目录

数学

扩展欧几里得 Exgcd

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

using namespace std;

struct Exgcd{
    int a,b,k;
    //类内没有判断无解，注意在外面判断是否有 k % __gcd(a,b) == 0
    Exgcd(int a,int b,int k):a(a),b(b),k(k){}
    array<__int128,4> work(){
        __int128 g = __gcd(a,b);
```

```
        int x,y;
        auto exgcd = [&](auto &&self,int a,int b){
            if(b == 0){
                x = 1;
                y = 0;
                return;
            }
            self(self,b,a%b);
            int t = x;
            x = y;
            y = t - a/b*y;
        };
        exgcd(exgcd,a,b);
        __int128 dx = b/g,dy = -a/g;
        __int128 x0 = (__int128)x*(k/g),y0 = (__int128)y*(k/g);
        return {x0,y0,dx,dy};
    }
};
```

数论分块

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
void solve(){
    int n,k;
    cin >> n >> k;
    //下取整
    long long r;
    for(int l = 1; l <= n; l = r + 1)
    {
        if(k/l == 0)
        {
            r = 8e18;
        }
        else r = min(k/(k/l),n);
    }

    //上取整
    long long r;
    for(int l = 1; l <= n ; l = r + 1)
    {
        if((k-1)/l == 0)
        {
            r = 8e18;
```

```
        }
        else r = min((k-1)/((k-1)/l),n);
    }
}
```

求组合数

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int mod;
struct Comb{
    int n;
    vector<int> fact,infact;
    Comb(int n1):n(n1),fact(n1+2),infact(n1+2){
        init();
    }
    int qmi(int a,int b)
    {
        int ret = 1;
        while(b)
        {
            if(b & 1) ret = (1ll * ret * a) % mod;
            a = (1ll*a*a) % mod;
            b >>= 1;
        }
        return ret;
    }
    void init()
    {
        fact[0] = infact[0] = 1;
        for(int i = 1; i <= n; i ++)
        {
            fact[i] = (1ll*fact[i-1]*i) % mod;
            infact[i] =(1ll*infact[i-1]*qmi(i,mod-2)) % mod;
            //cout << fact[i] << ' ';
        }
    }
    int C(int a,int b)
    {
        return (((1ll*fact[a]*infact[b])%mod)*infact[a-b]) % mod;
    }
}comb(5e5 + 10);
```

三分

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

void solve(){
    auto check = [&](double x) -> double{

    };
    double l = -2e6,r = 2e6;
    while(r - l > 1e-9) {
        double lmid = l + (r - l) / 3, rmid = r - (r - l) / 3;

        // 求凹函数的极小值
        if(check(lmid) <= check(rmid)) r = rmid;
        else l = lmid;

         // 求凸函数的极大值
        if(check(lmid) <= check(rmid)) l = lmid;
        else r = rmid;

    }
    auto check = [&](int x)->int{

    };

    int l = -2e6,r = 2e6;
    while(l < r){
        int lmid = l + (r-l)/3, rmid = r - (r-l)/3;

        //如果是中间凸的单峰函数是<号，中间凹是>号。
        //求凹函数的极小值
        if(check(lmid) > check(rmid)) l = lmid + 1;
        else r = rmid - 1;

        //求凸函数的极大值
        if(check(lmid) < check(rmid)) l = lmid + 1;
        else r = rmid - 1;
    }

}
```

矩阵快速幂

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

struct Matrix{
    int n,m;
    vector<vector<int>> a;
    Matrix(int n1,int m1) : n(n1),m(m1),a(n1+1,vector<int>(m1+1)){}
    vector<int>& operator[] (int i) { return a[i]; }
    Matrix operator*(const Matrix &r) const {
        Matrix ret(n,r.m);
        for(int i = 1; i <= n; i ++)
            for(int j = 1; j <= r.m; j ++)
            {
                for(int k = 1; k <= m; k ++)
                {
                    ret.a[i][j] = (ret.a[i][j] + a[i][k]*r.a[k][j]) % mod;
                }
            }
        return ret;
    }
};
 auto M_qmi = [&](Matrix a,int b){
        Matrix ret(a.n,a.n);
        for(int i = 1; i <= a.n; i ++) ret.a[i][i] = 1;
        while(b)
        {
            if(b&1) ret = ret * a;
            a = a * a;
            b >>= 1;
        }
        return ret;
    };
```

图论

Dijkstra

```
#include<bits/stdc++.h>
#define int long long
```

```
using namespace std;
typedef pair<int,int> PII;
void solve(){
    int n;
    vector<long long> dist(n+1,2e18),st(n+1);
    auto dijkstra = [&](int s) -> void
    {
        priority_queue<PII,vector<PII>,greater<PII>> heap;
        dist[s] = 0;
        heap.push({0,s});
        while(!heap.empty())
        {
            auto [fi,se] = heap.top();
            heap.pop();
            if(st[se]) continue;
            st[se] = 1;
            for(auto [d,i] : v[se])
            {
                if(dist[i] > fi + d)
                {
                    dist[i] = fi + d;
                    heap.push({dist[i],i});
                }
            }
        }
        return;
    };
    dijkstra(s);
}
```

Floyd 多源最短路

```
#include<bits/stdc++.h>
#define int long long

using namespace std;
void solve(){
    int n;
    vector<vector<int>> dis(n+1,vector<int>(n+1,1e18));
    auto floyd = [&](){
        for(int i = 1; i <= n; i ++)
            for(int j = 1; j <= n; j ++)
                for(int k = 1; k <= n; k ++){
                    dis[j][k] = min(dis[j][k],dis[j][i] + dis[i][k]);
                }
```

```
    };
    floyd();
}
```

拓扑排序

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

// 拓扑排序 (有向图)
    vector<int> onloop(n+1,1);
    queue<int> q;
    for(int i = 1; i <= n; i ++){
        if(!deg[i]) {
            q.push(i);
            onloop[i] = 0;
        }
    }
    while(q.size()){
        int t = q.front();
        q.pop();
        for(int i : v[t]){
            deg[i] --;
            if(!deg[i]){
                q.push(i);
                onloop[i] = 0;
            }
        }
    }
    // if(deg[i] >= 1)

// 拓扑排序 (无向图) 找环

    vector<int> deg(n+1);
    queue<int> q;
    for(int i = 1; i <= n; i ++)
    {
        deg[i] = v[i].size();
        if(deg[i] <= 1) q.push(i);
    }
    while(!q.empty())
    {
        auto t = q.front();
```

```
        q.pop();
        for(auto i : v[t])
        {
            deg[i] --;
            if(deg[i] == 1)
            {
                q.push(i);
            }
        }
    }
```

有向图强连通分量 SCC

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

struct SCC{
    int n;
    vector<vector<int>> &v;
    vector<int> stk;
    vector<int> dfn,low,bel;
    int cur,cnt;

    SCC(vector<vector<int>> &g) : v(g){
        n = g.size() - 1;
        stk.resize(n+1);
        dfn.resize(n+1);
        low.resize(n+1);
        bel.resize(n+1);
        cur = cnt = 0;
    }

    void tarjan(int u){
        dfn[u] = low[u] = ++ cur;
        stk.push_back(u);
        for(int i : v[u]){
            if(!dfn[i]){
                tarjan(i);
                low[u] = min(low[u],low[i]);
            }else if(!bel[i]){
                low[u] = min(low[u],dfn[i]);
            }
        }
```

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

    vector<int> work(){
        for(int i = 1; i <= n; i ++){
            if(!dfn[i]) tarjan(i);
        }
        return bel;
    }

    vector<vector<int>> rebuild(){
        vector<vector<int>> g(cnt + 1);
        set<pair<int,int>> s;
        for(int i = 1; i <= n; i ++){
            for(int j : v[i]){
                if(bel[j] != bel[i]){
                    s.insert({bel[i],bel[j]});
                }
            }
        }
        for(auto [i,j] : s){
            g[i].push_back(j);
        }
        return g;
    }
};
```

无向图边双连通分量 EBCC

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

struct EBCC{
```

```
int n;
vector<vector<int>> &v;
vector<int> dfn,low,stk,bel;
int cur,cnt;
EBCC(vector<vector<int>> &g) : v(g){
    n = g.size()-1;
    dfn.resize(n+1);
    low.resize(n+1);
    bel.resize(n+1);
    cur = cnt = 0;
}

void tarjan(int u,int p){
    low[u] = dfn[u] = ++ cur;
    stk.push_back(u);
    bool flag = false;
    for(int i : v[u]){
        if(i == p && !flag){
            flag = 1;
            continue;
        }
        if(!dfn[i]){
            tarjan(i,u);
            low[u] = min(low[u],low[i]);
        }
        else low[u] = min(low[u],dfn[i]);
    }
    if(dfn[u] == low[u]){
        cnt ++;
        int x;
        do{
            x = stk.back();
            bel[x] = cnt;
            stk.pop_back();
        }while(x != u);
    }
}

void work(){
    for(int i = 1; i <= n; i ++){
        if(!dfn[i]){
            tarjan(i,-1);
        }
    }
}
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 + 1;
        g.siz.resize(cnt + 1);
        g.cnte.resize(cnt + 1);
        for (int i = 1; i <= n; i++) {
            g.siz[bel[i]]++;
            for (auto j : v[i]) {
                if (bel[i] < bel[j]) {
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

无向图点双连通分量 VBCC

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

struct VBCC{
    int n;
    vector<vector<int>> &v;
    vector<int> dfn,low,stk,flag;
    vector<vector<int>> vbcc;//点双连通分量
    int cur;
    VBCC(vector<vector<int>> &g) : v(g){
        n = g.size()-1;
        dfn.resize(n+1);
        low.resize(n+1);
        flag.resize(n+1);//是否为割点
        cur = 0;
    }
```

```
    void tarjan(int u,int p){
        int son = 0;
        low[u] = dfn[u] = ++ cur;
        stk.push_back(u);
        for(int i : v[u]){
            if(!dfn[i]){
                son ++;
                tarjan(i,u);
                low[u] = min(low[u],low[i]);
                if(low[i] >= dfn[u]){
                    vector<int> tmp;
                    int x;
                    do{
                        x = stk.back();
                        tmp.push_back(x);
                        stk.pop_back();
                    }while(x != i);
                    if(p != -1)flag[u] = 1;
                    tmp.push_back(u);
                    vbcc.push_back(tmp);
                }
            }
            else if(i != p) low[u] = min(low[u],dfn[i]);
        }
        if(p == -1 && son == 0) {
            vbcc.push_back({u});
        }
        if(p == -1 && son >= 2) flag[u] = 1;
    }

    void work(){
        for(int i = 1; i <= n; i ++){
            if(!dfn[i]){
                tarjan(i,-1);
            }
        }
    }

};
```

最近公共祖先 LCA-jump

```
#include<bits/stdc++.h>
#define int long long
```

```cpp
using namespace std;

struct LCA{
    int n,k;
    vector<vector<int>> &v;
    vector<array<int,32>> fa;
    vector<int> depth;
    LCA(vector<vector<int>> &g): v(g){
        n = g.size()-1;
        k = __lg(n);
        fa.resize(n+1,{});
        depth.resize(n+1);
    }
    void dfs(int u,int p){
        depth[u] = depth[p] + 1;
        fa[u][0] = p;
        for(int i = 1; i <= k; i ++){
            fa[u][i] = fa[fa[u][i-1]][i-1];
        }
        for(int i : v[u]){
            if(i != p) dfs(i,u);
        }
    }

    void work(int s){
        dfs(s,0);
    }

    int jump(int u,int dis){
        if(dis == 0) return u;
        int t = depth[u] - dis, p = u;
        for(int i = k; i >= 0; i --){
            if(depth[fa[p][i]] > t) p = fa[p][i];
        }
        return fa[p][0];
    }

    int getlca(int a, int b) {
        if(depth[a] != depth[b]){
            if (depth[a] < depth[b])
                swap(a, b);
            for (int i = k ; i >= 0; i --) {
                if (depth[fa[a][i]] > depth[b])
                    a = fa[a][i];
            }
```

```
            a = fa[a][0];
        }
        if (a == b)
            return a;
        for (int i = k ; i >= 0 ; i --) {
            if (fa[a][i] != fa[b][i]) {
                a = fa[a][i];
                b = fa[b][i];
            }
        }
        return fa[a][0];
    }

};
```

最近公共祖先 LCA-Euler

```
#include<bits/stdc++.h>
#define int long long

using namespace std;
typedef pair<int,int> PII;
struct LCA{
    int n,k,tot;
    vector<vector<PII>> &v;//无权边时，此处改为 int
    vector<int> dfn,depth,pos;
    vector<array<int,24>> st,idx;
    LCA(vector<vector<PII>> &g): v(g){//无权边时，此处改为 int
        n = g.size()-1;
        k = __lg(2*n+1);
        pos.resize(n+1);
        depth.resize(n+1);
        dfn.resize(2*n+1);
        st.resize(2*n+1,{});
        idx.resize(2*n+1,{});
        tot = 0;
    }

    void dfs(int u,int p,int dis){
        dfn[++tot] = u;
        pos[u] = tot;
        depth[u] = dis;
        for(auto [d,i] : v[u]){//无边权，此处改为 auto i
            if(i != p){
                dfs(i,u,dis + d);//无边权时，此处改为 dis + 1
                dfn[++tot] = u;
            }
```

```
        }
    }

    void work(int s){
        dfs(s,-1,1);

        for(int i = 1; i <= tot; i++){
            st[i][0] = depth[dfn[i]];
            idx[i][0] = dfn[i];
        }

        for(int j = 1; j <= k; j++){
            for(int i = 1; i + (1 << j) - 1 <= tot; i++){
                if(st[i][j-1] < st[i + (1 << (j-1))][j-1]){
                    st[i][j] = st[i][j-1];
                    idx[i][j] = idx[i][j-1];
                }else{
                    st[i][j] = st[i + (1 << (j-1))][j-1];
                    idx[i][j] = idx[i + (1 << (j-1))][j-1];
                }
            }
        }
    }

    int query(int x,int y){
        int l = pos[x], r = pos[y];
        if(l > r) swap(l, r);
        int len = __lg(r - l + 1);

        if(st[l][len] < st[r - (1 << len) + 1][len]){
            return idx[l][len];
        }else{
            return idx[r - (1 << len) + 1][len];
        }
    }

    int dis(int x,int y){
        return depth[x] + depth[y] - 2*depth[query(x,y)];
    }
};
```

2-SAT

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

```cpp
const int N = 1e6 + 10;
int dfn[N],low[N],cnt;
stack<int> stk;
bool ins[N];
int id[N],scc_cnt,sz[N];
vector<int> g[N];
int n,m;

void tarjan(int u)
{
    dfn[u] = low[u] = cnt ++;
    stk.push(u);
    ins[u] = 1;
    for(auto j : g[u]){
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u],low[j]);
        }
        else if(ins[j])
        {
            low[u] = min(low[u],dfn[j]);
        }
    }
    if(dfn[u] == low[u])
    {
        int y;
        ++scc_cnt;
        do{
            y = stk.top();
            stk.pop();
            ins[y] = 0;
            id[y] = scc_cnt;
            sz[scc_cnt] ++;
        }while(y != u);
    }
}



    //判断有无解
    for(int i = 1; i <= 2*n; i ++)
    {
        if(!dfn[i]) tarjan(i);
    }
    for(int i = 1; i <= n; i ++)
```

```
    {
        if(id[i] == id[i+n])
        {
            cout << "NO\n";
            return;
        }
    }
    //存储答案
    vector<int> ans;
    for(int i = 1; i <= n; i ++)
    {
        if(id[i] < id[i + n])
            ans.push_back(i);
    }
```

树上启发式合并

```
#include<bits/stdc++.h>
#define int long long

using namespace std;
typedef function<void(int)> FVI;
struct DSUtree{
    int n;
    vector<vector<int>> v;
    vector<int> son;

    FVI addp,clrp,getans;

    DSUtree(vector<vector<int>> &v,FVI addp,FVI clrp,FVI
getans):v(v),addp(addp),clrp(clrp),getans(getans){
        n = v.size() - 1;
        son.resize(n+1);
    }

    void work(int s){
        dfs0(s,-1);
        dfs1(s,-1);
    }

    int dfs0(int u,int p){
        int s = -1,mx = -1,cnt = 0;
        for(int i : v[u]){
            if(i != p){
                int ret = dfs0(i,u);
```

```
                cnt += ret;
                if(ret > mx){
                    s = i;
                    mx = ret;
                }
            }
        }
        son[u] = s;
        return cnt + 1;
    }

    void add(int u,int p){
        addp(u);
        for(int i : v[u]){
            if(i != p) add(i,u);
        }
    }

    void clr(int u,int p){
        clrp(u);
        for(int i : v[u]){
            if(i != p) clr(i,u);
        }
    }

    void dfs1(int u,int p){
        for(int i : v[u]){
            if(i != p && i != son[u]){
                dfs1(i,u);
                clr(i,u);
            }
        }
        if(son[u] != -1) dfs1(son[u],u);
        addp(u);
        for(int i : v[u]){
            if(i != p && i != son[u]){
                add(i,u);
            }
        }
        getans(u);
    }

};

void solve(){
```

```
    vector<vector<int>> v;

    auto add = [&](int u) -> void{
        /*填入添加单个点的操作*/
    };
    auto clr = [&](int u) -> void{
        /*填入删除单个点的操作*/
    };
    auto getans = [&](int u) -> void{
        /*统计单点答案的操作*/
    };
    DSUtree dsutree(v,add,clr,getans);
    dsutree.work(1);

}
```

数据结构

树状数组

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

struct Fenwick{
    const int len;
    vector<int> a;
    Fenwick(int n1) : len(n1),a(len + 1,0){}
    #define lowbit(x) ((x) & (-x))
    void init(vector<int> &b)
    {
        for(int i = 1; i <= len; i ++)
        {
            a[i] += b[i];
            int j = i + lowbit(i);
            if(j <= len) a[j] += a[i];
        }
    }
    void add(int x,int c)
    {
        for(int i = x; i <= len; i += lowbit(i))
```

```
            a[i] += c;
    }
    int sum(int x)
    {
        int ret = 0;
        for(int i = x; i; i -= lowbit(i)) ret += a[i];
        return ret;
    }
    int sum(int l,int r)
    {
        return sum(r) - sum(l - 1);
    }
};
```

线段树

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

struct info {
    //需要修改! ! !
    int l,r;
    int sum;
    int mi,mx;
};
info operator+ (const info &l,const info &r) {
    //需要修改! ! !
    info a;
    a.l = min(l.l,r.l),a.r = max(l.r,r.r);
    a.sum = l.sum + r.sum;
    a.mx = max(l.mx,r.mx);
    a.mi = min(l.mi,r.mi);
    return a;
}
struct Segtree{
    const int len;
    vector<info> seg;
    vector<int> tag;
    Segtree(int n1) :len(n1),seg(4*n1+1),tag(4*n1+1) {}
    void add(int np,int v)
    {
        //需要修改! ! !
        tag[np] += v;
        seg[np].mx += v;
```

```
    seg[np].mi -= v;
    seg[np].sum += (seg[np].r-seg[np].l+1)*v;
}
void push(int np)
{
    add(np << 1,tag[np]),add(np << 1 | 1,tag[np]);
    tag[np] = 0;
}
void pull(int id) {
    seg[id] = seg[id << 1] + seg[id << 1 | 1];
}
void init(vector<int> init)
{
    auto build = [&](auto &&self,int np,int l,int r) -> void
    {
        if (l == r) seg[np] = {l, r, init[r]};
        else
        {
            seg[np] = {l, r};
            int mid = l + r >> 1;
            self(self,np << 1, l, mid), self(self,np << 1 | 1, mid + 1, r);
            pull(np);
        }
    };
    build(build,1,0,len);
}
void init(){
    auto build = [&](auto &&self,int np,int l,int r) -> void
    {
        if (l == r) seg[np] = {l, r, 0};
        else
        {
            seg[np] = {l, r};
            int mid = l + r >> 1;
            self(self,np << 1, l, mid), self(self,np << 1 | 1, mid + 1, r);
            pull(np);
        }
    };
    build(build,1,0,len);
}
void modify(int np,int x, int v)
{
    if(seg[np].l == seg[np].r)
    {
        //需要修改！！！
        seg[np].sum += v;
```

```
            seg[np].mx += v;
            seg[np].mi += v;
        }
        else
        {
            int mid = seg[np].l + seg[np].r >> 1;
            push(np);
            if(x <= mid) modify(np << 1,x,v);
            else modify(np << 1 | 1,x,v);
            pull(np);
        }
    }
    void rangeadd(int np,int x,int y,int v)
    {
        if(seg[np].l >= x && seg[np].r <= y)
        {
            add(np,v);
        }
        else{
            push(np);
            int mid = seg[np].l + seg[np].r >> 1;
            if(x <= mid) rangeadd(np << 1,x,y,v);
            if(y > mid) rangeadd(np << 1 | 1,x,y,v);
            pull(np);
        }
    }
    info query(int np,int x,int y)
    {
        if(seg[np].l >= x && seg[np].r <= y) return seg[np];
        push(np);
        int mid = seg[np].l + seg[np].r >> 1;
        if(x <= mid && y > mid)
        {
            return query(np << 1,x,y) + query(np << 1 | 1,x,y);
        }
        else if(x <= mid) return query(np << 1,x,y);
        else return query(np << 1 | 1,x,y);
    }
};
```

并查集

```
#include<bits/stdc++.h>
#define int long long
```

```cpp
using namespace std;

struct Dsu{
    int n;
    vector<int> f,siz;
    Dsu(int _n) : n(_n),f(n+1){
        iota(f.begin()+1,f.end(),1);
        siz.assign(n+1,1);
    }

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

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

    void merge(int x,int y)
    {
        x = find(x),y = find(y);
        if(x != y){
            siz[x] += siz[y];
            f[y] = x;
        }
    }

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

字符串

哈希

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

```cpp
struct Hash
{
    vector<int> h1,h2,p1,p2;
    int n;
    const int P = 13331;
    const int mod1 = 998244353,mod2 = 998244853;
    Hash (string &s)
    {
        //多个字符串哈希，p1，p2 可以只算一次，拆散了用，就行
        n = s.size();
        h1.resize(n+1);
        h2.resize(n+1);
        p1.resize(n+1);
        p2.resize(n+1);
        p1[0] = p2[0]= h1[0] = h2[0] = 1;
        for (int i = 1 ; i <= n; i ++) {
            h1[i] = (1ll*h1[i - 1] * P + s[i]) % mod1;
            h2[i] = (1ll*h2[i - 1] * P + s[i]) % mod2;
            p1[i] = (1ll*p1[i - 1] * P) % mod1;
            p2[i] = (1ll*p2[i - 1] * P) % mod2;
        }
    }
    pair<int,int> check(int l,int r)
    {
        return {(h1[r] - (1ll*h1[l - 1] * p1[r - l + 1]) % mod1 + mod1) % mod1,
            (h2[r] - (1ll * h2[l - 1] * p2[r - l + 1]) % mod2 + mod2) % mod2};
    }

};
```

Trie

```cpp
#include <iostream>

using namespace std;

const int N = 100010;

int son[N][26], cnt[N], idx;
char str[N];

void insert(char *str)
{
    int p = 0;
    for (int i = 0; str[i]; i ++ )
```

```
    {
        int u = str[i] - 'a';
        if (!son[p][u]) son[p][u] = ++ idx;
        p = son[p][u];
    }
    cnt[p] ++ ;
}

int query(char *str)
{
    int p = 0;
    for (int i = 0; str[i]; i ++ )
    {
        int u = str[i] - 'a';
        if (!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}

int main()
{
    int n;
    scanf("%d", &n);
    while (n -- )
    {
        char op[2];
        scanf("%s%s", op, str);
        if (*op == 'I') insert(str);
        else printf("%d\n", query(str));
    }

    return 0;
}
```

计算几何

```
#include <bits/stdc++.h>
using std::numeric_limits;
using std::abs, std::max, std::min, std::swap;
using std::pair, std::make_pair;
using std::tuple, std::make_tuple;
using std::vector, std::deque;
using std::set, std::multiset;
```

```cpp
using T=long double;  //全局数据类型

constexpr T eps=1e-8;
constexpr T INF=numeric_limits<T>::max();
constexpr T PI=3.14159265358979323846l;

// 点与向量
struct Point
{
    T x,y;

    bool operator==(const Point &a) const {return (abs(x-a.x)<=eps && abs(y-a.y)<=eps);}
    bool operator<(const Point &a) const {if (abs(x-a.x)<=eps) return y<a.y-eps; return x<a.x-eps;}
    bool operator>(const Point &a) const {return !(*this<a || *this==a);}
    Point operator+(const Point &a) const {return {x+a.x,y+a.y};}
    Point operator-(const Point &a) const {return {x-a.x,y-a.y};}
    Point operator-() const {return {-x,-y};}
    Point operator*(const T k) const {return {k*x,k*y};}
    Point operator/(const T k) const {return {x/k,y/k};}
    T operator*(const Point &a) const {return x*a.x+y*a.y;}  // 点积
    T operator^(const Point &a) const {return x*a.y-y*a.x;}  // 叉积，注意优先级
    int toleft(const Point &a) const {const auto t=(*this)^a; return (t>eps)-(t<-eps);}  // to-left 测试
    T len2() const {return (*this)*(*this);}  // 向量长度的平方
    T dis2(const Point &a) const {return (a-(*this)).len2();}  // 两点距离的平方
    int quad() const // 象限判断 0:原点 1:x 轴正 2:第一象限 3:y 轴正 4:第二象限 5:x 轴负 6:第三象限 7:y 轴负 8:第四象限
    {
        if (abs(x)<=eps && abs(y)<=eps) return 0;
        if (abs(y)<=eps) return x>eps ? 1 : 5;
        if (abs(x)<=eps) return y>eps ? 3 : 7;
        return y>eps ? (x>eps ? 2 : 4) : (x>eps ? 8 : 6);
    }

    // 必须用浮点数
    T len() const {return sqrtl(len2());}  // 向量长度
    T dis(const Point &a) const {return sqrtl(dis2(a));}  // 两点距离
    T ang(const Point &a) const {return acosl(max(-1.0l,min(1.0l,((*this)*a)/(len()*a.len()))));}  // 向量夹角
    Point rot(const T rad) const {return {x*cos(rad)-y*sin(rad),x*sin(rad)+y*cos(rad)};}  // 逆时针旋转（给定角度）
    Point rot(const T cosr,const T sinr) const {return {x*cosr-y*sinr,x*sinr+y*cosr};}  // 逆时针旋转（给定角度的正弦与余弦）
};

// 极角排序
struct Argcmp
```

```
{
    bool operator()(const Point &a,const Point &b) const
    {
        const int qa=a.quad(),qb=b.quad();
        if (qa!=qb) return qa<qb;
        const auto t=a^b;
        // if (abs(t)<=eps) return a*a<b*b-eps;  // 不同长度的向量需要分开
        return t>eps;
    }
};

// 直线
struct Line
{
    Point p,v;  // p 为直线上一点，v 为方向向量

    bool operator==(const Line &a) const {return v.toleft(a.v)==0 && v.toleft(p-a.p)==0;}
    int toleft(const Point &a) const {return v.toleft(a-p);}  // to-left 测试
    bool operator<(const Line &a) const  // 半平面交算法定义的排序
    {
        if (abs(v^a.v)<=eps && v*a.v>=-eps) return toleft(a.p)==-1;
        return Argcmp()(v,a.v);
    }

    // 必须用浮点数
    Point inter(const Line &a) const {return p+v*((a.v^(p-a.p))/(v^a.v));}  // 直线交点
    T dis(const Point &a) const {return abs(v^(a-p))/v.len();}  // 点到直线距离
    Point proj(const Point &a) const {return p+v*((v*(a-p))/(v*v));}  // 点在直线上的投影
};

//线段
struct Segment
{
    Point a,b;

    bool operator<(const Segment &s) const {return make_pair(a,b)<make_pair(s.a,s.b);}

    // 判定性函数建议在整数域使用

    // 判断点是否在线段上
    // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
    int is_on(const Point &p) const
    {
        if (p==a || p==b) return -1;
        return (p-a).toleft(p-b)==0 && (p-a)*(p-b)<-eps;
    }
```

```
    // 判断线段直线是否相交
    // -1 直线经过线段端点 | 0 线段和直线不相交 | 1 线段和直线严格相交
    int is_inter(const Line &l) const
    {
        if (l.toleft(a)==0 || l.toleft(b)==0) return -1;
        return l.toleft(a)!=l.toleft(b);
    }

    // 判断两线段是否相交
    // -1 在某一线段端点处相交 | 0 两线段不相交 | 1 两线段严格相交
    int is_inter(const Segment &s) const
    {
        if (is_on(s.a) || is_on(s.b) || s.is_on(a) || s.is_on(b)) return -1;
        const Line l{a,b-a},ls{s.a,s.b-s.a};
        return l.toleft(s.a)*l.toleft(s.b)==-1 && ls.toleft(a)*ls.toleft(b)==-1;
    }

    // 点到线段距离（必须用浮点数）
    T dis(const Point &p) const
    {
        if ((p-a)*(b-a)<-eps || (p-b)*(a-b)<-eps) return min(p.dis(a),p.dis(b));
        const Line l{a,b-a};
        return l.dis(p);
    }

    // 两线段间距离（必须用浮点数）
    T dis(const Segment &s) const
    {
        if (is_inter(s)) return 0;
        return min({dis(s.a),dis(s.b),s.dis(a),s.dis(b)});
    }
};

// 多边形
struct Polygon
{
    vector<Point> p;  // 以逆时针顺序存储

    size_t nxt(const size_t i) const {return i==p.size()-1?0:i+1;}
    size_t pre(const size_t i) const {return i==0?p.size()-1:i-1;}

    // 回转数
    // 返回值第一项表示点是否在多边形边上
    // 对于狭义多边形，回转数为 0 表示点在多边形外，否则点在多边形内
    pair<bool,int> winding(const Point &a) const
```

```
    {
        int cnt=0;
        for (size_t i=0;i<p.size();i++)
        {
            const Point u=p[i],v=p[nxt(i)];
            if (abs((a-u)^(a-v))<=eps && (a-u)*(a-v)<=eps) return {true,0};
            if (abs(u.y-v.y)<=eps) continue;
            const Line uv={u,v-u};
            if (u.y<v.y-eps && uv.toleft(a)<=0) continue;
            if (u.y>v.y+eps && uv.toleft(a)>=0) continue;
            if (u.y<a.y-eps && v.y>=a.y-eps) cnt++;
            if (u.y>=a.y-eps && v.y<a.y-eps) cnt--;
        }
        return {false,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
    T area() const
    {
        T sum=0;
        for (size_t i=0;i<p.size();i++) sum+=p[i]^p[nxt(i)];
        return sum;
    }

    // 多边形的周长
    T circ() const
    {
        T sum=0;
        for (size_t i=0;i<p.size();i++) sum+=p[i].dis(p[nxt(i)]);
        return sum;
    }
};

//凸多边形
struct Convex: Polygon
{
    // 闵可夫斯基和
    Convex operator+(const Convex &c) const
    {
        const auto &p=this->p;
        vector<Segment> e1(p.size()),e2(c.p.size()),edge(p.size()+c.p.size());
        vector<Point> res; res.reserve(p.size()+c.p.size());
        const auto cmp=[](const Segment &u,const Segment &v) {return Argcmp()(u.b-u.a,v.b-v.a);};
        for (size_t i=0;i<p.size();i++) e1[i]={p[i],p[this->nxt(i)]};
        for (size_t i=0;i<c.p.size();i++) e2[i]={c.p[i],c.p[c.nxt(i)]};
```

```
    rotate(e1.begin(),min_element(e1.begin(),e1.end(),cmp),e1.end());
    rotate(e2.begin(),min_element(e2.begin(),e2.end(),cmp),e2.end());
    merge(e1.begin(),e1.end(),e2.begin(),e2.end(),edge.begin(),cmp);
    const auto check=[](const vector<Point> &res,const Point &u)
    {
        const auto back1=res.back(),back2=*prev(res.end(),2);
        return (back1-back2).toleft(u-back1)==0 && (back1-back2)*(u-back1)>=-eps;
    };
    auto u=e1[0].a+e2[0].a;
    for (const auto &v:edge)
    {
        while (res.size()>1 && check(res,u)) res.pop_back();
        res.push_back(u);
        u=u+v.b-v.a;
    }
    if (res.size()>1 && check(res,res[0])) res.pop_back();
    return {res};
}

// 旋转卡壳
// 例：凸多边形的直径的平方
T rotcaliper() const
{
    const auto &p=this->p;
    if (p.size()==1) return 0;
    if (p.size()==2) return p[0].dis2(p[1]);
    const auto area=[](const Point &u,const Point &v,const Point &w){return (w-u)^(w-v);};
    T ans=0;
    for (size_t i=0,j=1;i<p.size();i++)
    {
        const auto nxti=this->nxt(i);
        ans=max({ans,p[j].dis2(p[i]),p[j].dis2(p[nxti])});
        while (area(p[this->nxt(j)],p[i],p[nxti])>=area(p[j],p[i],p[nxti]))
        {
            j=this->nxt(j);
            ans=max({ans,p[j].dis2(p[i]),p[j].dis2(p[nxti])});
        }
    }
    return ans;
}

// 判断点是否在凸多边形内
// 复杂度 O(logn)
// -1 点在多边形边上 | 0 点在多边形外 | 1 点在多边形内
int is_in(const Point &a) const
{
```

```
    const auto &p=this->p;
    if (p.size()==1) return a==p[0]?-1:0;
    if (p.size()==2) return Segment{p[0],p[1]}.is_on(a)?-1:0;
    if (a==p[0]) return -1;
    if ((p[1]-p[0]).toleft(a-p[0])==-1 || (p.back()-p[0]).toleft(a-p[0])==1) return 0;
    const auto cmp=[&](const Point &u,const Point &v){return (u-p[0]).toleft(v-p[0])==1;};
    const size_t i=lower_bound(p.begin()+1,p.end(),a,cmp)-p.begin();
    if (i==1) return Segment{p[0],p[i]}.is_on(a)?-1:0;
    if (i==p.size()-1 && Segment{p[0],p[i]}.is_on(a)) return -1;
    if (Segment{p[i-1],p[i]}.is_on(a)) return -1;
    return (p[i]-p[i-1]).toleft(a-p[i-1])>0;
}

// 凸多边形关于某一方向的极点
// 复杂度 O(logn)
// 参考资料：https://codeforces.com/blog/entry/48868
template<typename F> size_t extreme(const F &dir) const
{
    const auto &p=this->p;
    const auto check=[&](const size_t i){return dir(p[i]).toleft(p[this->nxt(i)]-p[i])>=0;};
    const auto dir0=dir(p[0]); const auto check0=check(0);
    if (!check0 && check(p.size()-1)) return 0;
    const auto cmp=[&](const Point &v)
    {
        const size_t vi=&v-p.data();
        if (vi==0) return 1;
        const auto checkv=check(vi);
        const auto t=dir0.toleft(v-p[0]);
        if (vi==1 && checkv==check0 && t==0) return 1;
        return checkv^(checkv==check0 && t<=0);
    };
    return partition_point(p.begin(),p.end(),cmp)-p.begin();
}

// 过凸多边形外一点求凸多边形的切线，返回切点下标
// 复杂度 O(logn)
// 必须保证点在多边形外
pair<size_t,size_t> tangent(const Point &a) const
{
    const size_t i=extreme([&](const Point &u){return u-a;});
    const size_t j=extreme([&](const Point &u){return a-u;});
    return {i,j};
}

// 求平行于给定直线的凸多边形的切线，返回切点下标
// 复杂度 O(logn)
```

```
    pair<size_t,size_t> tangent(const Line &a) const
    {
        const size_t i=extreme([&](...){return a.v;});
        const size_t j=extreme([&](...){return -a.v;});
        return {i,j};
    }
};

// 圆
struct Circle
{
    Point c;
    T r; // 一般来说必须用浮点数

    bool operator==(const Circle &a) const {return c==a.c && abs(r-a.r)<=eps;}
    T circ() const {return 2*PI*r;}  // 周长
    T area() const {return PI*r*r;}  // 面积

    // 点与圆的关系
    // -1 圆上 | 0 圆外 | 1 圆内
    int is_in(const Point &p) const {const T d=p.dis(c); return abs(d-r)<=eps?-1:d<r-eps;}

    // 直线与圆关系
    // 0 相离 | 1 相切 | 2 相交
    int relation(const Line &l) const
    {
        const T d=l.dis(c);
        if (d>r+eps) return 0;
        if (abs(d-r)<=eps) return 1;
        return 2;
    }

    // 圆与圆关系
    // -1 相同 | 0 相离 | 1 外切 | 2 相交 | 3 内切 | 4 内含
    int relation(const Circle &a) const
    {
        if (*this==a) return -1;
        const T d=c.dis(a.c);
        if (d>r+a.r+eps) return 0;
        if (abs(d-r-a.r)<=eps) return 1;
        if (abs(d-abs(r-a.r))<=eps) return 3;
        if (d<abs(r-a.r)-eps) return 4;
        return 2;
    }

    // 直线与圆的交点
```

```
vector<Point> inter(const Line &l) const
{
    const T d=l.dis(c);
    const Point p=l.proj(c);
    const int t=relation(l);
    if (t==0) return vector<Point>();
    if (t==1) return vector<Point>{p};
    const T k=sqrt(r*r-d*d);
    return vector<Point>{p-(l.v/l.v.len())*k,p+(l.v/l.v.len())*k};
}

// 圆与圆交点
vector<Point> inter(const Circle &a) const
{
    const T d=c.dis(a.c);
    const int t=relation(a);
    if (t==-1 || t==0 || t==4) return vector<Point>();
    Point e=a.c-c; e=e/e.len()*r;
    if (t==1 || t==3)
    {
        if (r*r+d*d-a.r*a.r>=-eps) return vector<Point>{c+e};
        return vector<Point>{c-e};
    }
    const T costh=(r*r+d*d-a.r*a.r)/(2*r*d),sinth=sqrt(1-costh*costh);
    return vector<Point>{c+e.rot(costh,-sinth),c+e.rot(costh,sinth)};
}

// 圆与圆交面积
T inter_area(const Circle &a) const
{
    const T d=c.dis(a.c);
    const int t=relation(a);
    if (t==-1) return area();
    if (t<2) return 0;
    if (t>2) return min(area(),a.area());
    const T costh1=(r*r+d*d-a.r*a.r)/(2*r*d),costh2=(a.r*a.r+d*d-r*r)/(2*a.r*d);
    const T sinth1=sqrt(1-costh1*costh1),sinth2=sqrt(1-costh2*costh2);
    const T th1=acos(costh1),th2=acos(costh2);
    return r*r*(th1-costh1*sinth1)+a.r*a.r*(th2-costh2*sinth2);
}

// 过圆外一点圆的切线
vector<Line> tangent(const Point &a) const
{
    const int t=is_in(a);
    if (t==1) return vector<Line>();
```

```
    if (t==-1)
    {
        const Point v={-(a-c).y,(a-c).x};
        return vector<Line>{{a,v}};
    }
    Point e=a-c; e=e/e.len()*r;
    const T costh=r/c.dis(a),sinth=sqrt(1-costh*costh);
    const Point t1=c+e.rot(costh,-sinth),t2=c+e.rot(costh,sinth);
    return vector<Line>{{a,t1-a},{a,t2-a}};
}

// 两圆的公切线
vector<Line> tangent(const Circle &a) const
{
    const int t=relation(a);
    vector<Line> lines;
    if (t==-1 || t==4) return lines;
    if (t==1 || t==3)
    {
        const Point p=inter(a)[0],v={-(a.c-c).y,(a.c-c).x};
        lines.push_back({p,v});
    }
    const T d=c.dis(a.c);
    const Point e=(a.c-c)/(a.c-c).len();
    if (t<=2)
    {
        const T costh=(r-a.r)/d,sinth=sqrt(1-costh*costh);
        const Point d1=e.rot(costh,-sinth),d2=e.rot(costh,sinth);
        const Point u1=c+d1*r,u2=c+d2*r,v1=a.c+d1*a.r,v2=a.c+d2*a.r;
        lines.push_back({u1,v1-u1}); lines.push_back({u2,v2-u2});
    }
    if (t==0)
    {
        const T costh=(r+a.r)/d,sinth=sqrt(1-costh*costh);
        const Point d1=e.rot(costh,-sinth),d2=e.rot(costh,sinth);
        const Point u1=c+d1*r,u2=c+d2*r,v1=a.c-d1*a.r,v2=a.c-d2*a.r;
        lines.push_back({u1,v1-u1}); lines.push_back({u2,v2-u2});
    }
    return lines;
}

// 圆的反演
// auto result = circle.inverse(line);
// if (std::holds_alternative<Circle>(result))
// Circle c = std::get<Circle>(result);
std::variant<Circle, Line> inverse(const Line &l) const
```

```
    {
        if (l.toleft(c)==0) return l;
        const Point v=l.toleft(c)==1?Point{l.v.y,-l.v.x}:Point{-l.v.y,l.v.x};
        const T d=r*r/l.dis(c);
        const Point p=c+v/v.len()*d;
        return Circle{(c+p)/2,d/2};
    }

    std::variant<Circle, Line> inverse(const Circle &a) const
    {
        const Point v=a.c-c;
        if (a.is_in(c)==-1)
        {
            const T d=r*r/(a.r+a.r);
            const Point p=c+v/v.len()*d;
            return Line{p,{-v.y,v.x}};
        }
        if (c==a.c) return Circle{c,r*r/a.r};
        const T d1=r*r/(c.dis(a.c)-a.r),d2=r*r/(c.dis(a.c)+a.r);
        const Point p=c+v/v.len()*d1,q=c+v/v.len()*d2;
        return Circle{(p+q)/2,p.dis(q)/2};
    }
};

// 圆与多边形面积交
T area_inter(const Circle &circ,const Polygon &poly)
{
    const auto cal=[](const Circle &circ,const Point &a,const Point &b)
    {
        if ((a-circ.c).toleft(b-circ.c)==0) return 0.0l;
        const auto ina=circ.is_in(a),inb=circ.is_in(b);
        const Line ab={a,b-a};
        if (ina && inb) return ((a-circ.c)^(b-circ.c))/2;
        if (ina && !inb)
        {
            const auto t=circ.inter(ab);
            const Point p=t.size()==1?t[0]:t[1];
            const T ans=((a-circ.c)^(p-circ.c))/2;
            const T th=(p-circ.c).ang(b-circ.c);
            const T d=circ.r*circ.r*th/2;
            if ((a-circ.c).toleft(b-circ.c)==1) return ans+d;
            return ans-d;
        }
        if (!ina && inb)
        {
            const Point p=circ.inter(ab)[0];
```

```
            const T ans=((p-circ.c)^(b-circ.c))/2;
            const T th=(a-circ.c).ang(p-circ.c);
            const T d=circ.r*circ.r*th/2;
            if ((a-circ.c).toleft(b-circ.c)==1) return ans+d;
            return ans-d;
        }
        const auto p=circ.inter(ab);
        if (p.size()==2 && Segment{a,b}.dis(circ.c)<=circ.r+eps)
        {
            const T ans=((p[0]-circ.c)^(p[1]-circ.c))/2;
            const T th1=(a-circ.c).ang(p[0]-circ.c),th2=(b-circ.c).ang(p[1]-circ.c);
            const T d1=circ.r*circ.r*th1/2,d2=circ.r*circ.r*th2/2;
            if ((a-circ.c).toleft(b-circ.c)==1) return ans+d1+d2;
            return ans-d1-d2;
        }
        const T th=(a-circ.c).ang(b-circ.c);
        if ((a-circ.c).toleft(b-circ.c)==1) return circ.r*circ.r*th/2;
        return -circ.r*circ.r*th/2;
    };

    T ans=0;
    for (size_t i=0;i<poly.p.size();i++)
    {
        const Point a=poly.p[i],b=poly.p[poly.nxt(i)];
        ans+=cal(circ,a,b);
    }
    return ans;
}

// 点集的凸包
// Andrew 算法，复杂度 O(nlogn)
Convex convexhull(vector<Point> p)
{
    vector<Point> st;
    if (p.empty()) return Convex{st};
    sort(p.begin(),p.end());
    const auto check=[](const vector<Point> &st,const Point &u)
    {
        const auto back1=st.back(),back2=*prev(st.end(),2);
        return (back1-back2).toleft(u-back1)<=0;
    };
    for (const Point &u:p)
    {
        while (st.size()>1 && check(st,u)) st.pop_back();
        st.push_back(u);
    }
```

```
    size_t k=st.size();
    p.pop_back(); reverse(p.begin(),p.end());
    for (const Point &u:p)
    {
        while (st.size()>k && check(st,u)) st.pop_back();
        st.push_back(u);
    }
    st.pop_back();
    return Convex{st};
}

// 半平面交
// 排序增量法，复杂度 O(nlogn)
// 输入与返回值都是用直线表示的半平面集合
vector<Line> halfinter(vector<Line> l, const T lim=1e9)
{
    const auto check=[](const Line &a,const Line &b,const Line &c){return a.toleft(b.inter(c))<0;};
    // 无精度误差的方法，但注意取值范围会扩大到三次方
    /*const auto check=[](const Line &a,const Line &b,const Line &c)
    {
        const Point p=a.v*(b.v^c.v),q=b.p*(b.v^c.v)+b.v*(c.v^(b.p-c.p))-a.p*(b.v^c.v);
        return p.toleft(q)<0;
    };*/
    l.push_back({{-lim,0},{0,-1}}); l.push_back({{0,-lim},{1,0}});
    l.push_back({{lim,0},{0,1}}); l.push_back({{0,lim},{-1,0}});
    sort(l.begin(),l.end());
    deque<Line> q;
    for (size_t i=0;i<l.size();i++)
    {
        if (i>0 && l[i-1].v.toleft(l[i].v)==0 && l[i-1].v*l[i].v>eps) continue;
        while (q.size()>1 && check(l[i],q.back(),q[q.size()-2])) q.pop_back();
        while (q.size()>1 && check(l[i],q[0],q[1])) q.pop_front();
        if (!q.empty() && q.back().v.toleft(l[i].v)<=0) return vector<Line>();
        q.push_back(l[i]);
    }
    while (q.size()>1 && check(q[0],q.back(),q[q.size()-2])) q.pop_back();
    while (q.size()>1 && check(q.back(),q[0],q[1])) q.pop_front();
    return vector<Line>(q.begin(),q.end());
}

// 点集形成的最小最大三角形
// 极角序扫描线，复杂度 O(n^2logn)
// 最大三角形问题可以使用凸包与旋转卡壳做到 O(n^2)
pair<T,T> minmax_triangle(const vector<Point> &vec)
{
    if (vec.size()<=2) return {0,0};
```

```
    vector<pair<int,int>> evt;
    evt.reserve(vec.size()*vec.size());
    T maxans=0,minans=INF;
    for (size_t i=0;i<vec.size();i++)
    {
        for (size_t j=0;j<vec.size();j++)
        {
            if (i==j) continue;
            if (vec[i]==vec[j]) minans=0;
            else evt.push_back({i,j});
        }
    }
    sort(evt.begin(),evt.end(),[&](const pair<int,int> &u,const pair<int,int> &v)
    {
        const Point du=vec[u.second]-vec[u.first],dv=vec[v.second]-vec[v.first];
        return Argcmp()({du.y,-du.x},{dv.y,-dv.x});
    });
    vector<size_t> vx(vec.size()),pos(vec.size());
    for (size_t i=0;i<vec.size();i++) vx[i]=i;
    sort(vx.begin(),vx.end(),[&](int x,int y){return vec[x]<vec[y];});
    for (size_t i=0;i<vx.size();i++) pos[vx[i]]=i;
    for (auto [u,v]:evt)
    {
        const size_t i=pos[u],j=pos[v];
        const size_t l=min(i,j),r=max(i,j);
        const Point vecu=vec[u],vecv=vec[v];
        if (l>0) minans=min(minans,abs((vec[vx[l-1]]-vecu)^(vec[vx[l-1]]-vecv)));
        if (r<vx.size()-1) minans=min(minans,abs((vec[vx[r+1]]-vecu)^(vec[vx[r+1]]-vecv)));
        maxans=max({maxans,abs((vec[vx[0]]-vecu)^(vec[vx[0]]-vecv)),abs((vec[vx.back()]-vecu)^(vec[vx.back()
]-vecv))});
        if (i<j) swap(vx[i],vx[j]),pos[u]=j,pos[v]=i;
    }
    return {minans,maxans};
}

// 平面最近点对
// 扫描线，复杂度 O(nlogn)
T closest_pair(vector<Point> points)
{
    sort(points.begin(),points.end());
    const auto cmpy=[](const Point &a,const Point &b){if (abs(a.y-b.y)<=eps) return a.x<b.x-eps; return
a.y<b.y-eps;};
    multiset<Point,decltype(cmpy)> s{cmpy};
    T ans=INF;
    for (size_t i=0,l=0;i<points.size();i++)
    {
```

```
        const T sqans=sqrtl(ans)+1;
        while (l<i && points[i].x-points[l].x>=sqans) s.erase(s.find(points[l++]));
        for (auto
it=s.lower_bound(Point{-INF,points[i].y-sqans});it!=s.end()&&it->y-points[i].y<=sqans;it++)
        {
            ans=min(ans,points[i].dis2(*it));
        }
        s.insert(points[i]);
    }
    return ans;
}

// 判断多条线段是否有交点
// 扫描线，复杂度 O(nlogn)
bool segs_inter(const vector<Segment> &segs)
{
    if (segs.empty()) return false;
    using seq_t=tuple<T,int,Segment>;  // x 坐标 出入点 线段
    const auto seqcmp=[](const seq_t &u, const seq_t &v)
    {
        const auto [u0,u1,u2]=u;
        const auto [v0,v1,v2]=v;
        if (abs(u0-v0)<=eps) return make_pair(u1,u2)<make_pair(v1,v2);
        return u0<v0-eps;
    };
    vector<seq_t> seq;
    for (auto seg:segs)
    {
        if (seg.a.x>seg.b.x+eps) swap(seg.a,seg.b);
        seq.push_back({seg.a.x,0,seg});
        seq.push_back({seg.b.x,1,seg});
    }
    sort(seq.begin(),seq.end(),seqcmp);
    T x_now;
    auto cmp=[&](const Segment &u, const Segment &v)
    {
        if (abs(u.a.x-u.b.x)<=eps || abs(v.a.x-v.b.x)<=eps) return u.a.y<v.a.y-eps;
        return
((x_now-u.a.x)*(u.b.y-u.a.y)+u.a.y*(u.b.x-u.a.x))*(v.b.x-v.a.x)<((x_now-v.a.x)*(v.b.y-v.a.y)+v.a.y*(v.b.x-
v.a.x))*(u.b.x-u.a.x)-eps;
    };
    multiset<Segment,decltype(cmp)> s{cmp};
    for (const auto [x,o,seg]:seq)
    {
        x_now=x;
        const auto it=s.lower_bound(seg);
```

```
        if (o==0)
        {
            if (it!=s.end() && seg.is_inter(*it)) return true;
            if (it!=s.begin() && seg.is_inter(*prev(it))) return true;
            s.insert(seg);
        }
        else
        {
            if (next(it)!=s.end() && it!=s.begin() && (*prev(it)).is_inter(*next(it))) return true;
            s.erase(it);
        }
    }
    return false;
}

// 多边形面积并
// 轮廓积分，复杂度 O(n^2logn)，n 为边数
// ans[i] 表示被至少覆盖了 i+1 次的区域的面积
vector<T> area_union(const vector<Polygon> &polys)
{
    const size_t siz=polys.size();
    vector<vector<pair<Point,Point>>> segs(siz);
    const auto check=[](const Point &u,const Segment &e){return !((u<e.a && u<e.b) || (u>e.a && u>e.b));};

    auto cut_edge=[&](const Segment &e,const size_t i)
    {
        const Line le{e.a,e.b-e.a};
        vector<pair<Point,int>> evt;
        evt.push_back({e.a,0}); evt.push_back({e.b,0});
        for (size_t j=0;j<polys.size();j++)
        {
            if (i==j) continue;
            const auto &pj=polys[j];
            for (size_t k=0;k<pj.p.size();k++)
            {
                const Segment s={pj.p[k],pj.p[pj.nxt(k)]};
                if (le.toleft(s.a)==0 && le.toleft(s.b)==0)
                {
                    evt.push_back({s.a,0});
                    evt.push_back({s.b,0});
                }
                else if (s.is_inter(le))
                {
                    const Line ls{s.a,s.b-s.a};
                    const Point u=le.inter(ls);
                    if (le.toleft(s.a)<0 && le.toleft(s.b)>=0) evt.push_back({u,-1});
```

```
                else if (le.toleft(s.a)>=0 && le.toleft(s.b)<0) evt.push_back({u,1});
            }
        }
        sort(evt.begin(),evt.end());
        if (e.a>e.b) reverse(evt.begin(),evt.end());
        int sum=0;
        for (size_t i=0;i<evt.size();i++)
        {
            sum+=evt[i].second;
            const Point u=evt[i].first,v=evt[i+1].first;
            if (!(u==v) && check(u,e) && check(v,e)) segs[sum].push_back({u,v});
            if (v==e.b) break;
        }
    };

    for (size_t i=0;i<polys.size();i++)
    {
        const auto &pi=polys[i];
        for (size_t k=0;k<pi.p.size();k++)
        {
            const Segment ei={pi.p[k],pi.p[pi.nxt(k)]};
            cut_edge(ei,i);
        }
    }
    vector<T> ans(siz);
    for (size_t i=0;i<siz;i++)
    {
        T sum=0;
        sort(segs[i].begin(),segs[i].end());
        int cnt=0;
        for (size_t j=0;j<segs[i].size();j++)
        {
            if (j>0 && segs[i][j]==segs[i][j-1]) segs[i+(++cnt)].push_back(segs[i][j]);
            else cnt=0,sum+=segs[i][j].first^segs[i][j].second;
        }
        ans[i]=sum/2;
    }
    return ans;
}

// 圆面积并
// 轮廓积分，复杂度 O(n^2logn)
// ans[i] 表示被至少覆盖了 i+1 次的区域的面积
vector<T> area_union(const vector<Circle> &circs)
{
```

```
const size_t siz=circs.size();
using arc_t=tuple<Point,T,T,T>;
vector<vector<arc_t>> arcs(siz);
const auto eq=[](const arc_t &u,const arc_t &v)
{
    const auto [u1,u2,u3,u4]=u;
    const auto [v1,v2,v3,v4]=v;
    return u1==v1 && abs(u2-v2)<=eps && abs(u3-v3)<=eps && abs(u4-v4)<=eps;
};

auto cut_circ=[&](const Circle &ci,const size_t i)
{
    vector<pair<T,int>> evt;
    evt.push_back({-PI,0}); evt.push_back({PI,0});
    int init=0;
    for (size_t j=0;j<circs.size();j++)
    {
        if (i==j) continue;
        const Circle &cj=circs[j];
        if (ci.r<cj.r-eps && ci.relation(cj)>=3) init++;
        const auto inters=ci.inter(cj);
        if (inters.size()==1) evt.push_back({atan2l((inters[0]-ci.c).y,(inters[0]-ci.c).x),0});
        if (inters.size()==2)
        {
            const Point dl=inters[0]-ci.c,dr=inters[1]-ci.c;
            T argl=atan2l(dl.y,dl.x),argr=atan2l(dr.y,dr.x);
            if (abs(argl+PI)<=eps) argl=PI;
            if (abs(argr+PI)<=eps) argr=PI;
            if (argl>argr+eps)
            {
                evt.push_back({argl,1}); evt.push_back({PI,-1});
                evt.push_back({-PI,1}); evt.push_back({argr,-1});
            }
            else
            {
                evt.push_back({argl,1});
                evt.push_back({argr,-1});
            }
        }
    }
    sort(evt.begin(),evt.end());
    int sum=init;
    for (size_t i=0;i<evt.size();i++)
    {
        sum+=evt[i].second;
```

```
            if (abs(evt[i].first-evt[i+1].first)>eps)
arcs[sum].push_back({ci.c,ci.r,evt[i].first,evt[i+1].first});
            if (abs(evt[i+1].first-PI)<=eps) break;
        }
    };

    const auto oint=[](const arc_t &arc)
    {
        const auto [cc,cr,l,r]=arc;
        if (abs(r-l-PI-PI)<=eps) return 2.0l*PI*cr*cr;
        return cr*cr*(r-l)+cc.x*cr*(sin(r)-sin(l))-cc.y*cr*(cos(r)-cos(l));
    };

    for (size_t i=0;i<circs.size();i++)
    {
        const auto &ci=circs[i];
        cut_circ(ci,i);
    }
    vector<T> ans(siz);
    for (size_t i=0;i<siz;i++)
    {
        T sum=0;
        sort(arcs[i].begin(),arcs[i].end());
        int cnt=0;
        for (size_t j=0;j<arcs[i].size();j++)
        {
            if (j>0 && eq(arcs[i][j],arcs[i][j-1])) arcs[i+(++cnt)].push_back(arcs[i][j]);
            else cnt=0,sum+=oint(arcs[i][j]);
        }
        ans[i]=sum/2;
    }
    return ans;
}
```