[i] - j;
        if(i <= rhs.len) {
            ret.s[i] -= rhs.s[i];
        }
        if(ret.s[i] < 0) {
            j = 1;
            ret.s[i] += BASE;
        } else {
            j = 0;
        }
    }
    ret.len = lhs.len;
    while(ret.len > 1 && !ret.s[ret.len]) {
        ret.len--;
    }
    return ret;
}

BigInt operator*(const BigInt &lhs, const BigInt &rhs) {
    BigInt ret;
    i64 g = 0;
    ret.len = lhs.len + rhs.len;
    ret.s[0] = 0;
    for(int i = 1; i <= ret.len; i++) {
        ret.s[i] = 0;
    }
    for(int k = 1; k <= ret.len; k++) {
        i64 tmp = g;
        int j = k + 1 - rhs.len;
        if(j < 1) {
            j = 1;
        }
        for(; j <= k && j <= lhs.len; j++) {
            tmp += (i64)lhs.s[j] * (i64)rhs.s[k + 1 - j];
        }
        g = tmp / BASE;
        ret.s[k] = tmp % BASE;
    }
    while(ret.len > 1 && !ret.s[ret.len]) {
        ret.len--;
    }
    return ret;
}
BigInt operator/(const BigInt &lhs, int num) {
    i64 g = 0;
    BigInt ret;
    ret.len = lhs.len;
    for(int i = lhs.len; i > 0; i--) {
        i64 tmp = g * BASE + lhs.s[i];
        ret.s[i] = tmp / num;
        g = tmp % num;
    }
    while(ret.len > 1 && ! ret.s[ret.len]) {
        ret.len--;
    }
    return ret;
}
BigInt operator/(const BigInt &lhs, const BigInt &rhs) {
    BigInt l = "0", r = lhs;
    while(l < r) {
        BigInt m = l + (r - l + "1") / 2;
        if(m * rhs <= lhs) {
            l = m;
        } else {
            r = m - "1";
        }
    }
    return l;
}

i64 BigMod(const BigInt &a, i64 m) {
    i64 d = 0;
    for(int i = a.len; i > 0; i--) {
        d = mul_mod(d, BASE, m);
        d = (d + a.s[i]) % m;
    }
    return d;
}
BigInt sqrt(const BigInt &a) {
    BigInt x, y = a;
    do {
        x = y;
        y = (x + a / x) / 2;
    }while(y < x);

    return x;
}
BigInt gcd(BigInt a, BigInt b) {
    BigInt c = "1";
    while(true) {
        if(a == b) {
            return a * c;
        } else if(a.s[1] % 2 == 0 && b.s[1] % 2 == 0) {
            a = a / 2;
            b = b / 2;
            c = c * "2";
        } else if(a.s[1] % 2 == 0) {
            a = a / 2;
        } else if(b.s[1] % 2 == 0) {
            b = b / 2;
        } else if(b < a) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
}
```

4.7 Miller-RabinAndPollard-Rho

```
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
```

```cpp
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19,
       23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
std::vector<i64> factorize(i64 n) {
    std::vector<i64> p;
    std::function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
```

```
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
        if (isprime(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {
            return (mul(x, x, n) + 1) % n;
        };
        i64 x0 = 2;
        while (true) {
            i64 x = x0;
            i64 y = x0;
            i64 d = 1;
            i64 power = 1, lam = 0;
            i64 v = 1;
            while (d == 1) {
                y = g(y);
                ++lam;
                v = mul(v, std::abs(x - y), n);
                if (lam % 127 == 0) {
                    d = std::gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = std::gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    std::sort(p.begin(), p.end());
    return p;
}
```

4.8 矩阵

```
struct Matrix {
    int n, m, mod;
    bool ok;
    std::vector<std::vector<i64>> a;

    void clear() {
        a.assign(n + 1, std::vector<i64>(m + 1, 0));
    }
    Matrix() : n(0), m(0), ok(false) {
        clear();
    }
    Matrix(int _n) : n(_n), m(_n), ok(false) {
        clear();
    }
    Matrix(int _n, int _m) : n(_n), m(_m), ok(false) {
        clear();
    }

    void setMod(int _m) {
        mod = _m;
    }

    void init() {
        clear();
        for (int i = 1; i <= n; i++) {
            a[i][i] = 1;
        }
    }
    void reset(int _n = 0, int _m = 0) {
        n = _n, m = _m;
        if (m == 0) {
            m = n;
        }
        clear();
    }

    Matrix operator+(const Matrix &b) const {
        Matrix res(n, m);
        for (int i = 1; i <= res.n; i++) {
            for (int j = 1; j <= res.m; j++) {
                res.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
            }
        }
        return res;
    }
    Matrix operator-(const Matrix &b) const {
        Matrix res(n, m);
        for (int i = 1; i <= res.n; i++) {
            for (int j = 1; j <= res.m; j++) {
                res.a[i][j] = (((a[i][j] - b.a[i][j]) % mod) +
                   mod) % mod;
            }
        }
        return res;
    }
    Matrix operator*(const Matrix &b) const {
        Matrix res(n, b.m);
        for (int i = 1; i <= res.n; i++) {
            for (int j = 1; j <= res.m; j++) {
                for (int k = 1; k <= m; k++) {
                    res.a[i][j] = (res.a[i][j] + a[i][k] * b.a[
                       k][j]) % mod;
                }
            }
        }
        return res;
    }
    Matrix operator/(Matrix b) const {
        auto c = b.getinv();
        assert(c.ok);
        return *this * c;
    }
    Matrix operator^(i64 x) const {
        Matrix res(n, n), a = *this;
        res.init();
        while (x) {
            if (x & 1) {
                res = res * a;
            }
            a = a * a;
            x >>= 1;
        }
        return res;
    }
    Matrix operator+=(const Matrix &b) { return *this + b; }
    Matrix operator-=(const Matrix &b) { return *this - b; }
    Matrix operator*=(const Matrix &b) { return *this * b; }
    Matrix operator/=(const Matrix &b) { return *this / b; }

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

    Matrix getinv() // 方阵求逆
    {
        if (n != m) {
            Matrix res;
            res.ok = 0;
            return res;
        }
        Matrix c = *this;
        c.m = 2 * n;
        for (int i = 1; i <= n; i++) { // 算增广矩阵的时候要开两倍空间，不然会出问题
            c.a[i].resize(2 * n + 1, 0);
            c.a[i][i + n] = 1;
        }
        for (int i = 1; i <= n; i++) {
            int pos = i;
            for (int j = i + 1; j <= n; j++) {
                if (abs(c.a[j][i]) > abs(c.a[pos][i])) {
                    pos = j;
                }
            }
            if (i != pos) {
                swap(c.a[i], c.a[pos]);
            }
            if (!c.a[i][i]) {
                // puts("No Solution");
                Matrix res;
                res.ok = 0;
                return res;
            }
            i64 inv = qpow(c.a[i][i], mod - 2);
            for (int j = 1; j <= n; j++) {
                if (j != i) {
                    i64 mul = c.a[j][i] * inv % mod;
                    for (int k = i; k <= 2 * n; k++) {
                        c.a[j][k] = ((c.a[j][k] - c.a[i][k] * mul) % mod + mod) % mod;
                    }
                }
            }
            for (int j = 1; j <= 2 * n; j++) {
                c.a[i][j] = c.a[i][j] * inv % mod;
            }
        }
        // 增广矩阵
        // Matrix res(n, 2 * n);
        // res.ok = 1;
        // for (int i = 1; i <= n; i++)
        //     for (int j = 1; j <= 2 * n; j++)
        //         res.a[i][j] = c.a[i][n + j];
        // return res;
        Matrix res(n, n);
        res.ok = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                res.a[i][j] = c.a[i][n + j];
            }
        }
        return res;
    }

    friend constexpr auto &operator>>(istream &in, Matrix &a) {
        for (int i = 1; i <= a.n; i++) {
            for (int j = 1; j <= a.m; j++) {
                in >> a.a[i][j];
            }
        }
        return in;
    }
    friend constexpr auto &operator<<(ostream &out, const
       Matrix &a) {
        for (int i = 1; i <= a.n; i++) {
            for (int j = 1; j <= a.m; j++) {
                out << a.a[i][j] << ' ';
            }
            cout << endl;
        }
        return out;
    }
};
```

字符串专题模板

5.1 StringHash

```
template<const long long N>
struct StringHash {
    using i64 = long long;
    using PII = std::pair<i64, i64>;
    const i64 mod1 = 1e9 + 97, mod2 = 998244853, p1 = 131, p2 =
        233;
    std::array<i64, N> a1, a2;
    std::array<i64, N> Phs1, Phs2;
    std::array<i64, N> Shs1, Shs2;
    StringHash() {
        init(N - 1);
    }
    StringHash(const std::string& S) {
        init(N - 1);
        work(S);
    }
    void work(const std::string& s) {
        i64 n = s.size();
        assert(n + 1 <= N);
        for (int i = 0; i < n; ++i) {
            i64 t = n - i - 1;
            Phs1[i + 1] = ((i64)Phs1[i] * p1 + s[i]) % mod1;
            Phs2[i + 1] = ((i64)Phs2[i] * p2 + s[i]) % mod2;
            Shs1[t + 1] = ((i64)Shs1[t + 2] * p1 + s[t]) % mod1
                ;
            Shs2[t + 1] = ((i64)Shs2[t + 2] * p2 + s[t]) % mod2
                ;
        }
    }
    PII PreHash(i64 l, i64 r) {
        assert(l <= r);
        i64 P1 = (Phs1[r] - (i64)Phs1[l - 1] * a1[r - l + 1] %
            mod1 + mod1) % mod1;
        i64 P2 = (Phs2[r] - (i64)Phs2[l - 1] * a2[r - l + 1] %
            mod2 + mod2) % mod2;
        return PII(P1, P2);
    };
    PII SufHash(i64 l, i64 r) {
        assert(l <= r);
        i64 S1 = (Shs1[l] - (i64)Shs1[r + 1] * a1[r - l + 1] %
            mod1 + mod1) % mod1;
        i64 S2 = (Shs2[l] - (i64)Shs2[r + 1] * a2[r - l + 1] %
            mod2 + mod2) % mod2;
        return PII(S1, S2);
    }
    bool isPlalindrome(i64 l, i64 r) {
        auto [P1, P2] = PreHash(l, r);
        auto [S1, S2] = SufHash(l, r);
        return P1 == S1 && P2 == S2;
    }
    void init(i64 n) {
        a1[0] = a2[0] = 1;
        for (int i = 0; i < n; ++i) {
            a1[i + 1] = (i64)a1[i] * p1 % mod1;
            a2[i + 1] = (i64)a2[i] * p2 % mod2;
        }
    }
};
static const int N = 1e5 + 5;
StringHash<N> h;
```

5.2 AC 自动机

```
template<u64 MAXN, u64 MAXS>
struct AhoCorasick {
    static constexpr int ALPHABEL = 26;

    int cnt;
    std::vector<int> end;
    std::vector<std::array<int, ALPHABEL>> tree;
    std::vector<int> fail;
    std::vector<bool> alert;
    std::vector<int> times;


    AhoCorasick() {
        end.assign(MAXN + 5, -1);
        tree.assign(MAXS + 5, {});
        fail.assign(MAXS + 5, 0);
        alert.assign(MAXS + 5, false);
        times.assign(MAXS + 5, 0);
        cnt = 0;
    }

    void add(int i, const std::string &s) {
        int u = 0;
        for(int j = 0, c; j < s.size(); j++) {
            c = s[j] - 'a';
            if(tree[u][c] == 0) {
                tree[u][c] = ++cnt;
            }
            u = tree[u][c];
        }
        end[i] = u;
        alert[u] = true;
    }

    void setfail() {
        std::queue<int> q;
        for(int i = 0; i < ALPHABEL; i++) {
            if(tree[0][i] > 0) {
                q.push(tree[0][i]);
            }
        }
        while(q.size()) {
            int u = q.front();
            q.pop();

            for(int i = 0; i < ALPHABEL; i++) {
                if(tree[u][i] == 0) {
                    tree[u][i] = tree[fail[u]][i];
                } else {
                    fail[tree[u][i]] = tree[fail[u]][i];
                    q.push(tree[u][i]);
                }
            }

            if(alert[fail[u]]) {
                alert[u] = true;
            }
        }
    }

    template<bool Counter = true>
    void work() {
        setfail();
        if constexpr (Counter) {
            std::string s;
            std::cin >> s;

            for(int i = 0, u = 0; i < s.size(); i++) {
                u = tree[u][s[i] - 'a'];
                times[u]++;
            }

            std::vector<std::vector<int>> g(MAXS + 5);

            auto add_Edge = [&](int u, int v) -> void {
                g[u].push_back(v);
            };

            auto dfs = [&] (auto &&self, int u) -> void {
                for(auto v : g[u]) {
                    self(self, v);
                    times[u] += times[v];
                }
            };

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

                add_Edge(fail[i], i);
            }

            dfs(dfs, 0);
        }
    }

    int get_index_i_times(int i) {
        return times[end[i]];
    }

};
```

5.3 马拉车

```
// the real length in s from p is p[i] - 1
std::vector<int> manacher(std::string s) {
    std::string t = "#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    int n = t.size();
    std::vector<int> p(n);
    for(int i = 0, r = 0, c = 0, len; i < n; i++) {
        len = r > i ? std::min(p[2 * c - i], r - i) : 1;
        while(i + len < n && i - len >= 0 && t[i + len] == t[i
           - len]) {
            len++;
        }
        if(i + len > r) {
            r = i + len;
            c = i;
        }
        p[i] = len;
    }
    return p;
}
```

5.4 Z 函数

```
std::vector<int> zFunction(std::string s) {
    int n = s.size();
    std::vector<int> z(n + 1);
    z[0] = n;
    for(int i = 1, c = 1, r = 1, len; i < n; i++) {
        len = r > i ? std::min(r - i, z[i - c]) : 0;
        while(i + len < n and s[i + len] == len) {
            len++;
        }
        if(i + len > r) {
            r = i + len;
            c = i;
        }
        z[i] = len;
    }
    return z;
}
```

5.5 后缀数组

```
struct SuffixArray {
    int n;
    std::vector<int> sa, rk, lc;
    SuffixArray(const std::string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(), [&](int a, int b) {
           return s[a] < s[b];});
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i -
               1]]);
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; --i)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            std::swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; ++i)
                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] <
                   tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i
                    - 1] + k] < tmp[sa[i] + k]);
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j
                    < n && s[i + j] == s[sa[rk[i] - 1] + j]; )
                    ++j;
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

5.5.1 使用方法示例

```
// deepseek
```

```cpp
#include <iostream>
#include <string>
#include <vector>

struct SAM {
    // ... (用户提供的模板代码)
};

int main() {
    SAM sam;
    int last = 1;  // 初始状态
    std::string s = "abba";
    for (char c : s) {
        last = sam.extend(last, c, 'a');
    }

    // 统计不同子串数量
    int count = 0;
    for (int i = 2; i < sam.size(); ++i) {
        count += sam.len(i) - sam.len(sam.link(i));
    }
    std::cout << "不同子串数量: " << count << std::endl;  // 输
        出 10

    // 检查子串是否存在
    auto is_substring = [&](const std::string& t) {
        int p = 1;
        for (char c : t) {
            p = sam.next(p, c, 'a');
            if (p == 0) return false;
        }
        return true;
    };
    std::cout << "子串 'ab' 是否存在: " << is_substring("ab")
        << std::endl;  // 输出 1

    // 查找最长重复子串
    int max_len = 0;
    for (int i = 2; i < sam.size(); ++i) {
        max_len = std::max(max_len, sam.len(i));
    }
    std::cout << "最长重复子串长度: " << max_len << std::endl;
        // 输出 4

    return 0;
}
```

5.6 回文自动机

```
struct PAM {
    static constexpr int ALPHABET_SIZE = 28;
    struct Node {
        int len;
        int link;
        int cnt;
        std::array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, cnt{}, next{} {}
    };
    std::vector<Node> t;
    int suff;
    std::string s;
    PAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].len = -1;
        suff = 1;
        s.clear();
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    bool add(char c, char offset = 'a') {
        int pos = s.size();
        s += c;
        int let = c - offset;
        int cur = suff, curlen = 0;

        while (true) {
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
                s[pos])
                break;
            cur = t[cur].link;
        }
        if (t[cur].next[let]) {
            suff = t[cur].next[let];
            return false;
        }
        int num = newNode();
        suff = num;
        t[num].len = t[cur].len + 2;
        t[cur].next[let] = num;

        if (t[num].len == 1) {
            t[num].link = 1;
            t[num].cnt = 1;
            return true;
        }

        while (true) {
            cur = t[cur].link;
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
                s[pos]) {
                t[num].link = t[cur].next[let];
                break;
            }
        }
        t[num].cnt = 1 + t[t[num].link].cnt;
        return true;
    }
};
PAM pam;
```

5.7 KMP

```
template <typename T>
struct KMP {
    using value_type = typename T::value_type;
    using size_type = uint32_t;
    T m_seq;
    std::vector<size_type> m_pi;
    void init() {
        m_seq.push_back({});
        m_pi.push_back(0);
    }
    KMP() {
        clear();
    }
    template <typename InitMapping>
    KMP(size_type length, InitMapping mapping) {
        resize(length, mapping);
    }
    template <typename Iterator>
    KMP(Iterator first, Iterator last) {
        reset(first, last);
    }
    KMP(const T &seq) : KMP(seq.begin(), seq.end()) {}
    template <typename InitMapping>
    void resize(size_type length, InitMapping mapping) {
        reserve(length);
        for (size_type i = 0; i != length; i++) {
            push_back(mapping(i));
        }
    }
    template <typename Iterator>
    void reset(Iterator first, Iterator last) {
        resize(last - first, [&](size_type i) { return *(first
             + i); });
    }
    void reserve(size_type length) {
        clear();
        m_seq.reserve(length);
        m_pi.reserve(length);
    }
    void clear() {
        m_seq.clear();
        m_pi.clear();
        init();
    }
    void push_back(const value_type &elem) {
        m_seq.push_back(elem);
        if (size() > 1) {
            size_type pi = jump(m_pi.back(), elem);
            m_pi.push_back(pi + (m_seq[pi + 1] == elem));
        } else
            m_pi.push_back(0);
    }
    void pop_back() {
        m_seq.pop_back();
        m_pi.pop_back();
    }
    size_type size() const { return m_seq.size() - 1; }
    size_type jump(size_type last_pi, const value_type &elem)
       const {
        size_type len = last_pi;
        while (len && (len == size() || m_seq[len + 1] != elem)
           ) {
            len = m_pi[len];
        }
        return len;
    }
    // Check if it is included in a certain sequence
    // if not exist, return -1
    template <typename Iterator>
    size_type contained_by(Iterator first, Iterator last) const
        {
        if (!size()) return 0;
        size_type len = 0;
        for (auto it = first; it != last; ++it) {
            const value_type &elem = *it;
            while (len && m_seq[len + 1] != elem) {
                len = m_pi[len];
            }
            if (m_seq[len + 1] == elem) {
                len++;
            }
            if (len == size()) {
                return (it - first) - len + 1;
            }
        }
        return -1;
    }
    // Call callback for all borders at a certain location
    /*
    kmp.do_for_each_border(j, [&](int pi) {
                if (pi != j) cout << "/";
                cout << p.substr(0, pi);
            });
    */
    template <typename Callback>
    void do_for_each_border(size_type init_border, Callback &&
       call) {
        size_type pi = init_border;
        while (pi) {
            call(pi);
            pi = query_Pi(pi - 1);
        }
    }
    // If there is a prefix string and a suffix string that are
        the same
    // then return greater than 0 O(1)
    size_type query_Pi(size_type i) const {
        return m_pi[i + 1];
    }
};
using KMP_string = KMP<std::string>;
template <typename ValueType>
using KMP_vector = KMP<std::vector<ValueType>>;
```

算法杂项专题模板

6.1 FastIO

```cpp
namespace io_lib {
#ifdef FREAD
#define MAXBUFFERSIZE 1000000
inline char fgetc() {
  static char buf[MAXBUFFERSIZE + 5], *p1 = buf, *p2 = buf;
  return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1,
     MAXBUFFERSIZE, stdin), p1 == p2) ? EOF : *p1++;
}
#undef MAXBUFFERSIZE
#define getchar fgetc
#endif
#define gc getchar
struct IOReader {
  template <typename T, typename std::enable_if<std::
     is_integral<T>::value, int>::type = 0>
  inline const IOReader& operator>>(T& a) const {
    a = 0;
    bool flg = false;
    char ch = gc();
    while (ch < '0' || ch > '9') {
      if (ch == '-') flg ^= 1;
      ch = gc();
    }
    while (ch >= '0' && ch <= '9') {
      a = (a << 3) + (a << 1) + (ch ^ '0');
      ch = gc();
    }
    if (flg) a = -a;
    return *this;
  }
  inline const IOReader& operator>>(std::string& a) const {
    a.clear();
    char ch = gc();
    while (isspace(ch) && ch != EOF) ch = gc();
    while (!isspace(ch) && ch != EOF) a += ch, ch = gc();
    return *this;
  }
  inline const IOReader& operator>>(char* a) const {
    char ch = gc();
    while (isspace(ch) && ch != EOF) ch = gc();
    while (!isspace(ch) && ch != EOF) *(a++) = ch, ch = gc();
    *a = '\0';
    return *this;
  }
  inline const IOReader& operator>>(char& a) const {
    a = gc();
    while (isspace(a)) a = gc();
    return *this;
  }
  template <typename T, typename std::enable_if<std::
     is_floating_point<T>::value, int>::type = 0>
  inline const IOReader& operator>>(T& a) const {
    a = 0;
    bool flg = false;
    char ch = gc();
    while ((ch < '0' || ch > '9') && ch != '.') {
      if (ch == '-') flg ^= 1;
      ch = gc();
    }
    while (ch >= '0' && ch <= '9') {
      a = a * 10 + (ch ^ '0');
      ch = gc();
    }
    if (ch == '.') {
      ch = gc();
      T p = 0.1;
      while (ch >= '0' && ch <= '9') {
        a += p * (ch ^ '0');
        ch = gc();
        p *= 0.1;
      }
    }
    if (flg) a = -a;
    return *this;
  }
  template <typename T1, typename T2>
  inline const IOReader& operator>>(std::pair<T1, T2>& p) const
    {
    return operator>>(p.first), operator>>(p.second), *this;
  }
  template <typename T, const unsigned long long N>
  inline const IOReader& operator>>(std::array<T, N>& p) const
    {
    for (unsigned long long i = 0; i < N; i ++)
      operator>>(p[i]);
    return *this;
  }
  template <typename... Ts>
  inline const IOReader& operator>>(std::tuple<Ts...>& p) const
    ;
#undef importRealReader
};
const IOReader io;
#undef gc
template <typename T>
void read(T& val) { io >> val; }
template <typename T>
void read(int l, int r, T& A) { for (int i = l; i <= r; i++) io
    >> A[i]; }
template <typename T>
void write(const T& A, int l, int r, const char* sp, const char
    * end = "") { for (int i = l; i <= r; i++) printf(sp, A[i]);
    printf("%s", end); }
template <typename T>
void write(const auto& A, const T* sp, const char* end = "") {
    for (auto e : A) printf(sp, e); printf("%s", end); }
template <typename T = int>
T read() { T res; io >> res; return res; }
template <typename T, int N>
std::array<T, N> read() { return read<std::array<T, N>>(); }
template <typename Tuple, typename Func, size_t... N>
```

```cpp
void func_call_tuple(Tuple& t, Func&& func, std::index_sequence
    <N...>) { static_cast<void>(std::initializer_list<int>{(func
    (std::get<N>(t)), 0)...}); }
template <typename... Args, typename Func>
void travel_tuple(std::tuple<Args...>& t, Func&& func) {
    func_call_tuple(t, std::forward<Func>(func), std::
    make_index_sequence<sizeof...(Args)>{}); }
template <typename... Ts>
std::tuple<Ts...> reads() {
  std::tuple<Ts...> res;
  travel_tuple(res, [&](auto&& val) { io >> val; });
  return res;
}
template <typename... Ts>
inline const IOReader& IOReader::operator>>(std::tuple<Ts...>&
    p) const { return p = reads<Ts...>(), *this; }
template <typename T = int>
std::vector<T> getv(int n, int start = 0) {
  std::vector<T> res(start + n);
  for (int i = start; i < start + n; i++) io >> res[i];
  return res;
}
template <typename T, typename T1, typename... Ts>
std::vector<std::tuple<T, T1, Ts...>> getv(int n, int start =
    0) {
  std::vector<std::tuple<T, T1, Ts...>> res(start + n);
  for (int i = start; i < start + n; i++) io >> res[i];
  return res;
}}  // namespace io_lib
using namespace io_lib;

#define cin io
```

6.2 defs

```cpp
namespace defs {
#define YES cout << "YES" << endl;
#define NO cout << "NO" << endl;
#define Yes cout << "Yes" << endl;
```

```cpp
#define No cout << "No" << endl;
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define rep(i, j, k) for(int i = (j); i <= k; ++i)
#define per(i, j, k) for(int i = (j); i >= k; --i)
#define multiCase()     \
    int totCases; std::cin >> totCases; \
    for(int currCase = 1; currCase <= totCases; currCase++)
using i32 = int;
using u32 = unsigned int;
using i64 = long long;
using u64 = unsigned long long;
using i128 = __int128;
using u128 = __uint128_t;
using f32 = float;
using f64 = double;
using TII = std::tuple<int, int, int>;
const i64 mod = 1'000'000'007 /* 998'244'353 */;
template <typename T> void sort(T& v) { std::sort(all(v)); }
template <typename T> T sorted(T v) { return std::sort(v), v; }
template <typename T> void rsort(T& v) { std::sort(rall(v)); }
template <typename T, typename T2> void sort(T& v, T2 compare)
    { std::sort(all(v), compare); }
template <typename T, typename T2> T sorted(T v, T2 compare) {
    return std::sort(v, compare), v; }
template <typename T> void reverse(T& v) { std::reverse(all(v))
    ; }
template <typename T> T reversed(T v) { return std::reverse(v),
     v; }
template <typename T> void unique(vector<T>& v) { v.erase(std::
    unique(all(v)), v.end()); }
template <typename T> vector<T> uniqued(vector<T> v) { return
    std::unique(v), v; }
template <typename T> T min(const vector<T> &v) { return *std::
    min_element(all(v)); }
template <typename T> T max(const vector<T> &v) { return *std::
    max_element(all(v)); }
template <typename T> T acc(const vector<T> &v) { return std::
    accumulate(v.begin(), v.end(), T(0LL)); }
```

```
template <typename T> istream& operator>>(istream& is, std::
    vector<T>& v) { for(auto& x : v) { is >> x; } return is; }

}
using namespace defs;
```

6.3 Int128

```
#if defined(__GNUC__) || defined(__clang__)
using i128 = __int128;
using u128 = unsigned __int128;
#else
#error "int128 is only supported on GCC and Clang compilers"
#endif

namespace std {
template <>
class numeric_limits<i128> {
public:
    static constexpr bool is_specialized = true;
    static constexpr i128 min() { return static_cast<u128>(1)
        << 127; }
    static constexpr i128 max() { return ~(static_cast<u128>(1)
         << 127); };
};
} // namespace std

std::ostream& operator<<(std::ostream& os, i128 n) {
    if (n == 0) return os << '0';

    const bool is_negative = n < 0;
    u128 abs_n = is_negative ? -static_cast<u128>(n) :
        static_cast<u128>(n);

    char buffer[40] = {0};
    char* ptr = buffer + sizeof(buffer) - 1;

    while (abs_n > 0) {
        *--ptr = '0' + abs_n % 10;
```

```cpp
        abs_n /= 10;
    }

    if (is_negative) *--ptr = '-';
    return os << ptr;
}


std::istream& operator>>(std::istream& is, i128& n) {
    std::string s;
    is >> s;

    try {
        n = toi128(s);
    } catch (const std::exception& e) {
        is.setstate(std::ios::failbit);
        throw;
    }
    return is;
}

i128 toi128(const std::string& s) {
    if (s.empty()) throw std::invalid_argument("Empty input
       string");

    size_t pos = 0;
    const bool negative = (s[0] == '-');
    if (negative || s[0] == '+') pos++;
    if (pos >= s.size()) throw std::invalid_argument("Invalid
       number format");

    constexpr i128 max_prev = std::numeric_limits<i128>::max()
       / 10;
    constexpr i128 max_digit = std::numeric_limits<i128>::max()
        % 10;

    i128 result = 0;
    for (; pos < s.size(); ++pos) {
        if (!std::isdigit(s[pos])) {
            throw std::invalid_argument("Non-digit character in input");
        }

        const int digit = s[pos] - '0';
        if (result > max_prev || (result == max_prev && digit > max_digit + negative)) {
            throw std::overflow_error("i128 overflow");
        }

        result = result * 10 + digit;
    }
    return negative ? -result : result;
}

i128 sqrti128(i128 n) {
    if (n < 0) throw std::domain_error("Square root of negative number");
    if (n == 0) return 0;

    i128 low = 0;
    i128 high = (static_cast<u128>(1) << 63) - 1;

    while (low < high) {
        const i128 mid = (low + high + 1) / 2;

        bool overflow = false;
        i128 square;
        if (mid > std::numeric_limits<i128>::max() / mid) {
            overflow = true;
        } else {
            square = mid * mid;
        }

        if (overflow || square > n) {
            high = mid - 1;
        } else {
            low = mid;
        }
```

```cpp
    }
    return low;
}

i128 gcd(i128 a, i128 b) noexcept {
    if (a == 0) return b;
    if (b == 0) return a;

    int shift = 0;
    while (((a | b) & 1) == 0) {
        a >>= 1;
        b >>= 1;
        ++shift;
    }

    while (a != b) {
        while ((a & 1) == 0) a >>= 1;
        while ((b & 1) == 0) b >>= 1;
        if (a > b) std::swap(a, b);
        b -= a;
    }
    return a << shift;
}
```

6.4 二分搜索

```cpp
// return the first ans in [lo, hi], such as check(md) = true
// if no such ans, return hi + 1
template<class T, class Func>
T binary_min_left(T lo, T hi, Func check) {
      T ans = hi + 1;
      while(lo <= hi) {
            T md = lo + (hi - lo) >> 1;
        if(check(md)) {
            ans = md;
            hi = md - 1;
        } else {
            lo = md + 1;
        }
```

```
    }
    return ans;
}


// return the last ans in [lo, hi], such as check(md) = true
// if no such ans, return lo - 1
template<class T, class Func>
T binary_max_right(T lo, T hi, Func check) {
    T ans = lo - 1;
    while(lo <= hi) {
        T md = lo + (hi - lo) >> 1;
        if(check(md)) {
            ans = md;
            lo = md + 1;
        } else {
            hi = md - 1;
        }
    }
      return ans;
}
```

6.5 自定义哈希

```
struct custom_hash_64 {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            std::chrono::steady_clock::now().time_since_epoch()
               .count();
        return splitmix64(x ^ FIXED_RANDOM);
    }
};
```

```cpp

struct custom_hash_32 {
    uint64_t operator()(uint32_t x) const {
        static const uint32_t RANDOM =
            std::chrono::steady_clock::now().time_since_epoch()
               .count();
        return (x ^ RANDOM) * 0x9e3779b1;
    }
};

// unordered_map<int, int, custom_hash_xxx> cnt;
```

平面几何专题模板

7.1 平面几何

```cpp
template<class T>
int sgn(const T& v) {
    static constexpr T eps = 1e-8;
    return v > eps ? 1 : v < -eps ? -1 : 0;
}
template<class T>
struct Point {// Point or Vector
    T x, y;
    Point() : x(0), y(0) {}
    Point(T x, T y) : x(x), y(y) {}
    template<class U>
    explicit operator Point<U>() const {
        return Point<U>(U(x), U(y));
    }
    Point operator+(const Point& o) const {
        return Point(x + o.x, y + o.y);
    }
    Point operator-(const Point& o) const {
        return Point(x - o.x, y - o.y);
    }
    Point operator-() const {
        return Point(-x, -y);
    }
    Point operator*(const T& v) const {
        return Point(x * v, y * v);
    }
    friend Point operator*(const T& v, const Point<T>& o) {
        return Point(o.x * v, o.y * v);
    }
    Point operator/(const T& v) const {
        return Point(x / v, y / v);
    }
    Point operator+=(const Point& o) {
        x += o.x, y += o.y;
        return *this;
    }
    Point operator-=(const Point& o) {
        x -= o.x, y -= o.y;
        return *this;
    }
    Point operator*=(const T& v) {
        x *= v, y *= v;
        return *this;
    }
    Point operator/=(const T& v) {
        x /= v, y /= v;
        return *this;
    }
    bool operator==(const Point& o) const {
        return sgn(x - o.x) == 0 and sgn(y - o.y) == 0;
    }
    bool operator!=(const Point& o) const {
        return sgn(x - o.x) != 0 or sgn(y - o.y) != 0;
    }
    bool operator<(const Point& o) const {
        return sgn(x - o.x) < 0 or sgn(x - o.x) == 0 and sgn(y
           - o.y) < 0;
    }
    bool operator>(const Point& o) const {
        return sgn(x - o.x) > 0 or sgn(x - o.x) == 0 and sgn(y
           - o.y) > 0;
    }
    static bool argcmp(const Point& a, const Point& b) {
        static auto get = [&](const Point& o) {
            if(sgn(o.x) == 0 and sgn(o.y) == 0) return 0;
            if(sgn(o.y) > 0 or sgn(o.y) == 0 and sgn(o.x) < 0) return 1;
            return -1;
        };
        int ta = get(a), tb = get(b);
        if(ta != tb) return ta < tb;
        return a.toLeft(b) == 1;// 不关注极径
        // int tole = a.toLeft(b);
        // if(tole != 0) return tole == 1;
        // return sgn(a.square()-b.square()) < 0;// 极角相同按极径排
    }
    T dot(const Point& o) const {
        return x * o.x + y * o.y;
    }
    T cross(const Point& o) const {
        return x * o.y - y * o.x;
    }
    int toLeft(const Point& o) const {
        return sgn(cross(o));
    }
    T square() const {
        return x * x + y * y;
    }
    T interSquare(const Point& o) const {
        return (*this - o).square();
    }
    friend istream& operator>>(istream& in, Point& o) {
        return in >> o.x >> o.y;
    }
    friend ostream& operator<<(ostream& out, Point const& o) {
        return out << "(" << o.x << "," << o.y << ")";
    }
    // 涉及浮点数
    double length() const {
```

```cpp
        return sqrtl(square());
    }
    double distance(const Point& o) const {
        return (*this - o).length();
    }
    // 逆时针旋转 rad
    template<class U>
    Point<U> rotate(U cosr, U sinr) const {
        return Point(x * cosr - y * sinr, x * sinr + y * cosr);
    }
    // 两向量夹角范围是 [0,PI]
    double ang(const Point& o) const {
        return acosl(max(-1.0l, min(1.0l, dot(o) / (length() *
            o.length()))));
    }
};
template<class T>
struct Line {// Line or Segment
    Point<T> a, b;// 方向为 a->b
    Line() {}
    Line(const Point<T>& a, const Point<T>& b) : a(a), b(b) {}
    template<class U>
    Line(const Point<U>& a, const Point<U>& b) : a(a), b(b) {}
    Point<T> vec() const {
        return b - a;
    }
    // Line
        ----------------------------------------------------------

    bool parallel(const Line& l) const {
        return sgn((b - a).cross(l.b - l.a)) == 0;
    }
    int toLeft(const Point<T>& o) const {
        return (b - a).toLeft(o - a);
    }
    // 涉及浮点数
    // 直线交点
    Point<double> lineIntersection(const Line& l) const {
        return Point<double>(a) + Point<double>(b - a) *
```

```
            (1. * (l.b - l.a).cross(a - l.a) / (l.b - l.a).
                cross(a - b));
    }
    // 点到直线的距离
    double distanceLP(const Point<T>& o) const {
        return abs((a - b).cross(a - o)) / (a - b).length();
    }
    // 点在直线上的投影
    Point<T> projection(const Point<T>& o) const {
        return a + (b - a) * (1. * (b - a).dot(o - a) / (b - a)
            .square());
    }
    // Segment
        ----------------------------------------------------------

    // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
    int contain(const Point<T>& o) const {
        if(o == a or o == b) return -1;
        return (o - a).toLeft(o - b) == 0 and sgn((o - a).dot(o
            - b)) < 0;
    }
    // 判断线段直线是否相交
    // 0 线段和直线不相交 | 1 线段和直线严格相交 | 2 仅在某一线
        段端点处相交 | 3 直线包含线段
    int interWithLine(const Line& l) const {
        int num = !l.toLeft(a) + !l.toLeft(b);
        if(num) return num + 1;
        return l.toLeft(a) != l.toLeft(b);
    }
    // 判断两线段是否相交
    // 0 两线段不相交 | 1 两线段严格相交 | 2 仅在某一线段端点处
        相交 | 3 两线段有重叠
    int interWithSegment(Line s) const {
        if((a < b) != (s.a < s.b))
            swap(s.a, s.b);
        int num = (contain(s.a) != 0) + (contain(s.b) != 0)
            + (s.contain(a) != 0) + (s.contain(b) != 0);
        if(parallel(s)) {
            if(!num) return 0;
            if(b == s.a or a == s.b) return 2;// -.-
            return 3;
        }
        if(num) return 2;
        return toLeft(s.a) * toLeft(s.b) == -1 and s.toLeft(a)
            * s.toLeft(b) == -1;
    }
    // 点到线段的距离
    double distanceSP(const Point<T>& o) const {
        if(sgn((o - a).dot(b - a)) < 0) return o.distance(a);
        if(sgn((o - b).dot(a - b)) < 0) return o.distance(b);
        return abs((a - b).cross(a - o)) / (a - b).length();
    }
    // 两线段间距离
    double distanceSS(const Line& s) const {
        if(interWithSegment(s)) return 0;
        return min({distanceSP(s.a),distanceSP(s.b),
                s.distanceSP(a),s.distanceSP(b)});
    }
};
template<class T>
struct Polygon {
    int n;
    vector<Point<T>> p;
    // p 以逆时针顺序存储 2 遍
    Polygon(vector<Point<T>> const& p_) : n(p_.size()), p(p_) {
        p.insert(p.end(), p_.begin(), p_.end());
    }
    // 返回 回转数 = 逆时针转头圈数-顺时针转头圈数
    // 1e9 在多边形上 / 0 不在多边形内 / !=0 在多边形内
    int contain(const Point<T>& o) const {
        int cnt = 0;
        for(int i = 0; i < n; ++i) {
            Point<T> const& u = p[i], v = p[i + 1];
            Line<T> const l(u, v);
            if(l.contain(o)) return 1e9;
            cnt += l.toLeft(o) > 0 and sgn(u.y - o.y) < 0 and
               sgn(v.y - o.y) >= 0;
```

```
                cnt -= l.toLeft(o) < 0 and sgn(u.y - o.y) >= 0 and
                    sgn(v.y - o.y) < 0;
            }
            return cnt;
        }
        // 多边形面积的两倍，可用于判断点的存储顺序是顺时针或逆时针
            （逆正顺负）
        T area() const {
            T sum = 0;
            for(int i = 0; i < n; ++i)
                sum += p[i].cross(p[i + 1]);
            return sum;
        }
        // 多边形的周长
        double perimeter() const {
            double sum = 0;
            for(int i = 0; i < n; ++i)
                sum += p[i].distance(p[i + 1]);
            return sum;
        }
};

template<class T>
struct Convex : Polygon<T> {
        using Polygon<T>::n;
        using Polygon<T>::p;
        Convex(vector<Point<T>> const& p, bool keepRaw) : Polygon<T
            >(p) {}
        Convex(vector<Point<T>> const& p_) : Polygon<T>(andrew(p_))
            {}
        // 对点集 p 求凸包
        static auto andrew(vector<Point<T>> p) {
            sort(p.begin(), p.end());
            p.erase(unique(p.begin(), p.end()), p.end());
            if(p.size() <= 1) return p;
            vector<Point<T>> st;
            for(auto& e : p) {
                while(st.size() > 1 and
```

```cpp
                (st.back() - st.end()[-2]).toLeft(e - st.back()) <= 0)
                st.pop_back();
            st.push_back(e);
        }
        int sz = st.size();
        for(int i = (int)p.size() - 2; i >= 0; --i) {
            while(st.size() > sz and
                (st.back() - st.end()[-2]).toLeft(p[i] - st.back()) <= 0)
                st.pop_back();
            st.push_back(p[i]);
        }
        st.pop_back();
        return st;
    }
    // O(logn)判断点是否在凸多边形内
    // -1 在边界上 | 0 在外部 | 1 严格在内部
    int contain(const Point<T>& o) const {
        if(n == 1) return p[0] == o ? -1 : 0;
        int fTo = (p[1] - p[0]).toLeft(o - p[0]);
        int bTo = (p.back() - p[0]).toLeft(o - p[0]);
        if(fTo == -1 or bTo == 1) return 0;
        if(fTo == 0) return sgn((o - p[0]).dot(o - p[1])) <= 0 ? -1 : 0;
        if(bTo == 0) return sgn((o - p[0]).dot(o - p.back())) <= 0 ? -1 : 0;

        int i = partition_point(p.begin() + 2, p.begin() + n, [&](Point<T> const& v) {
            return (v - p[0]).toLeft(o - p[0]) >= 0;
        }) - p.begin();
        Line<T> const l(p[i - 1], p[i]);
        return l.contain(o) ? -1 : l.toLeft(o) == 1;
    }
    // O(logn) 二分找到 f 方向上的切点 i，满足 p[i]-p[i-1],f(p[i]),p[i+1]-p[i] 逆时旋转
    template<class Func>
    int extreme(Func const& f) {
        assert(n > 2);
        Point<ll> const divVec = f(p[0]);
        bool const flag = (p[0] - p[n - 1]).toLeft(divVec) < 0;
        return partition_point(p.begin(), p.begin() + n, [&](
            Point<T> const& a) {
            if(divVec.toLeft(a - p[0]) > 0) return flag;
            return (*(&a + 1) - a).toLeft(f(a)) > 0;
        }) - p.begin();
    }
    // O(logn) 二分找到 v 方向 和 -v 方向上的切点, 返回值切点下
        标 in [0,n-1]
    array<int, 2> tangentByLine(Point<T> const& v) {
        int i = extreme([&](...) {return v; });
        int j = extreme([&](...) {return -v; });
        return {i,j};
    }
    // O(logn) 过点 o 向凸包做两条切线（先左后右）, 返回值切点
        下标 in [0,n-1]
    // 需要保证 o 在凸包外面
    array<int, 2> tangentByPoint(Point<T> const& o) {
        int i = extreme([&](Point<T> const& a) {return o - a;
            });
        int j = extreme([&](Point<T> const& a) {return a - o;
            });
        return {i, j};
    }
};
```

7.2 线段在多边形内

```cpp
// struct Polygon {
// O(|p|) 判断线段在多边形内
// 可以用整型判断 / 如果用浮点型, 可能要把精度调松一点
bool contain(const Line<T>& s) const {
    if(!contain(s.a) or !contain(s.b))
        return false;
    if(s.a == s.b)
        return true;
    vector<int> t(p.size());
    for(int i=0; i<p.size(); ++i) {
        auto& u = p[i];
        auto& v = p[nxt(i)];
        Line<T> uv(u,v);
        t[i] = s.interWithSegment(uv);
        if(t[i] == 0) continue;// not intersect
        if(t[i] == 1) return false;// strickly intersect
        if(t[i] == 2 and uv.contain(s.a)==1 and uv.toLeft(s.b)
           ==-1)
            return false;
        if(t[i] == 3) {// overlap
            if(s.contain(v)==1 and uv.toLeft(p[nxt(nxt(i))])
               ==1)
                return false;
            if(s.contain(u)==1 and uv.toLeft(p[pre(i)])==1)
                return false;
        }
    }
    for(int i=0; i<p.size(); ++i) {
        if(!(t[i]==2 and t[nxt(i)]==2)) continue;
        auto& v = p[nxt(i)];
        // intersect at v
        if(s.contain(v) and s.b != v) {
            auto& u = p[i];
            auto& w = p[nxt(nxt(i))];
            if((v-u).toLeft(w-u)==1) {
                if(s.toLeft(u)==1 and s.toLeft(w)==-1);
                else return false;
            }else {
                if(s.toLeft(u)==-1 and s.toLeft(w)==1)
                    return false;
            }
        }
    }
    return true;
}
```

7.3 钝角直角三角形计数问题

```cpp
for(int k=0; k<n; ++k)  {
    std::vector<Point<ll>> b;
    b.reserve(n-1);
    for(int i=0; i<n; ++i) {
        if(a[i] == a[k]) continue;
        b.emplace_back(a[i]-a[k]);
    }
    sort(ALL(b), Point<ll>::argcmp);
    int sz = b.size();
    if(!sz) continue;
    b.insert(b.end(), b.begin(), b.end());
    for(int i=0,l=0,r=0; i<sz; ++i) {
        // 0
        auto eq0 = [&](int j) {
            return b[i].cross(b[j])==0 and b[i].dot(b[j])>0;
        };
        // [0,89]
        auto le89 = [&](int j) {
            return b[i].cross(b[j])>=0 and b[i].dot(b[j])>0;
        };
        // [0,180)
        auto le179 = [&](int j) {
            int t = b[i].toLeft(b[j]);
            return t>0 or t==0 and b[i].dot(b[j])==0;
        };
        // [l,r) -> [90,180)
        l = max(l, i);
        while(l<i+sz and le89(l)) l++;
        r = max(r, l);
        while(r<i+sz and le179(r) and !eq0(r)) r++;
        ans += r-l;
    }
}
```

7.4 向量夹角

```cpp
double alpha = atan2(v.cross(w), v.dot(w));// v,w 不是零向量
```

7.5 凸包上旋转卡尺算法的其他应用

```
// struct Convex {
// 旋转卡尺求直径的平方
T rotatingCalipers() const {
    if(p.size()==1) return 0;
    if(p.size()==2) return p[0].interSquare(p[1]);
    T ans = 0;
    for(int i=0, j=1; i<p.size(); ++i) {
        Point<T> v = p[nxt(i)]-p[i];
        while(v.cross(p[nxt(j)]-p[i]) > v.cross(p[j]-p[i]))
            j=nxt(j);
        ans = std::max({ans, p[i].interSquare(p[j]),
                  p[nxt(i)].interSquare(p[j])});
        if(v.cross(p[nxt(j)]-p[i]) == v.cross(p[j]-p[i]))
            ans = std::max({ans, p[i].interSquare(p[nxt(j)]),
                      p[nxt(i)].interSquare(p[nxt(j)])});
    }
    return ans;
}
// 结论：覆盖凸包的最小面积/周长矩形，一定有一条边和凸包某条边
   重叠
// 旋转卡尺求最小面积矩形
double minErea() const {
    if(p.size()<=2) return 0;
    double ans = std::numeric_limits<double>::max();
    for(int i=0, j=1, k=1, l=1; i<p.size(); ++i) {
        Point<T> v = p[nxt(i)]-p[i];
        while(v.cross(p[nxt(j)]-p[i]) > v.cross(p[j]-p[i])) j =
            nxt(j);
        while(v.dot(p[nxt(k)]-p[i]) > v.dot(p[k]-p[i])) k = nxt
           (k);
        if(!i) l = j;
        while(v.dot(p[nxt(l)]-p[i]) < v.dot(p[l]-p[i])) l = nxt
           (l);
        ans = std::min(ans, 1. * v.cross(p[j]-p[i]) / v.square
           ()
                        * (v.dot(p[k]-p[i]) - v.dot(p[l]-p[i]))
                          );
    }
    return ans;
}
// 旋转卡尺求最小宽度
double minWidth() const {
    if(p.size()<=2) return 0;
    double ans = std::numeric_limits<double>::max();
    for(int i=0, j=1; i<p.size(); ++i) {
        Point<T> v = p[nxt(i)]-p[i];
        while(v.cross(p[nxt(j)]-p[i]) > v.cross(p[j]-p[i])) j =
            nxt(j);
        ans = std::min(ans, v.cross(p[j]-p[i])/v.length());
    }
    return ans;
}
// 计算两个相离的凸包之间的最短距离，注意调用两次取min
double distance(const Convex<double>& B) {
    double ans = std::numeric_limits<double>::max();
    for(int i=0,j=0; i<p.size(); ++i) {
        Point<double> v(p[nxt(i)]-p[i]);
        Line<double> s(p[i], p[nxt(i)]);
        if(i == 0) {
            double mx = std::numeric_limits<double>::min();
            for(int k=0; k<B.p.size(); ++k) {
                double cro = v.cross(B.p[k]-p[i]);
                if(sgn(cro-mx)>0) {
                    mx = cro;
                    j = k;
                }
            }
            ans = std::min(ans, s.distanceSP(!j ? B.p.back(): B
               .p[j-1]));
        }
        ans = std::min(ans, s.distanceSP(B.p[j]));
        while(sgn(v.cross(B.p[B.nxt(j)]-p[i])-v.cross(B.p[j]-p[
           i])) >= 0) {
            j = B.nxt(j);
            ans = std::min(ans, s.distanceSP(B.p[j]));
        }
    }
    return ans;
}
```

7.6 最大最小三角形

```
// 求出的是最大/最小三角形面积的两倍
template<class T>
std::pair<T,T> minMaxTriangle(const std::vector<Point<T>>& a) {
    int n = a.size();
    T mn = numeric_limits<T>::max();
    T mx = 0;
    using Node = std::tuple<int,int,Point<T>>;
    std::vector<Node> all;
    all.reserve(n*n);
    std::vector<int> id(n), pos(n);
    for(int i=0; i<n; ++i) {
        id[i] = i;
        for(int j=0; j<n; ++j) {
            if(i==j) continue;
            if(a[i]==a[j]) mn=0;
            else all.emplace_back(i,j,a[j]-a[i]);
        }
    }
    std::sort(all.begin(), all.end(), [&](const Node& x,const Node& y) {
        return Point<T>::argcmp(get<2>(x), get<2>(y));
    });
    std::sort(id.begin(), id.end(), [&](const int& i,const int& j) {
        return a[i].y < a[j].y or a[i].y==a[j].y and a[i].x>a[j].x;
    });
    for(int i=0; i<n; ++i)
        pos[id[i]] = i;
    for(auto [i,j,v]:all) {
        // 如果没有三点共线 assert(pos[i] = pos[j] + 1);
        if(pos[i] > pos[j]) {
            std::swap(id[pos[i]], id[pos[j]]);
```

```
                std::swap(pos[i], pos[j]);
            }
            int t = std::max(pos[i],pos[j])+1;
            if(t<n) {
                mn = std::min(mn, (a[id[t]]-a[i]).cross(v));
                mx = std::max(mx, (a[id.back()]-a[i]).cross(v));
            }
        }
        return {mn, mx};
}
```

7.7 动态凸包

```
template<class T>
struct DynamicConvex {
    /// @note operator< 使用极角序，并考虑极径
    Point<T> o;
    std::set<Point<T>> s; // 坐标扩大三倍，使得三角形中心为整数
    using Iter = decltype(s.begin());
    auto nxt(Iter it) const {
        return next(it) == s.end() ? s.begin() : next(it);
    }
    auto pre(Iter it) const {
        return it == s.begin() ? --s.end() : prev(it);
    }
    bool contain(Point<T> const& a) const {
        if(s.size() == 0) return 0;
        if(s.size() == 1) return *s.begin() == a * 3;
        if(s.size() == 2) return Line<T>(*s.begin(), *s.rbegin
           ()).contain(a);
        auto it = s.lower_bound(a * 3 - o);
        if(it == s.end()) it = s.begin();
        return (*it - *pre(it)).toLeft(a * 3 - o - *pre(it)) >=
            0;
    }
    void add(Point<T> a) {
        if(s.size() <= 1) {
            s.insert(a * 3);
            return;
        }
        if(s.size() == 2) {
            auto u = *s.begin(), v = *s.rbegin();
            if((u - v).toLeft(a * 3 - v) == 0) return;
            o = (u + v + a * 3) / 3;
            s = {u - o, v - o, a * 3 - o};
            for(auto it = s.begin(); it != s.end(); ++it)
                addEdge(it, nxt(it));
            return;
        }
        if(contain(a)) return;
        a = a * 3 - o;
        auto it = s.insert(a).first, np = nxt(it), pp = pre(it)
            ;
        delEdge(pp, np);
        while(s.size() > 3 and ((*np - a).toLeft(*nxt(np) - *np
            )) != 1) {
            delEdge(np, nxt(np));
            s.erase(np);
            np = nxt(it);
        }
        while(s.size() > 3 and ((*pp - *pre(pp)).toLeft(a - *pp
            )) != 1) {
            delEdge(pre(pp), pp);
            s.erase(pp);
            pp = pre(it);
        }
        addEdge(pre(it), it);
        addEdge(it, nxt(it));
    }

    double D;// 周长
    std::map<Point<T>, Iter> edge;
    void addEdge(Iter it, Iter nit) {// s.size()>=3 时维护信息
        D += it->distance(*nit);
        edge[*nit - *it] = it;
    }
    void delEdge(Iter it, Iter nit) {
        D -= it->distance(*nit);
```

```
            edge.erase(*nit - *it);
        }
        /// @note 调用前注意直线 ax+by=c 的坐标扩大三倍 ax+by=3c
        std::array<Point<T>, 2> extremeByLine(Point<T> const& v)
            const {
            assert(s.size() > 2);
            auto get = [&](Point<T> const& v) {
                auto it = edge.lower_bound(v);
                if(it == edge.end()) it = edge.begin();
                return *(it->second) + this->o;
            };
            return {get(v), get(-v)};
        }
};
```

7.8 闵可夫斯基和

```
// A+B = {a+b | a \in A, b \in B}, 复杂度 O(n)
template<class T>
Convex<T> MinkowskiSum(Convex<T> const& A, Convex<T> const& B)
    {
    auto cmp = [&](Point<T> const& a, Point<T> const& b) {
        return a.y > b.y or a.y == b.y and a.x < b.x;
    };
    int a = std::min_element(A.p.begin(), A.p.begin() + A.n,
        cmp) - A.p.begin();
    int b = std::min_element(B.p.begin(), B.p.begin() + B.n,
        cmp) - B.p.begin();
    Point<T> s(A.p[a] + B.p[b]);
    std::vector<Point<T>> ps(1, s);
    auto popC = [&](Point<T> const& e, Point<T> const& f) {
        return (e - f).toLeft(s - e) == 0 and sgn((e - f).dot(s
            - e)) >= 0;
    };
    auto f = [&](int owner, int i) {
        return !owner ? A.p[a + i + 1] - A.p[a + i] : B.p[b + i
            + 1] - B.p[b + i];
    };
    for(int i = 0, j = 0; i < A.n or j < B.n; ) {
        if(j >= B.n or i < A.n and Point<T>::argcmp(f(0, i), f
           (1, j))) s += f(0, i++);
        else s += f(1, j++);
        while(ps.size() > 1 and popC(ps.back(), ps.end()[-2]))
           ps.pop_back();
        ps.emplace_back(s);
    }
    ps.pop_back();
    return Convex<T>(ps, true);
};
```

7.9 半平面交

```
template<class T>
std::vector<Line<T>> hp(std::vector<Line<T>> vs, T inf = T(1e9)
    ) {
    vs.emplace_back(Point<T>(inf, -inf), Point<T>(inf, inf));
    vs.emplace_back(Point<T>(inf, inf), Point<T>(-inf, inf));
    vs.emplace_back(Point<T>(-inf, inf), Point<T>(-inf, -inf));
    vs.emplace_back(Point<T>(-inf, -inf), Point<T>(inf, -inf));
    auto sameDir = [&](Line<T> const& a, Line<T> const& b) {
        return a.parallel(b) and sgn(a.vec().dot(b.vec())) >=
            0;
    };
    std::sort(vs.begin(), vs.end(), [&](Line<T> const& a, Line<
        T> const& b) {
        if(sameDir(a, b)) return a.toLeft(b.a) == -1;
        return Point<T>::argcmp(a.vec(), b.vec());
    });
    auto canPop = [&](Line<T> const& a, Line<T> const& b, Line<
        T> const& c) {
        if constexpr(!is_same_v<T, double>) {
            __int128_t x = (c.b - c.a).cross(b.a - c.a), y = (c
                .b - c.a).cross(b.a - b.b);
            using P = Point<__int128_t>;
            return P(a.vec()).toLeft(P(b.a) * y + P(b.vec()) *
                x - P(a.a) * y) == -sgn(y);
        }
```

```cpp
        return Point<double>(a.vec()).toLeft(b.lineIntersection(c) - Point<double>(a.a)) < 0;
    };
    std::deque<Line<T>> q;
    for(auto& v : vs) {
        if(q.size() and sameDir(q.back(), v)) continue;
        while(q.size() > 1 and canPop(v, q.back(), q[q.size() - 2])) q.pop_back();
        while(q.size() > 1 and canPop(v, q[0], q[1])) q.pop_front();
        if(q.size() and q.back().vec().toLeft(v.vec()) <= 0) return {};
        q.push_back(v);
    }
    while(q.size() > 1 and canPop(q[0], q.back(), q[q.size() - 2])) q.pop_back();
    while(q.size() > 1 and canPop(q.back(), q[0], q[1])) q.pop_front();
    return std::vector<Line<T>>(q.begin(), q.end());
}
```

常见模型

8.1 子数组最大累加和

```cpp
// index-base 0
// nums数组 该数组里面每个数的值
std::vector<int> dp(n);
dp[0] = nums[0];
for(int i = 1; i < n; i++) {
    dp[i] = std::max(nums[i], dp[i - 1] + nums[i]);
}
std::cout << *std::max_element(dp.begin(), dp.end());
```

8.2 最长递增子序列

```cpp
// index-base 0
// nums数组 该序列的每个数
// len 最长长度
auto binarysearch = [](std::vector<int>& ends, int len, int num
    ) -> int {
    int l = 0, r = len - 1, m, ans = -1;
    while(l <= r) {
        m = (l + r) >> 1;
        if(ends[m] >= num) {
            ans = m;
            r = m - 1;
        } else {
            l = m + 1;
        }
    }
    return ans;
};

auto get = [&]() -> int {
    std::vector<int> ends(n);
    int len = 0;
    for(int i = 0, find; i < n; i++) {
        find = binarysearch(ends, len, nums[i]);
        if(find == -1) {
            ends[len++] = nums[i];
        } else {
            ends[find] = nums[i];
        }
    }
    return len;
};
```

8.3 最长不下降子序列

```
// index-base 0
// nums数组 该序列的每个数
// len 最长长度
auto binarysearch = [](std::vector<int>& ends, int len, int num
    ) -> int {
    int l = 0, r = len - 1, m, ans = -1;
    while(l <= r) {
```

```cpp
        m = (l + r) >> 1;
        if(ends[m] > num) {
            ans = m;
            r = m - 1;
        } else {
            l = m + 1;
        }
    }
    return ans;
};

auto get = [&]() -> int {
    std::vector<int> ends(n);
    int len = 0;
    for(int i = 0, find; i < n; i++) {
        find = binarysearch(ends, len, nums[i]);
        if(find == -1) {
            ends[len++] = nums[i];
        } else {
            ends[find] = nums[i];
        }
    }
    return len;
};
```

8.4 01 背包

8.4.1 无空间优化

```cpp
// index-base 1
// n 物品编号
// t 最大容量
// cost数组 每个物品的容量
// val数组 每个物品的价值
std::vector<std::vector<int>> dp(n + 1, std::vector<int>(t + 1)
   );
for(int i = 1; i <= n; i++) {
    for(int j = 0; j <= t; j++) {
        dp[i][j] = dp[i - 1][j];
        if(j - cost[i] >= 0) {
            dp[i][j] = std::max(dp[i][j], dp[i - 1][j - cost[i
               ]] + val[i]);
        }
    }
}
std::cout << dp[n][t];
```

8.4.2 空间优化

```cpp
// index-base 1
std::vector<int> dp(t + 1);
for(int i = 1; i <= n; i++) {
    for(int j = t; j >= cost[i]; j--) {
        dp[j] = std::max(dp[j], dp[j - cost[i]] + val[i]);
    }
}
std::cout << dp[t];
```

8.5 分组背包

8.5.1 无空间优化

```cpp
// index-base 1
// m 物品总重量
// n 物品数量
// arr[i][0] i号物品的体积
// arr[i][1] i号物品的价值
// arr[i][2] i号物品的组号
// teams 物品组数
int m, n;
std::cin >> m >> n;
std::vector<std::array<int, 3>> nums(n + 1);
for(int i = 1; i <= n; i++) {
    std::cin >> nums[i][0] >> nums[i][1] >> nums[i][2];
}
std::sort(nums.begin() + 1, nums.end(), [](std::array<int, 3>
   a1, std::array<int, 3> a2) {
    return a1[2] >= a2[2];
});

int teams = 1;
for(int i = 2; i <= n; i++) {
    if(nums[i - 1][2] != nums[i][2]) {
        teams++;
    }
}
std::vector<std::vector<int>> dp(teams + 1, std::vector<int>(m
   + 1));
for(int start = 1, end = 2, i = 1; start <= n; i++) {
    while(end <= n && nums[end][2] == nums[start][2]) {
        end++;
    }
    for(int j = 0; j <= m; j++) {
        dp[i][j] = dp[i - 1][j];
        for(int k = start; k < end; k++) {
            if(j - nums[k][0] >= 0) {
                dp[i][j] = std::max(dp[i][j], dp[i - 1][j -
                   nums[k][0]] + nums[k][1]);
            }
        }
    }
    start = end++;
}
std::cout << dp[teams][m];
```

8.5.2 空间优化

```cpp
// index-base 1
std::vector<int> dp(m + 1);
for(int start = 1, end = 2; start <= n;) {
    while(end <= n and nums[end][2] == nums[start][2]) {
        end++;
    }
    for(int j = m; j >= 0; j--) {
        for(int k = start; k < end; k++) {
            if(j - nums[k][0] >= 0) {
                dp[j] = std::max(dp[j], nums[k][1] + dp[j -
                   nums[k][0]]);
            }
        }
    }
    start = end++;
}
std::cout << dp[m];
```

8.6 完全背包

8.6.1 无空间优化

```
// index-base 1
// t 背包总容量
// m 物品个数
// cost数组 每个物品的容量
// val数组 每个物品的价值
int t, m;
std::cin >> t >> m;
std::vector<int> cost(m + 1), val(m + 1);
for(int i = 1; i <= m; i++) {
    std::cin >> cost[i] >> val[i];
}
std::vector<std::vector<long long>> dp(m + 1, std::vector<long
   long>(t + 1));
for(int i = 1; i <= m; i++) {
    for(int j = 0; j <= t; j++) {
        dp[i][j] = dp[i - 1][j];
        if(j - cost[i] >= 0) {
            dp[i][j] = std::max(dp[i][j], dp[i][j - cost[i]] +
               val[i]);
        }
    }
}
std::cout << dp[m][t];
```

8.6.2 空间优化

```cpp
// index-base 1
std::vector<long long> dp(t + 1);
for(int i = 1; i <= m; i++) {
    for(int j = cost[i]; j <= t; j++) {
        dp[j] = std::max(dp[j], dp[j - cost[i]] + val[i]);
    }
}
std::cout << dp[t];
```

常用 STL

表 1: 关键值说明表

值	含义	备注
区间	[first, last)	左闭右开
前缀 *val*	基本数据类型	与同行中的 *val_x* 同类型
前缀 *T*	任意数据类型	模板类
排序	排序操作相关	默认按照 $std :: less()$ 排序
compare	自定义比较类型	需重载操作符 () 或使用 lambda 表达式
iterator	迭代器类型	迭代器
dest	容器	目标容器
bid	$std :: back\_inserter(dest)$	一个定义

9.1 算法库

9.1.1 搜索操作

```cpp
// 判断区间的数是否全符合compare
bool all_of(first,last,compare)
// 判断区间的数是否存在符合compare
bool any_of(first,last,compare)
// 判断区间的数是否都不符合compare
bool none_of(first,last,compare)
// 返回区间中第一个等于val_a的位置,否则返回last
iterator find(first,last,val_a)
// 返回区间中第一个满足compare的位置,否则返回last
iterator find_if(first,last,compare)
```

```
// 返回区间中第一个不满足compare的位置,否则返回last
iterator find_if_not(first,last,compare)
// 返回区间中等于val_a的个数
size_t count(first,last,val_a)
// 返回区间中满足compare的个数
size_t count_if(first,last,compare)
```

9.1.2 交换操作

```
// 交换T_a和T_b之间的数据
void swap(T_a, T_b)
```

9.1.3 生成操作

```
// 填充区间中的值为val_a
void fill(first,last,val_a)
```

9.1.4 移除操作

```
// 对区间中重复的元素去重,返回重复元素的第一个位置
iterator unique(first,last)
```

9.1.5 顺序变更操作

```
// 翻转区间中的元素
void reverse(first,last)
```

9.1.6 划分操作

```
// 对区间中的数根据compare进行分割
// 返回不满足compare位置的起始位置的指针
iterator partition(first,last,compare)
```

9.1.7 排序操作

```
// 对区间的数以非降序排序
void sort(first,last)
// 对区间范围的数据根据compare规则进行排序
void sort(first,last,compare)
```

```
// 判断区间的数是否以非降序排序
bool is_sorted(first,last)
// 判断区间的数是否按照compare规则排序
bool is_sorted(first,last,compare)
```

9.1.8 二分搜索操作

```
// 返回第一个大于等于val_a元素的指针,否则返回last
iterator lower_bound(first,last,val_a)
// 返回第一个大于val_a元素的指针,否则返回last
iterator upper_bound(first,last,val_a)
```

9.1.9 集合操作（在已排序范围上）

```
// 将范围1和范围2中的元素取并集放在dest容器中
iterator set_union(first1,last1,first2,last2,bid)
// 将范围1和范围2中的元素取交集放在dest容器中
iterator set_intersection(first1,last1,first2,lats2,bid)
// 将范围1和范围2中的元素取差集放在dest容器中
iterator set_difference(first1,last1,first2,lats2,bid)
```

9.1.10 最小/最大操作

```
// 返回val_a,val_b间的最大
val max(val_a,val_b)
// 返回val_a,val_b间的最小
val min(val_a,val_b)
// 返回区间中的最小值的指针
iterator min_element(first,last)
// 返回区间中的最大值的指针
iterator max_element(first,last)
```

9.1.11 排列操作

```
// 对区间中的元素进行全排列, 通常结合do_while()使用;
bool next_permutation(first,last)
```

9.1.12 数值运算

```
// 对区间的数以初始值为val_a按顺序递增填充
void iota(first,last,val_a)
// 对区间中的数以val_a为初始值进行求和并返回
val accumulate(first,last,val_a)
```

9.1.13 位运算操作

```
// 返回x二进制下含1的数量,例如x=15=(1111)时答案为4
__builtin_popcount(x)
// 返回x右数第一个1的位置(1-idx), 1(1) 返回1, 8(1000) 返回4,
    26(11010)返回2
__builtin_ffs(x)
// 返回x二进制下后导0的个数, 1(1) 返回0, 8(1000) 返回3
__builtin_ctz(x)
// 返回x二进制下的位数, 9(1001) 返回4, 26(11010) 返回5
bit_width(x)
```

常见定理

10.1 数学定理

定理 10.1 (欧几里得). 若 a, b 为整数，$b \neq 0$，则有 $\gcd(a,b) = \gcd(b, a \bmod b)$。

```
// 求最大公约数
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

定理 10.2 (扩展欧几里得). 设 $a, b \in \mathbb{Z}$ 且不全为零，则存在整数 x, y 使得：

$$ax + by = \gcd(a,b)$$

```
// 求一组解 (x, y) 满足 ax + by = gcd(a, b)
int exgcd(int a, int b, int &x, int &y) {
    if (!b) { x = 1; y = 0; return a; }
    int d = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return d;
}
```

定理 10.3 (模逆元)**.** 若 $\gcd(a, m) = 1$，则存在整数 x，使得：

$$ax \equiv 1 \pmod{m}$$

称 x 为 a 在模 m 下的乘法逆元

```cpp
// 计算 a 在 mod m 下的逆元（gcd(a, m) == 1）
int modInverse(int a, int m) {
    int x, y;
    int d = exgcd(a, m, x, y);
    if (d != 1) return -1;
    return (x % m + m) % m;
}
```

定理 10.4 (线性不定方程求解)**.** 方程 $ax + by = c$ 有整数解当且仅当 $\gcd(a, b) \mid c$。
如果 $ax + by = d$　d 为 $\gcd(a, b)$，其中一个特解是 (x_0, y_0)
则通解为：

$$x = x_0 + \frac{b}{d} * n \qquad y = y_0 - \frac{a}{d} * n \quad n \in \mathbb{Z}$$

如果 $ax + by = c$　c 为 d 的整数倍，根据上面的特解，可以得到该等式的一个特解 (x_0', y_0')
其中 $x_0' = x_0 * \frac{c}{d}$　$y_0' = y_0 * \frac{c}{d}$
则通解为：

$$x = x_0' + \frac{b}{d} * n \quad y = y_0' - \frac{a}{d} * n \quad n \in \mathbb{Z}$$

```cpp
// 解线性不定方程，返回一组整数解
bool solveDiophantine(int a, int b, int c, int &x, int &y) {
    int d = exgcd(a, b, x, y);
    if (c % d != 0) return false;
    int k = c / d;
    x *= k;
    y *= k;
    return true;
}
```

定理 10.5 (解模线性方程)**.** 模线性方程 $ax \equiv b \pmod{m}$ 有解当且仅当 $gcd(a, m) \mid b$。

```cpp
// 解 ax  b mod m 的最小正整数解
int modLinearSolve(int a, int b, int m) {
    int x, y;
    int d = exgcd(a, m, x, y);
    if (b % d != 0) return -1;
    x = x * (b / d);
    return (x % (m / d) + (m / d)) % (m / d);
}
```

定理 10.6 (中国剩余定理). 若 $\{m_i\}$ 两两互质，则同余方程组

$$x \equiv a_i \pmod{m_i}, \quad i = 1, 2, ..., n$$

有唯一解 $x \mod M$，其中 $M = \prod m_i$。

```cpp
// CRT 模数互质情况，返回最小非负解 x
using long long = i64;

i64 CRT(const std::vector<int>& a, const std::vector<int>& m) {
    i64 M = 1;
    for (int mi : m) M *= mi;

    i64 res = 0;
    for (int i = 0; i < m.size(); ++i) {
        i64 Mi = M / m[i];
        int inv = modInverse(Mi % m[i], m[i]);
        res = (res + 1LL * a[i] * Mi % M * inv % M) % M;
    }
    return (res + M) % M;
}
```

定理 10.7 (扩展中国剩余定理). 对于模数不互质的同余方程组，若

$$\gcd(m_i, m_j) \mid (a_i - a_j), \quad \forall i, j$$

则存在整数解 x，并可通过逐步合并方式递推构造。

```cpp
// exCRT 模数不互质情况，返回最小非负解 x
using long long = i64;

i64 exCRT(const std::vector<i64>& a, const std::vector<i64>& m)
    {
    i64 x = a[0], mod = m[0];
    for (int i = 1; i < a.size(); ++i) {
        i64 a1 = mod, a2 = m[i];
        i64 b = (a[i] - x % a2 + a2) % a2;

        i64 s, t;
        i64 d = exgcd(a1, a2, s, t);
        if (b % d != 0) return -1;

        i64 k = b / d;
        s = (s % a2 + a2) % a2;
        i64 tmp = (k * s) % (a2 / d);
        x = x + tmp * mod;
        mod = mod / d * a2;
        x = (x % mod + mod) % mod;
    }
    return x;
}
```

定义 10.1 (欧拉函数). 对正整数 n, $\varphi(n)$ 表示不超过 n 且与 n 互质的正整数个数。

```
// 单点欧拉函数
int euler(int n) {
    int res = n;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            res = res / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    if (n > 1) res = res / n * (n - 1);
    return res;
}
```

定理 10.8 (鞋带公式). 设有多边形的顶点 $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$，按照顺时针或逆时针顺序排列，闭合多边形的面积 A 为：

$$A = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right|$$

其中 (x_{n+1}, y_{n+1}) 被认为是 (x_1, y_1)。

```
// 鞋带公式计算多边形面积
double shoelaceFormula(std::vector<pair<int, int>>& points) {
    int n = points.size();
    double area = 0.0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        area += (points[i].first * points[j].second - points[i
            ].second * points[j].first);
    }
    return abs(area) / 2.0;
}
```

定理 10.9 (Pick 定理). 设一个简单多边形的顶点均为整数坐标，且该多边形的面积为 A，内部的格点数为 I，边上的格点数为 B，则有：

$$A = I + \frac{B}{2} - 1$$

其中 A 表示多边形面积，I 表示多边形内部的格点数，B 表示边上的格点数。

```
// Pick 定理的计算：给定顶点和边界点数，计算面积
int pickTheorem(int I, int B) {
    return I + B / 2 - 1;
}
```

定理 10.10 (二项式定理). 对于任意实数 x 和 y，以及非负整数 n，有：

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

其中 $\binom{n}{k}$ 为二项式系数，表示为：

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

定理 10.11 (二项式反演). 二项式反演的四种形式

$$g(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} f(i) \iff f(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} g(i) \tag{1}$$

$$g(n) = \sum_{i=0}^{n} \binom{n}{i} f(i) \iff f(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} g(i) \tag{2}$$

$$g(n) = \sum_{i=n}^{N} (-1)^i \binom{i}{n} f(i) \iff f(n) = \sum_{i=n}^{N} (-1)^i \binom{i}{n} g(i) \tag{3}$$

$$g(n) = \sum_{i=n}^{N} \binom{i}{n} f(i) \iff f(n) = \sum_{i=n}^{N} (-1)^{i-n} \binom{i}{n} g(i) \tag{4}$$

定理 10.12 (卢卡斯定理). 若 p 为质数，n 和 k 为非负整数，则二项式系数 $\binom{n}{k} \pmod p$ 可以通过以下递归关系计算：

$$\binom{n}{k} \equiv \prod_{i=0}^{m} \binom{n_i}{k_i} \pmod p$$

其中，n_i 和 k_i 是 n 和 k 在基 p 下的每一位的数值。

定理 10.13 (卡特兰数). 卡特兰公式：

$$f(n) = \binom{2n}{n} - \binom{2n}{n-1} \tag{1}$$

$$f(n) = \frac{\binom{2n}{n}}{(n+1)} \tag{2}$$

$$f(n) = f(n-1) * \frac{4n-2}{n+1} \tag{3}$$

$$f(n) = \sum_{i=0}^{n-1} f(i) * f(n-1-i) \tag{4}$$