

## Contents

## 1 DP

1.1	divide-and-conquer-optimization	1
1.2	knuth-optimization	1
1.3	li-chao-tree	1
1.4	zero-matrix	1

## 2 DS

2.1	BIT	1
2.2	Heavy-Light-Decomposition	1
2.3	LCA	2
2.4	Lazy Propagation	2
2.5	MO with update	2
2.6	bipartite-disjoint-set-union	2
2.7	centroid decomposition	2
2.8	dsu-rollback	3
2.9	fenwick-tree-2d	3
2.10	link cut tree	3
2.11	monotone <sub>ht</sub>	4
2.12	segment tree beats	4
2.13	sparse table 2d	5
2.14	treap	5

## 3 Extra

3.1	Header	6
-----	--------	---

## 4 Game

4.1	HackenBush	6
-----	------------	---

## 5 Geo

5.1	3dGeo	6
5.2	Circle Cover	7
5.3	Circle Union Area	7
5.4	basic-area-geometry	7
5.5	geo-formulae	8
5.6	half-plane-intersection	8
5.7	heart-of-geometry-2d	8
5.8	intersecting-segments-pair	10
5.9	radiant-geo	11
5.10	trianlge-ear-clipping	13
5.11	vertical-decomposition	13

## 6 Graph

6.1	DMST	14
6.2	Flow With Demands	15
6.3	LCA	15
6.4	articulation-vertex	15
6.5	bellman-ford	15
6.6	bridge	15
6.7	edmond-blossom	16
6.8	euler-path	16
6.9	hopcraft-karp	16

6.10	hungarian-algorithm	17
6.11	max-flow-dinic	18
6.12	min-cost-max-flow	18
6.13	online-bridge	18
6.14	scc + 2 Sat	19

## 7 Math

7.1	BerleKampMassey	19
7.2	FloorSum	19
7.3	Stern Brocot Tree	19
7.4	Sum Of Kth Power	19
7.5	combination-generator	19
7.6	continued-fractions	20
7.7	convolution	20
7.8	crt anachor	20
7.9	discrete-root	20
7.10	fast-walsh-hadamard	20
7.11	fft	21
7.12	formulas	21
7.13	integer-factorization	21
7.14	integration-simpson	21
7.15	linear-diophantine-equation-gray-code	21
7.16	linear-equation-system	22
7.17	math	22
7.18	nCr mod $p^a$	22
7.19	ntt	22
7.20	primality-test	23
7.21	prime counting function	23

## 8 String

8.1	Hashing	23
8.2	KMP	23
8.3	aho-corasick	23
8.4	manacher	23
8.5	palindromic tree	23
8.6	suffix array da	24
8.7	suffix-automaton	24
8.8	z-algorithm	25

## 1 DP

## 1.1 divide-and-conquer-optimization

```

int m, n;
vector<long long> dp_before(n), dp_cur(n);
long long C(int i, int j);
// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr){
    if (l > r)
        return;
    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};
    for (int k = optl; k <= min(mid, optr); k++){
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k,
            mid), k});
    }
    dp_cur[mid] = best.first;
    int opt = best.second;
    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

```

```

}
int solve(){
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);
    for (int i = 1; i < m; i++){
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }
    return dp_before[n - 1];
}

```

## 1.2 knuth-optimization

```

int solve() {
    int N;
    ... // read N and input
    int dp[N][N], opt[N][N];
    auto C = [&](int i, int j) {
        ... // Implement cost function C.
    };
    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i] according to the problem
    }
    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1,
                opt[i+1][j]); k++) {
                if (mn >= dp[i][k] + dp[k+1][j] + cost) {
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn;
        }
    }
    cout << dp[0][N-1] << endl;
}

```

## 1.3 li-chao-tree

```

typedef long long ll;
class LiChaoTree{
    ll L,R;
    bool minimize;
    int lines;
    struct Node{
        pair<ll,ll> line;
        Node *children[2];
        Node(pair<ll,ll> ln= {0,1000000000000000000}){
            line=ln;
            children[0]=0;
            children[1]=0;
        }
    } *root;
    ll f(pair<ll,ll> a, ll x){
        return a.first*x+a.second;
    }
    void clear(Node* &node){
        if(node->children[0]){
            clear(node->children[0]);
        }
        if(node->children[1]){
            clear(node->children[1]);
        }
        delete node;
    }
    void add_line(pair<ll,ll> nw, Node* &node, ll l, ll r){
        if(node==0){
            node=new Node(nw);
            return;
        }
        ll m = (l + r) / 2;
        bool lef = (f(nw, l) < f(node->line,
            l))&&minimize||(!minimize)&&f(nw, l) >
            f(node->line, l));
        bool mid = (f(nw, m) < f(node->line,
            m))&&minimize||(!minimize)&&f(nw, m) >

```

```

        f(node->line, m));
    if(mid){
        swap(node->line, nw);
    }
    if(r - 1 == 1){
        return;
    }
    else if(lef != mid){
        add_line(nw, node->children[0], 1, m);
    }
    else{
        add_line(nw, node->children[1], m, r);
    }
}
ll get(ll x, Node* &node, ll l, ll r){
    ll m = (l + r) / 2;
    if(r - l == 1){
        return f(node->line, x);
    }
    else if(x < m){
        if(node->children[0]==0) return f(node->line, x);
        if(minimize) return min(f(node->line, x), get(x,
            node->children[0], 1, m));
        else return max(f(node->line, x), get(x,
            node->children[0], 1, m));
    }
    else{
        if(node->children[1]==0) return f(node->line, x);
        if(minimize) return min(f(node->line, x), get(x,
            node->children[1], m, r));
        else return max(f(node->line, x), get(x,
            node->children[1], m, r));
    }
}
public:
LiChaoTree(ll l=-1000000001, ll r=1000000001, bool mn=false){
    L=l;
    R=r;
    root=0;
    minimize=mn;
    lines=0;
}
void AddLine(pair<ll,ll> ln){
    add_line({ln.first,ln.second},root,L,R);
    lines++;
}
int number_of_lines(){
    return lines;
}
ll getOptimumValue(ll x){
    return get(x,root,L,R);
}
LiChaoTree(){
    if(root!=0) clear(root);
}
};

```

#### 1.4 zero-matrix

```

int zero_matrix(vector<vector<int>> a) {
    int n = a.size();
    int m = a[0].size();
    int ans = 0;
    vector<int> d(m, -1), d1(m), d2(m);
    stack<int> st;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (a[i][j] == 1)
                d[j] = i;
        }
        for (int j = 0; j < m; ++j) {
            while (!st.empty() && d[st.top()] <= d[j])
                st.pop();
            d1[j] = st.empty() ? -1 : st.top();
            st.push(j);
        }
        while (!st.empty())
            st.pop();
        for (int j = m - 1; j >= 0; --j) {
            while (!st.empty() && d[st.top()] <= d[j])

```

```

                st.pop();
                d2[j] = st.empty() ? m : st.top();
                st.push(j);
            }
            while (!st.empty())
                st.pop();
            for (int j = 0; j < m; ++j)
                ans = max(ans, (i - d1[j]) * (d2[j] - d1[j] - 1));
            return ans;
        }
    }
}

```

## 2 DS

### 2.1 BIT

```

#include<bits/stdc++.h>
using namespace std;
const int MaxIdx=1e+5;
int tree[MaxIdx+1];
int read(int idx) {int sum = 0;
    while (idx > 0) {sum += tree[idx];idx -= (idx & -idx);}
    return sum;}
void update(int idx, int val) {while (idx <= MaxIdx) {
    tree[idx] += val;idx += (idx & -idx);}}
int readSingle(int idx) {
    int sum = tree[idx];
    if (idx > 0) {int z = idx - (idx & -idx);idx--;
        while (idx != z) {sum += tree[idx];idx -= (idx & -idx);}}
    return sum;}
int findG(int cumFre) {int idx = 0;int bitMask=(1<<16);
    while (bitMask != 0) {int tIdx = idx + bitMask;bitMask >>=
        1;
        if (tIdx > MaxIdx)continue;
        if (cumFre >= tree[tIdx]) {idx = tIdx;cumFre -=
            tree[tIdx];}}
    if (cumFre != 0)return -1;else return idx;}

```

### 2.2 Heavy-Light-Decomposition

```

namespace HLD{
    struct Node{
        int mn=INT_MAX,unp=INT_MAX;
        Node(){}
        Node(int mn,int unp): mn(mn),unp(unp){};
        inline Node combine(Node a,Node b){
            return {min(a.mn,b.mn),INT_MAX};}
        inline Node propagate(Node to,Node from,int len){
            if(from.unp==INT_MAX)
                return to;
            to.mn=min(to.mn,from.unp);
            to.unp=min(to.unp,from.unp);
            return to;}
        #define MAX_SIZE 100001
        vector<vector<int>> >G;
        vector<int> parent,depth, heavy,head,pos;
        vector<int> node_val;
        vector<int> node_val_order;
        SegTree<Node> sgt( MAX_SIZE,combine,propagate);
        int cur_pos;
        int dfs(int node){
            int sz=1;
            int max_c_size=0;
            for(auto c:G[node]){
                if(c!=parent[node]){
                    parent[c]=node;
                    depth[c]=depth[node]+1;
                    int c_size=dfs(c);
                    sz+=c_size;
                    if(c_size>max_c_size){
                        max_c_size=c_size;
                        heavy[node]=c; }}}
            return sz;}
        void decompose(int node,int h){
            pos[node]=cur_pos++;
            head[node]=h;
            if(heavy[node]!=-1)
                decompose(heavy[node],h);
            for(int c:G[node]){
                if(c!=parent[node]&& c!=heavy[node])

```

```

                decompose(c,c);}
            return;}
        //for query on path the node_val of a node is the cost of
        //the edge to parent
        //exclude=true for query on path,it excludes the value
        //stored in lca(a,b)
        int query(int a,int b,int n,bool exclude=false){//n number
        //of node in the tree
        Node res;//not really generalized,for min max update
        accordingly
        while (head[a]!=head[b]){
            if(depth[head[a]]> depth[head[b]])swap(a,b);
            res=combine(res, sgt.query(pos[head[b]],pos[b]));
            b=parent[head[b]];}
        if(depth[a]>depth[b])swap(a,b);
        res=combine(res,exclude? sgt.query(pos[a]+1,pos[b])
            :sgt.query(pos[a],pos[b]));
        return res.mn;}
        // void update(int node,int val)
        // {
        //     sgt.update(pos[node],val);
        // }
        void update(int a,int b,int val){
            while(head[a]!=head[b]){
                if(depth[head[a]]> depth[head[b]])
                    swap(a,b);
                //res=combine(res,sgt.query(pos[head[b]],pos[b]));
                sgt.update(pos[head[b]], pos[b],{0,val});
                b=parent[head[b]];
            }
            if(depth[a]>depth[b])swap(a,b);
            //res=combine(res,exclude? sgt.query(pos[a]+1, pos[b]):
            //sgt.query(pos[a],pos[b]));
            sgt.update(pos[a],pos[b],{0,val});
            //return res.mn;}
        void init(int n){
            parent.resize(n);
            depth.resize(n);
            heavy.assign(n,-1);
            head.resize(n);
            pos.resize(n);
            parent[0]=0;//might change later
            dfs(0);
            cur_pos=0;
            decompose(0,0);
            node_val_order.resize(n);
            for(int i=0;i<n;++i){
                node_val_order[pos[i]]= node_val[i];}
            sgt.n=n;
            sgt.build(node_val_order, 1,0,n-1); }
}

```

### 2.3 LCA

```

int lca(int u,int v)
{
    if(depth[v]>depth[u])
        v=pth_ancestor(v,depth[v]-depth[u]);
    if(depth[u]>depth[v])
        u=pth_ancestor(u,depth[u]-depth[v]);
    if(u==v)
        return u;
    for(int i=log2(n-1);i>=0;i--){
        if(bparent[u][i]!=bparent[v][i]){
            u=bparent[u][i];
            v=bparent[v][i];
        }
    }
    return bparent[u][0];}

```

### 2.4 Lazy Propagation

```

struct node{
    int sum, prop;
    node(){sum=0, prop=0;}
} st[500000];
void propagate(int p, int c, int len){
    st[c].prop+=st[p].prop;
    st[c].sum+=len*st[p].prop;}
void combine(int v, int l, int r){
    st[v].prop=0;

```

```

    st[v].sum=st[l].sum+st[r].sum;}
void update(int v, int vl, int vr, int l, int r, int u){
    if(l>r) return;
    if(vl==l and vr==r){
        st[v].sum+=(vr-vl+1)*u;
        st[v].prop+=u;
        return;}
    int mid=(vl+vr)/2;
    propagate(v, 2*v, mid-vl+1);
    propagate(v, 2*v+1, vr-mid);
    if(r<=mid) update(2*v, vl, mid, l, r, u);
    else if(l>mid) update(2*v+1, mid+1, vr, l, r, u);
    else {
        update(2*v, vl, mid, l, mid, u);
        update(2*v+1, mid+1, vr, mid+1, r, u);}
    combine(v, 2*v, 2*v+1);}
int query(int v, int vl, int vr, int l, int r){
    if(l>r) return 0;
    if(vl==l and vr==r) return st[v].sum;
    int mid=(vl+vr)/2;
    propagate(v, 2*v, mid-vl+1);
    propagate(v, 2*v+1, vr-mid);
    int qres;
    if(r<=mid) qres=query(2*v, vl, mid, l, r);
    else if(l>mid) qres=query(2*v+1, mid+1, vr, l, r);
    else qres=query(2*v, vl, mid, l, mid)+query(2*v+1, mid+1,
        vr, mid+1, r);
    combine(v, 2*v, 2*v+1);
    return qres;}

```

## 2.5 MO with update

```

const int N = 1e5 + 5;
const int P = 2000; //block size = (2*n^2)^(1/3)
struct query{
    int t, l, r, k, i;
};
vector<query> q;
vector<array<int, 3>> upd;
vector<int> ans; a;
void add(int x); void rem(int x); int get_answer();
void mos_algorithm(){
    sort(q.begin(), q.end(), [](const query &a, const query &b){
        if (a.t / P != b.t / P) return a.t < b.t;
        if (a.l / P != b.l / P) return a.l < b.l;
        if ((a.l / P) & 1) return a.r < b.r;
        return a.r > b.r;
    });
    for(int i=upd.size()-1; i>=0; --i) a[upd[i][0]] = upd[i][1];
    int L = 0, R = -1, T = 0;
    auto apply = [&](int i, int f1){
        int p = upd[i][0], x = upd[i][f1 + 1];
        if (L <= p && p <= R){ rem(a[p]); add(x); }
        a[p] = x;
    };
    ans.clear(); ans.resize(q.size());
    for (auto qr : q){
        int t = qr.t, l = qr.l, r = qr.r, k = qr.k;
        while (T < t) apply(T++, 1);
        while (T > t) apply(--T, 0);
        while (R < r) add(a[++R]);
        while (L > l) add(a[--L]);
        while (R > r) rem(a[R--]);
        while (L < l) rem(a[L++]);
        ans[qr.i] = get_answer();
    }
}
void TEST_CASES(int cas){
    cin>>n>>m; a.resize(n); for(int i=0; i<n; i++) cin>>a[i];
    for(int i=0; i<m; i++){ int tp; scanf("%d", &tp);
        if (tp == 1){ int l, r, k; cin>>l>>r>>k;
            q.push_back({upd.size(), l - 1, r - 1, k, q.size()}); }
        else{ int p, x; cin>>p>>x; --p;
            upd.push_back({p, a[p], x}); a[p] = x;
        }
    }
    mos_algorithm();
}

```

## 2.6 bipartite-disjoint-set-union

```

void make_set(int v) {
    parent[v] = make_pair(v, 0); rank[v] = 0; bipartite[v] = true;}
pair<int, int> find_set(int v) { if (v != parent[v].first) {
    int parity = parent[v].second; parent[v] = find_set(
    parent[v].first); parent[v].second ^= parity;}
    return parent[v];
}
void add_edge(int a, int b) {
    pair<int, int> pa = find_set(a);
    a = pa.first; int x = pa.second;
    pair<int, int> pb = find_set(b); b = pb.first;
    int y = pb.second;
    if (a == b) {
        if (x == y) bipartite[a] = false;
    } else {
        if (rank[a] < rank[b]) swap (a, b);
        parent[b] = make_pair(a, x^y^1);
        bipartite[a] ^= bipartite[b];
        if (rank[a] == rank[b]) ++rank[a];
    }
}
bool is_bipartite(int v){ return bipartite[find_set(v).first];}

```

## 2.7 centroid decomposition

```

vector<int> tree[sz], ctree[sz];
int sub[sz], vis[sz], cparent[sz];
void dfs(int a, int p)
{
    sub[a]=1;
    //printf("loop\n");
    for(auto v: tree[a]){
        if(v!=p && vis[v]==0){
            dfs(v,a);
            sub[a]+=sub[v];
        }
    }
    //cout<<a<<" "<<sub[a]<<endl;
}
int findCentroid(int a, int p, int num)
{
    //printf("loop\n");
    bool sig=true;
    while(sig){
        sig=false;
        for(auto v: tree[a]){
            if(v!=p && vis[v]==0 && 2*sub[v]>num){
                p=a;
                a=v;
                sig=true;
                break;
            }
        }
    }
    return a;
}
int Decompose(int a, int p, int num)
{
    dfs(a,p);
    int centroid=findCentroid(a,p,num);
    vis[centroid]=1;
    //printf("%d\n",centroid);
    for(auto v: tree[centroid]){
        if(vis[v]==0){
            if(sub[v]>sub[centroid])
                ctree[centroid].pb(Decompose(v,centroid,num-sub[centroid]));
            else
                ctree[centroid].pb(Decompose(v,centroid,sub[v]));
        }
        cparent[centroid].back()=centroid;
    }
    return centroid;
}

```

## 2.8 dsu-rollback

```

struct dsu_save {

```

```

    int v, rnkv, u, rnku; dsu_save() {}
    dsu_save(int _v, int _rnkv, int _u, int _rnku)
        : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) {}
};
struct dsu_with_rollback {
    vector<int> p, rnk; int comps; stack<dsu_save> op;
    dsu_with_rollback() {}
    dsu_with_rollback(int n) { p.resize(n); rnk.resize(n);
        for (int i = 0; i < n; i++) { p[i] = i; rnk[i] = 0; }
    }
    int find_set(int v){return (v == p[v])?v:find_set(p[v]);}
    bool unite(int v, int u) { v = find_set(v); u = find_set(u);
        if (v == u) return false; comps--;
        if (rnk[v] > rnk[u]) swap(v, u);
        op.push(dsu_save(v, rnk[v], u, rnk[u])); p[v] = u;
        if (rnk[u] == rnk[v]) rnk[u]++; return true;
    }
    void rollback() { if (op.empty()) return;
        dsu_save x = op.top(); op.pop(); comps++; p[x.v] = x.v;
        rnk[x.v] = x.rnkv; p[x.u] = x.u; rnk[x.u] = x.rnku;
    }
};
struct query {
    int v, u; bool united;
    query(int _v, int _u) : v(_v), u(_u) {}
};
struct QueryTree {
    vector<vector<query>> t; dsu_with_rollback dsu; int T;
    QueryTree() {}
    QueryTree(int _T, int n) : T(_T) {
        dsu = dsu_with_rollback(n); t.resize(4 * T + 4); }
    void add_to_tree(int v, int l, int r, int ul, int ur, query& q){
        if (ul > ur) return;
        if (l == ul && r == ur) { t[v].push_back(q); return; }
        int mid = (l + r) / 2;
        add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
        add_to_tree(2*v+1, mid+1, r, max(ul, mid + 1), ur, q);
    }
    void add_query(query q, int l, int r) {
        add_to_tree(1, 0, T - 1, l, r, q); }
    void dfs(int v, int l, int r, vector<int>& ans) {
        for (query& q : t[v]) q.united = dsu.unite(q.v, q.u);
        if (l == r) ans[l] = dsu.comps;
        else { int mid = (l + r) / 2;
            dfs(2 * v, l, mid, ans); dfs(2 * v + 1, mid + 1, r, ans); }
        for (query q : t[v]) { if (q.united) dsu.rollback(); }
    }
    vector<int> solve() {
        vector<int> ans(T); dfs(1, 0, T - 1, ans); return ans;
    }
};

```

## 2.9 fenwick-tree-2d

```

struct FenwickTree2D {
    vector<vector<int>> bit;
    int n, m;
    // init(...) { ... }
    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
            for (int j = y; j >= 0; j = (j & (j + 1)) - 1)
                ret += bit[i][j];
        return ret;
    }
    void add(int x, int y, int delta) {
        for (int i = x; i < n; i = i | (i + 1))
            for (int j = y; j < m; j = j | (j + 1))
                bit[i][j] += delta;
    }
};

```

## 2.10 link cut tree

```

const int MOD = 998244353;
int sum(int a, int b) {

```

```

    return a+b >= MOD ? a+b-MOD : a+b;
}
int mul(int a, int b) {
    return (a*1LL*b)%MOD;
}
typedef pair< int , int >Linear;
Linear compose(const Linear &p, const Linear &q) {
    return Linear(mul(p.first, q.first), sum(mul(q.second,
        p.first), p.second));
}
struct SplayTree {
    struct Node {
        int ch[2] = {0, 0}, p = 0;
        long long self = 0, path = 0;    // Path aggregates
        long long sub = 0, vir = 0;    // Subtree aggregates
        bool flip = 0;    // Lazy tags
        int size = 1;
        Linear _self[1, 0], _path_shoja[1, 0], _path_ulta[1, 0];
    };
    vector<Node> T;
    SplayTree(int n) : T(n + 1) {
        T[0].size = 0;
    }
    void push(int x) {
        if (!x || !T[x].flip) return;
        int l = T[x].ch[0], r = T[x].ch[1];
        T[l].flip ^= 1, T[r].flip ^= 1;
        swap(T[x].ch[0], T[x].ch[1]);
        T[x].flip = 0;
        swap(T[x]._path_shoja, T[x]._path_ulta);
    }
    void pull(int x) {
        int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);
        T[x].size = T[l].size + T[r].size + 1;
        T[x].path = T[l].path + T[x].self + T[r].path;
        T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
        T[x]._path_shoja = compose(T[l]._path_shoja,
            compose(T[x]._self, T[l]._path_shoja));
        T[x]._path_ulta = compose(T[l]._path_ulta,
            compose(T[x]._self, T[r]._path_ulta));
    }
    void set(int x, int d, int y) {
        T[x].ch[d] = y; T[y].p = x; pull(x);
    }
    void splay(int x) {
        auto dir = [&](int x) {
            int p = T[x].p; if (!p) return -1;
            return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 :
                -1;
        };
        auto rotate = [&](int x) {
            int y = T[x].p, z = T[y].p, dx = dir(x), dy =
                dir(y);
            set(y, dx, T[x].ch[!dx]);
            set(x, !dx, y);
            if (~dy) set(z, dy, x);
            T[x].p = z;
        };
        for (push(x); ~dir(x); ) {
            int y = T[x].p, z = T[y].p;
            push(z); push(y); push(x);
            int dx = dir(x), dy = dir(y);
            if (~dy) rotate(dx != dy ? x : y);
            rotate(x);
        }
    }
    int KthNext(int x, int k) {
        assert(k > 0);
        splay(x);
        x = T[x].ch[1];
        if (T[x].size < k) return -1;
        while (true) {
            push(x);
            int l = T[x].ch[0], r = T[x].ch[1];
            if (T[l].size+1 == k) return x;
            if (k <= T[l].size) x = l;
            else k -= T[l].size+1, x = r;
        }
    }
}

```

```

}
};
struct LinkCut : SplayTree {
    LinkCut(int n) : SplayTree(n) {}
    int access(int x) {
        int u = x, v = 0;
        for (; u; v = u, u = T[u].p) {
            splay(u);
            int& ov = T[u].ch[1];
            T[u].vir += T[ov].sub;
            T[u].vir -= T[v].sub;
            ov = v; pull(u);
        }
        splay(x);
        return v;
    }
    void reroot(int x) {
        access(x); T[x].flip ^= 1; push(x);
    }
    //makes v parent of u (optional: u must be a root)
    void Link(int u, int v) {
        reroot(u); access(v);
        T[v].vir += T[u].sub;
        T[u].p = v; pull(v);
    }
    //removes edge between u and v
    void Cut(int u, int v) {
        int _u = FindRoot(u);
        reroot(u); access(v);
        T[v].ch[0] = T[u].p = 0; pull(v);
        reroot(_u);
    }
    // Rooted tree LCA. Returns 0 if u and v arent connected.
    int LCA(int u, int v) {
        if (u == v) return u;
        access(u); int ret = access(v);
        return T[u].p ? ret : 0;
    }
    // Query subtree of u where v is outside the subtree.
    long long Subtree(int u, int v) {
        int _v = FindRoot(v);
        reroot(v); access(u);
        long long ans = T[u].vir + T[u].self;
        reroot(_v);
        return ans;
    }
    // Query path [u..v]
    long long Path(int u, int v) {
        int _u = FindRoot(u);
        reroot(u); access(v);
        long long ans = T[v].path;
        reroot(_u);
        return ans;
    }
    Linear _Path(int u, int v) {
        reroot(u); access(v); return T[v]._path_shoja;
    }
    // Update vertex u with value v
    void Update(int u, long long v) {
        access(u); T[u].self = v; pull(u);
    }
    // Update vertex u with value v
    void _Update(int u, Linear v) {
        access(u); T[u]._self = v; pull(u);
    }
    int FindRoot(int u) {
        access(u);
        while (T[u].ch[0]) {
            u = T[u].ch[0];
            push(u);
        }
        access(u);
        return u;
    }
    //k-th node (0-indexed) on the path from u to v
    int KthOnPath(int u, int v, int k) {
        if (u == v) return k == 0 ? u : -1;
        int _u = FindRoot(u);
        reroot(u); access(v);
    }
}

```

```

    int ans = KthNext(u, k);
    reroot(_u);
    return ans;
}
};
int main() {
    cin >> n >> q;
    LinkCut lct(n);
    for (int i = 1; i <= n; i++) {
        Linear l;
        cin >> l.first >> l.second;
        lct._Update(i, l);
    }
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        lct.Link(u+1, v+1);
    }
    while (q--) {
        int op;
        cin >> op;
        if (op == 0) {
            int u, v, w, x;
            cin >> u >> v >> w >> x;
            lct.Cut(u+1, v+1);
            lct.Link(w+1, x+1);
        } else if (op == 1) {
            int p; Linear l;
            cin >> p >> l.first >> l.second;
            lct._Update(p+1, l);
        } else {
            int u, v, x;
            cin >> u >> v >> x;
            Linear l = lct._Path(u+1, v+1);
            cout << sum(mul(l.first, x), l.second) << "\n";
        }
    }
    return 0;
}

```

## 2.11 monotone<sub>cht</sub>

```

/*Fully Generalized implementation for Monotone
slope,Arbitrary query
runtime insert() O(1) query() O(logn) */
class MonotoneCHT{
    deque<Line> Q;
    int type;
    void insertBack(Line nl){
        //handle parallel line insertion
        if(!Q.empty() && Q.back().parallel(nl)){
            if(type<2){
                if(Q.back().c>nl.c)
                    Q.pop_back();
                else
                    return;
            }
            else{
                if(Q.back().c<nl.c)
                    Q.pop_back();
                else
                    return;
            }
        }
        while(Q.size()>1 && Q.back().intersect(nl)<Q[Q.size()-2].intersect(nl))
            Q.pop_back();
        Q.push_back(nl);
    }
    void insertFront(Line nl){
        if(!Q.empty() && Q[0].parallel(nl)){
            if(type<2){
                if(Q[0].c>nl.c)
                    Q.pop_front();
                else
                    return;
            }
            else{
                if(Q[0].c<nl.c)
                    Q.pop_front();
                else
                    return;
            }
        }
        while(Q.size()>1 && Q[0].intersect(nl)>Q[1].intersect(nl))
            Q.pop_front();
        Q.push_front(nl);
    }
    pii bSearch(ll x){
    }
}

```



```

    if(Q.size()==1||Q[0].intersect(Q[1]).first>=x)
        return {0,0};
    int l=1,r=(int)Q.size()-1;
    while(r>l+1){
        int mid=(l+r)/2;
        if(Q[mid].intersect(Q[mid-1]).first<x)
            l=mid;
        else
            r=mid;}
    return {l,r};}

public:
    //slope increasing or decreasing(not query point
    //query point is arbitrary),querying for maximum or minimum
    MonotoneCHT(bool increasing,bool maximum){
        type=increasing;
        if(maximum)
            type|=2;}
    void insert(Line nl){
        if(type==3||type==0)
            insertBack(nl);
        else
            insertFront(nl);}
    ll fastQuery(ll x){//if monotone query satisfied //not
        tested,although it should be ok
    {
        #ifndef INCREASING_QUERY
        while(Q.size())>1&&Q[0].intersect(Q[1]).first<x)
            Q.pop_front();
        return Q[0](x);
        #else
        while(Q.size())>1&&Q.back().intersect(Q[(int)Q.size()-2]).first>=x)
            Q.pop_back();
        return Q.back()(x);
        #endif}
    ll query(ll x){
        pii indx=bSearch(x);
        if(type<2)
            return min(Q[indx.first](x),Q[indx.second](x));
        return max(Q[indx.first](x),Q[indx.second](x));}
    void clear()
    {Q.clear();};}

```

## 2.12 segment tree beats

```

#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int MAXN = 200001; // 1-based
int N;
ll A[MAXN];
struct Node {
    ll sum,max1,max2,maxc,min1,
    min2,minc,lazy;
    //max1->ith max,maxc->maxcount
} T[MAXN * 4];
void merge(int t) {
    // sum
    T[t].sum = T[t << 1].sum + T[t << 1 | 1].sum;
    // max
    if (T[t << 1].max1 == T[t << 1 | 1].max1) {
        T[t].max1 = T[t << 1].max1;
        T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max2);
        T[t].maxc = T[t << 1].maxc + T[t << 1 | 1].maxc;
    } else {
        if (T[t << 1].max1 > T[t << 1 | 1].max1) {
            T[t].max1 = T[t << 1].max1;
            T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max1);
            T[t].maxc = T[t << 1].maxc;
        } else {
            T[t].max1 = T[t << 1 | 1].max1;
            T[t].max2 = max(T[t << 1].max1, T[t << 1 | 1].max2);
            T[t].maxc = T[t << 1 | 1].maxc;
        }
    }
    // min
    if (T[t << 1].min1 == T[t << 1 | 1].min1) {
        T[t].min1 = T[t << 1].min1;
        T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min2);
        T[t].minc = T[t << 1].minc + T[t << 1 | 1].minc;
    } else {

```

```

        if (T[t << 1].min1 < T[t << 1 | 1].min1) {
            T[t].min1 = T[t << 1].min1;
            T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min1);
            T[t].minc = T[t << 1].minc;
        } else {
            T[t].min1 = T[t << 1 | 1].min1;
            T[t].min2 = min(T[t << 1].min1, T[t << 1 | 1].min2);
            T[t].minc = T[t << 1 | 1].minc;
        }
    }
}

void push_add(int t, int tl, int tr, ll v) {
    if (v == 0)
        return;
    T[t].sum += (tr - tl + 1) * v;
    T[t].max1 += v;
    if (T[t].max2 != -11INF) {
        T[t].max2 += v;
    }
    T[t].min1 += v;
    if (T[t].min2 != 11INF) {
        T[t].min2 += v;
    }
    T[t].lazy += v;
}

// corresponds to a chmin update
void push_max(int t, ll v, bool l) {
    if (v >= T[t].max1)
        return;
    T[t].sum -= T[t].max1 * T[t].maxc;
    T[t].max1 = v;
    T[t].sum += T[t].max1 * T[t].maxc;
    if (l) {
        T[t].min1 = T[t].max1;
    } else {
        if (v <= T[t].min1) {
            T[t].min1 = v;
        } else if (v < T[t].min2) {
            T[t].min2 = v;
        }
    }
}

// corresponds to a chmax update
void push_min(int t, ll v, bool l) {
    if (v <= T[t].min1)
        return;
    T[t].sum -= T[t].min1 * T[t].minc;
    T[t].min1 = v;
    T[t].sum += T[t].min1 * T[t].minc;
    if (l) {
        T[t].max1 = T[t].min1;
    } else {
        if (v >= T[t].max1) {
            T[t].max1 = v;
        } else if (v > T[t].max2) {
            T[t].max2 = v;
        }
    }
}

void pushdown(int t, int tl, int tr) {
    if (tl == tr)
        return;
    // sum
    int tm = (tl + tr) >> 1;
    push_add(t << 1, tl, tm, T[t].lazy);
    push_add(t << 1 | 1, tm + 1, tr, T[t].lazy);
    T[t].lazy = 0;
    // max
    push_max(t << 1, T[t].max1, tl == tm);
    push_max(t << 1 | 1, T[t].max1, tm + 1 == tr);
    // min
    push_min(t << 1, T[t].min1, tl == tm);
    push_min(t << 1 | 1, T[t].min1, tm + 1 == tr);
}

void build(int t=1, int tl=0, int tr=N-1) {
    T[t].lazy = 0;
    if (tl == tr) {
        T[t].sum = T[t].max1 = T[t].min1 = A[tl];
        T[t].maxc = T[t].minc = 1;
        T[t].max2 = -11INF;

```

```

        T[t].min2 = 11INF;
        return;
    }
    int tm = (tl + tr) >> 1;
    build(t << 1, tl, tm);
    build(t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_add(int l, int r, ll v, int t=1, int tl=0, int
    tr=N-1) {
    if (r < tl || tr < l)
        return;
    if (l <= tl && tr <= r) {
        push_add(t, tl, tr, v);
        return;
    }
    pushdown(t, tl, tr);
    int tm = (tl + tr) >> 1;
    update_add(l, r, v, t << 1, tl, tm);
    update_add(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_chmin(int l, int r, ll v, int t=1, int tl=0, int
    tr=N-1) {
    if (r < tl || tr < l || v >= T[t].max1) {
        return;
    }
    if (l <= tl && tr <= r && v > T[t].max2) {
        push_max(t, v, tl == tr);
        return;
    }
    pushdown(t, tl, tr);
    int tm = (tl + tr) >> 1;
    update_chmin(l, r, v, t << 1, tl, tm);
    update_chmin(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_chmax(int l, int r, ll v, int t=1, int tl=0, int
    tr=N-1) {
    if (r < tl || tr < l || v <= T[t].min1)
        return;
    if (l <= tl && tr <= r && v < T[t].min2) {
        push_min(t, v, tl == tr);
        return;
    }
    pushdown(t, tl, tr);
    int tm = (tl + tr) >> 1;
    update_chmax(l, r, v, t << 1, tl, tm);
    update_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

ll query_sum(int l, int r, int t=1, int tl=0, int tr=N-1) {
    if (r < tl || tr < l)
        return 0;
    if (l <= tl && tr <= r)
        return T[t].sum;
    pushdown(t, tl, tr);
    int tm = (tl + tr) >> 1;
    return query_sum(l, r, t << 1, tl, tm) + query_sum(l, r, t
        << 1 | 1, tm + 1, tr);
}

int main() {
    int Q;
    cin >> N >> Q;
    for (int i = 0; i < N; i++) {
        cin >> A[i];
    }
    build();
    for (int q = 0; q < Q; q++) {
        int t; cin >> t;
        if (t == 0) {
            int l, r;
            ll x;
            cin >> l >> r >> x;
            update_chmin(l, r - 1, x);
        } else if (t == 1) {
            int l, r;
            ll x;
            cin >> l >> r >> x;
            update_chmax(l, r - 1, x);

```

```

} else if (t == 2) {
    int l, r;
    ll x;
    cin >> l >> r >> x;
    update_add(l, r - 1, x);
} else if (t == 3) {
    int l, r;
    cin >> l >> r;
    cout << query_sum(l, r - 1) << '\n';
}
}
}

```

## 2.13 sparse table 2d

```

const int N=500; const int K = 8 ; /// k >= ceil(lg22(n)) +1
int arr[N][N]; int st[K+1][K+1][N][N]; int lg2[N+1];
void ini(){ lg2[1] = 0;
    for (int i = 2; i <= N; i++) lg2[i] = lg2[i/2] + 1; }
int f(int i, int j){ return max(i,j); }
void pre( int n,int m){
    for(int x=0;x<=K;x++){
        for(int y=0;y<=K;y++){
            for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                    if(i+(1<<x)>n or j+(1<<y) > m) continue;
                    if(x>0) st[x][y][i][j] = f(st[x-1][y][i][j] ,
                                                st[x-1][y][i+(1<<(x-1))][j]);
                    else if(y>0) st[x][y][i][j] = f(st[x][y-1][i][j] ,
                                                st[x][y-1][i][j+(1<<(y-1))]);
                    else st[x][y][i][j] = f(arr[i][j] , arr[i][j]);
                } } } }
    int getf( int R1 ,int C1 , int R2 , int C2){
        if(R1>R2) swap(R1,R2); if(C1>C2) swap(C1,C2);
        int x = lg2[R2 - R1 + 1]; int y = lg2[C2 - C1 + 1];
        return f(f(st[x][y][R1][C1],st[x][y][R2-(1<<x)+1][C1]),
                f(st[x][y][R1][C2-(1<<y)+1],st[x][y][R2-(1<<x)+1][C2-(1<<y)+1]));
    }
}

```

## 2.14 treap

```

struct node {
    int prior, size;
    ll val, sum, lazy, rev;
    node *l, *r;
    node(int v = 0) {
        val = sum = v;
        lazy = 0; rev = 0;
        prior = rand();
        size = 1;
        l = r = NULL;
    }
} *root;
///If you initialize the root (root = new node()) then treap
///will be 1 based indexing and root will be the 0th index
///Otherwise it will be 0 based indexing, so decide carefully
typedef node* pnode;

int sz(pnode t) { return t ? t->size : 0; }
void upd_sz(pnode t) { if(t) t->size = sz(t->l) + 1 + sz(t->r); }

void push(pnode t) {
    if(!t) return;
    if(t->lazy){
        t->val += t->lazy;
        t->sum += t->lazy * sz(t);
        if(t->l) t->l->lazy += t->lazy;
        if(t->r) t->r->lazy += t->lazy;
        t->lazy = 0;
    }
    //Reverse
    if(t->rev){
        swap(t->l,t->r);
        if(t->l) t->l->rev ^= t->rev;
        if(t->r) t->r->rev ^= t->rev;
        t->rev = 0;
    }
}

```

```

void combine(pnode t) {
    // Note: This function should reset datas of t first
    (MUST!!)
    // Then update datas from l and r.
    if(!t) return;
    push(t->l);
    push(t->r);
    t->sum = t->val; // Reset
    if(t->l) t->sum += t->l->sum;
    if(t->r) t->sum += t->r->sum;
}

```

```

void split(pnode t, pnode &l, pnode &r, int pos, int add = 0) {
    if(!t) return void(l = r = NULL);
    push(t);
    int curr = sz(t->l) + add;
    if(curr <= pos)
        split(t->r, t->r, r, pos, curr + 1), l = t;
    else split(t->l, l, t->l, pos, add), r = t;
    upd_sz(t);
    combine(t);
}

```

```

void merge(pnode &t, pnode l, pnode r) {
    push(l), push(r);
    if(!l || !r)
        t = l ? l : r;
    else if(l->prior > r->prior)
        merge(l->r, l->r, r), t = l;
    else merge(r->l, l, r->l), t = r;
    upd_sz(t);
    combine(t);
}

```

```

ll query(pnode t, int l, int r) {
    pnode L, mid, R;
    split(t, L, mid, l - 1);
    split(mid, t, R, r - 1);
    ll ans = t->sum;
    merge(mid, L, t);
    merge(t, mid, R);
    return ans;
}

```

```

void update(pnode t, int l, int r, ll v) {
    pnode L, mid, R;
    split(t, L, mid, l - 1);
    split(mid, t, R, r - 1);
    t->lazy += v;
    merge(mid, L, t);
    merge(t, mid, R);
}

```

```

void insert(pnode &t, int pos, int v) {
    pnode L, R, tmp, y = new node(v);
    split(t, L, R, pos - 1);
    merge(tmp, L, y);
    merge(t, tmp, R);
}

```

```

void Del(pnode &t, int pos) {
    pnode L, R, mid;
    split(t, L, mid, pos - 1);
    split(mid, t, R, 0);
    pnode tmp = t;
    merge(t, L, R);
    free(tmp);
}

```

```

void Reverse(pnode t, int l, int r) {
    pnode L, mid, R;
    split(t, L, mid, l - 1);
    split(mid, t, R, r - 1);
    t->rev = 1;
    merge(mid, L, t);
    merge(t, mid, R);
}

```

## 3 Extra

### 3.1 Header

```

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef double ftype;
typedef pair<ftype,ftype> pff;
#define all(a) a.begin(), a.end()
#define some(a,l,r) a.begin()+l,a.begin()+r+1
#define csort(a) sort(all(a))
#define pub push_back
#define puf push_front
#define pob pop_back
#define pof pop_front
#define fi first
#define se second
#define fastio ios_base::sync_with_stdio(false);cin.tie(NULL)
#ifdef COMEDIANS
#define infile ;
#define outfile ;
#define Gene template< class
#define Rics printer& operator,
Gene c> struct rge{c b, e;};
Gene c> rge<c> range(c i, c j){ return {i, j};}
struct printer{
    ~printer(){cerr<<endl;}
    Gene c >Rics(c x){ cerr<<boolalpha<<x; return *this;}
    Rics(string x){cerr<<x;return *this;}
    Gene c, class d >Rics(pair<c, d> x){ return
        *this,"(x.first, ", x.second,")";}
    Gene ... d, Gene ... > class c >Rics(c d... x){ return
        *this, range(begin(x), end(x));}
    Gene c >Rics(rge<c> x){
        *this,"["; for(auto it = x.b; it != x.e; ++it)
            *this,(it==x.b?"":", "),*it; return *this,"]";}
};
#define stop getchar()
#define debug() cerr<<"LINE "<<__LINE__<<" >> ", printer()
#define dbg(x) debug(), "[" ,#x,": ",(x),"] "
#define test_handle(T) cin>>T

void err(istream_iterator<string> it) {cout<<endl;}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args){
    cout << *it << " = " << a << " ";err(++it, args...);
}
template<class T1, class T2>
ostream &operator <<(ostream &os, pair<T1,T2>&p) {
    os<<"{"<<p.first<<"", "<<p.second<<""} ";
    return os;
}
#define dbg2(args...) { string _s = #args; replace(_s.begin(),
    _s.end(), ',', ' '); stringstream _ss(_s);
    istream_iterator<string> _it(_ss); err(_it, args); }
#else
#define dbg(x) ;
#define infile ;
#define outfile ;
#define test_handle(T) T = 1
#define dbg2(args...) ;
#endif
//Use -DCOMEDIANS in compiler flag in others tab, or remove
#ifdef
mt19937
rng(chrono::steady_clock::now().time_since_epoch().count());
const ftype EPS = 1e-10;
const ftype PI = acos(-1);
const int MAX = 3e5+5;
const int BMAX = 18;
const int MOD = 1e9+7;
using namespace __gnu_pbds;
/*
find_by_order(k) --> returns iterator to the kth largest
element counting from 0
order_of_key(val) --> returns the number of items in a set
that are strictly smaller than our item
*/

```

```

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;
// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC target("avx2,bmi,bmi2,lzcnt")
// mt19937
rng(chrono::system_clock::now().time_since_epoch().count());
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct custom_hash {
static uint64_t splitmix64(uint64_t x) {
x += 0x9e3779b97f4a7c15; //Random
x=(x^(x>>30))*0xbf58476d1ce4e5b9; //Random
x=(x^(x>>27))*0x94d049bb133111eb; //Random
return x^(x>>31);
}
const uint64_t FIXED_RANDOM = chrono::
steady_clock::now().time_since_epoch().count();
size_t operator()(uint64_t x) const {
return splitmix64(x + FIXED_RANDOM);
}
}
size_t operator()(pair<int, int> x) const {
return splitmix64((uint64_t(x.first)<<32) +
x.second + FIXED_RANDOM);
}
};
gp_hash_table<pair<int,int>,int,custom_hash> ht;
namespace my_gcc_ints {
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wpedantic"
using int128 = __int128;
#pragma GCC diagnostic pop
}
# stresstester GENERATOR SOL1 SOL2 ITERATIONS
for i in $(seq 1 "$4") ; do
echo -en "\rAttempt $i/$4"
$1 > in.txt
$2 < in.txt > out1.txt
$3 < in.txt > out2.txt
diff -y out1.txt out2.txt > diff.txt
if [ $? -ne 0 ] ; then
echo -e "\nTestcase Found:"; cat in.txt
echo -e "\nOutputs:"; cat diff.txt
exit
fi
done
//generate all submask
int s = m;
while (s > 0) {
s = (s-1) & m;
}

```

## 4 Game

### 4.1 HackenBush

```

/* tree case: g[u] = for all v : XOR[ g[v] + 1]
lose if no moves available
1. Colon Principle: Grundy number of a tree is the
xor of Grundy number of child subtrees.
2. Fusion Principle: Consider a pair of adjacent
vertices u, v that has another path (i.e.,
they are in a cycle). Then, we can contract u
and v without changing Grundy number.
We first decompose graph into two-edge connected
components. Then, by contracting each components by
using Fusion Principle, we obtain a tree (and many
self loops) that has the same Grundy number to the
original graph. By using Colon Principle, we can
compute the Grundy number. O(m + n). */
struct hackenbush {
int n; vector<vector<int>> adj;
hackenbush(int n) : n(n), adj(n) {}
void add_edge(int u, int v) {
adj[u].push_back(v);
if(u!=v) adj[v].push_back(u);
}
}

```

```

int Grundy(int r) {
vector<int> num(n), low(n); int t = 0;
function<int(int, int)> dfs=&[(int p,int u) {
num[u] = low[u] = ++t; int ans = 0;
for (int v : adj[u]) {
if (v == p) { p += 2 * n; continue; }
if (num[v] == 0) {
int res = dfs(u, v);
low[u] = min(low[u], low[v]);
if (low[v] > num[u]) ans ^= (1+res)^1;
else ans ^= res; // non bridge
}
else low[u] = min(low[u], num[v]);
}
if (p > n) p -= 2 * n;
for (int v : adj[u])
if (v != p && num[u] <= num[v]) ans ^= 1;
return ans;
};
return dfs(-1, r);
}
};

```

## 5 Geo

### 5.1 3dGeo

```

int dcmp(double x) { return abs(x) < EPS ? 0 : (x<0 ? -1 : 1); }
double degreeToRadian(double rad) { return rad*PI/180; }
struct Point {
double x, y, z; Point() : x(0), y(0), z(0) {}
Point(double X, double Y, double Z) : x(X), y(Y), z(Z) {}
Point operator + (const Point& u) const {
return Point(x + u.x, y + u.y, z + u.z); }
Point operator - (const Point& u) const {
return Point(x - u.x, y - u.y, z - u.z); }
Point operator * (const double u) const {
return Point(x * u, y * u, z * u); }
Point operator / (const double u) const {
return Point(x / u, y / u, z / u); }
};
double dot(Point a, Point b) { return a.x*b.x+a.y*b.y+a.z*b.z; }
Point cross(Point a, Point b) { return
Point(a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z, a.x*b.y - a.y*b.x); }
double length(Point a) { return sqrt(dot(a, a)); }
double distance(Point a, Point b) { return length(a-b); }
Point unit(const Point &p) { return p/length(p); }
// Rotate p around axis x, with angle radians.
Point rotate(Point p, Point axis, double angle) {
axis = unit(axis); Point comp1 = p * cos(angle);
Point comp2 = axis * (1-cos(angle)) * dot(axis, p);
Point comp3 = cross(axis, p) * sin(angle);
return comp1 + comp2 + comp3;
}
struct Line {Point a, v;}; //a+tv
// returns the distance from point a to line l
double distancePointLine(Point p, Line l) {
return length(cross(l.v, p - l.a)) / length(l.v); }
// distance from Line ab to Line cd
double distanceLineLine(Line a, Line b) {
Point cr = cross(a.v, b.v); double crl = length(cr);
if (dcmp(crl) == 0) return distancePointLine(a.a, b);
return abs(dot(cr, a.a-b.a))/crl; }
struct Plane {
Point normal; // Normal = (A, B, C)
double d; // dot(Normal) = d <--> Ax + By + Cz = d
Point P; // anyPoint on the plane, optional
Plane(Point normal, double d) {
double len = length(normal); assert(dcmp(len) > 0);
normal = normal / len; d = d / len;
if (dcmp(normal.x) > 0) P = Point(d/normal.x, 0, 0);
else if (dcmp(normal.y) > 0) P = Point(0, d/normal.y, 0);
else P = Point(0, 0, d/normal.z);
}
//Plane given by three Non-Collinear Points
Plane(Point a, Point b, Point c) {
normal = unit(cross(b-a, c-a)); d = dot(normal,a); P=a;
}
}

```

```

}
bool onPlane(Point a) { return dcmp(dot(normal,a)-d) == 0; }
double distance(Point a) { return abs(dot(normal, a) - d); }
double isParallel(Line l) { return dcmp(dot(l.v,normal)) == 0; }
//return t st l.a + t*l.v is a point on the plane, check
//parallel first
double intersectLine(Line l) {
return dot(P-l.a, normal)/dot(l.v, normal); }
};

```

### 5.2 Circle Cover

```

//Check if the all of the area of circ(0, R) in
//Circ(00, RR) is covered by some other circle
bool CoverCircle(PT O, double R, vector<PT> &cen,
vector<double> &rad, PT OO, double RR) {
int n = cen.size();
vector<pair<double, double>> arcs;
for (int i=0; i<n; i++) {
PT P = cen[i]; double r = rad[i];
if (i!=0 && R + sqrt(dist2(O, P))<r) return 1;
if (i==0 && r + sqrt(dist2(O, P))<R) return 1;
vector<PT> inter =
CircleCircleIntersection(O, P, R, r);
if (inter.size() <= 1) continue;
PT X = inter[0], Y = inter[1];
if (cross(O, X, Y) < 0) swap(X, Y);
if (!(cross(O, X, P) >= 0 &&
cross(O, Y, P) <= 0)) swap(X, Y);
if (i==0) swap(X, Y);
X = X-O; Y=Y-O;
double ll = atan2(X.y, X.x);
double rr = atan2(Y.y, Y.x);
if (rr < ll) rr += 2*PI;
arcs.emplace_back(ll, rr);
}
if (arcs.empty()) return false;
sort(arcs.begin(), arcs.end());
double st = arcs[0].ff, en = arcs[0].ss, ans = 0;
for (int i=1; i<arcs.size(); i++) {
if (arcs[i].first <= en + EPS)
en = max(en, arcs[i].second);
else st = arcs[i].first, en = arcs[i].second;
ans = max(ans, en-st);
}
return ans >= 2*PI;
}

```

### 5.3 Circle Union Area

```

struct Point {
LD x, y;
LD operator*(const Point &a) const {
return x*a.y-y*a.x; }
LD operator/(const Point &a) const {
return sqrt((a.x-x)*(a.x-x)+(a.y-y)*(a.y-y)); }
}
LD po[N];
LD r[N];
int sgn(LD x) { return fabs(x)<EPS?0:(x>0.0?1:-1); }
pair<LD, bool> ARG[2*N];
LD cir_union(Point c[], LD r[], int n) {
LD sum = 0.0, sum1 = 0.0, d, p1, p2, p3;
for (int i = 0; i < n; i++) {
bool f = 1;
for (int j = 0; j < n; j++) {
if (i!=j && sgn(r[j]-r[i]-c[i]/c[j])!=1) f=0;
if (!f) swap(r[i], r[j]), swap(c[i], c[j]);
}
for (int i = 0; i < n; i++) {
int k = 0, cnt = 0;
for (int j = 0; j < n; j++) {
if (i!=j && sgn((d=c[i]/c[j]-r[i]-r[j])<=0){
p3=acos((r[i]*r[i]+d*d-r[j]*r[j])/(
2.0*r[i]*d));
p2=atan2(c[j].y-c[i].y,c[j].x-c[i].x);
p1 = p2-p3; p2 = p2+p3;
if (sgn(p1+PI)==-1) p1+=2*PI, cnt++;
if (sgn(p2-PI)==1) p2-=2*PI, cnt++;
ARG[k++] = make_pair(p1, 0);
ARG[k++] = make_pair(p2, 1);
}
}
}

```

```

    }
}
if(k) {
    sort(ARG, ARG+k);
    p1 = ARG[k-1].first-2*PI;
    p3 = r[i]*r[i];
    for(int j = 0; j < k; j++) {
        p2 = ARG[j].first;
        if(cnt==0) {
            sum+=(p2-p1-sin(p2-p1))*p3;
            sum1+=(c[i]+Point(cos(p1), sin(p1))*
                r[i])*(c[i]+
                Point(cos(p2), sin(p2))*r[i]);
        }
        p1 = p2;
        ARG[j].second ? cnt--:cnt++;
    }
}
else sum += 2*PI*r[i]*r[i];
}
return (sum+fabs(sum1))*0.5;
}

```

#### 5.4 basic-area-geometry

```

struct point2d {
    ftype x, y; point2d() {}
    point2d(ftype x, ftype y): x(x), y(y) {}
    point2d& operator+=(const point2d &t) {
        x += t.x; y += t.y; return *this;
    }
    point2d& operator-=(const point2d &t) {
        x -= t.x; y -= t.y; return *this;
    }
    point2d& operator==(ftype t) {
        x == t; y == t; return *this;
    }
    point2d& operator/=(ftype t) {
        x /= t; y /= t; return *this;
    }
    point2d operator+(const point2d &t) const {
        return point2d(*this) += t;
    }
    point2d operator-(const point2d &t) const {
        return point2d(*this) -= t;
    }
    point2d operator*(ftype t) const {
        return point2d(*this) *= t;
    }
    point2d operator/(ftype t) const {
        return point2d(*this) /= t;
    }
};
point2d operator*(ftype a, point2d b) {
    return b * a;
}

struct point3d {
    ftype x, y, z; point3d() {}
    point3d(ftype x, ftype y, ftype z): x(x), y(y), z(z) {}
    point3d& operator+=(const point3d &t) {
        x += t.x; y += t.y; z += t.z; return *this;
    }
    point3d& operator-=(const point3d &t) {
        x -= t.x; y -= t.y; z -= t.z; return *this;
    }
    point3d& operator==(ftype t) {
        x == t; y == t; z == t; return *this;
    }
    point3d& operator/=(ftype t) {
        x /= t; y /= t; z /= t; return *this;
    }
    point3d operator+(const point3d &t) const {
        return point3d(*this) += t;
    }
    point3d operator-(const point3d &t) const {
        return point3d(*this) -= t;
    }
    point3d operator*(ftype t) const {

```

```

        return point3d(*this) *= t;
    }
    point3d operator/(ftype t) const {
        return point3d(*this) /= t;
    }
};
point3d operator*(ftype a, point3d b) {
    return b * a;
}
ftype dot(point2d a, point2d b) {
    return a.x * b.x + a.y * b.y;
}
ftype dot(point3d a, point3d b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
ftype norm(point2d a) {
    return dot(a, a);
}
double abs(point2d a) {
    return sqrt(norm(a));
}
double proj(point2d a, point2d b) {
    return dot(a, b) / abs(b);
}
double angle(point2d a, point2d b) {
    return acos(dot(a, b) / abs(a) / abs(b));
}
point3d cross(point3d a, point3d b) {
    return point3d(a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
}
ftype triple(point3d a, point3d b, point3d c) {
    return dot(a, cross(b, c));
}
ftype cross(point2d a, point2d b) {
    return a.x * b.y - a.y * b.x;
}
point2d intersect(point2d a1, point2d d1,
    point2d a2, point2d d2) {
    return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
}
point3d intersect(point3d a1, point3d n1, point3d a2,
    point3d n2, point3d a3, point3d n3) {
    point3d x(n1.x, n2.x, n3.x); point3d y(n1.y, n2.y, n3.y);
    point3d z(n1.z, n2.z, n3.z);
    point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
    return point3d(triple(d, y, z), triple(x, d, z),
        triple(x, y, d)) / triple(n1, n2, n3);
}
int signed_area_parallelogram(p2d p1, point2d p2, point2d p3) {
    return cross(p2 - p1, p3 - p2);
}
double triangle_area(point2d p1, point2d p2, point2d p3) {
    return abs(signed_area_parallelogram(p1, p2, p3)) / 2.0;
}
bool clockwise(point2d p1, point2d p2, point2d p3) {
    return signed_area_parallelogram(p1, p2, p3) < 0;
}
bool counter_clockwise(point2d p1, point2d p2, point2d p3) {
    return signed_area_parallelogram(p1, p2, p3) > 0;
}
double area(const vector<point>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        point p = i? fig[i-1] : fig.back(); point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}
//Pick: S = I + B/2 - 1
int count_lattices(Fraction k, Fraction b, long long n) {
    auto fk = k.floor(); auto fb = b.floor(); auto cnt = 0LL;
    if (k >= 1 || b >= 1) {
        cnt+=(fk*(n-1) + 2 * fb) * n / 2; k -= fk; b -= fb;
    }
    auto t = k * n + b; auto ft = t.floor();
    if (ft >= 1)
        cnt += count_lattices(1 / k, (t - t.floor()) / k, t.floor());
}

```

```

    return cnt;
}

```

#### 5.5 geo-formulae

##### Triangle Centers and Radii

- Incenter:  $\left(\frac{ax_1+bx_2+cx_3}{a+b+c}, \frac{ay_1+by_2+cy_3}{a+b+c}\right)$ .
- Inradius:  $\sqrt{\frac{(s-a)(s-b)(s-c)}{s}}$ .
- Excenter:  $\left(\frac{-ax_1+bx_2+cx_3}{-a+b+c}, \frac{-ay_1+by_2+cy_3}{-a+b+c}\right)$ .
- Exradius:  $\sqrt{\frac{s(s-b)(s-c)}{(s-a)}}$ .
- Circumcenter:  $\left(\frac{x_1 \sin 2A + x_2 \sin 2B + x_3 \sin 2C}{\sin 2A + \sin 2B + \sin 2C}, \frac{y_1 \sin 2A + y_2 \sin 2B + y_3 \sin 2C}{\sin 2A + \sin 2B + \sin 2C}\right)$ .
- Circumradius:  $\frac{abc}{\sqrt{(a+b+c)(b+c-a)(c+a-b)(a+b-c)}}$ .

#### 5.6 half-plane-intersection

```

class HalfPlaneIntersection{
    static double eps, inf;
public:
    struct Point{
        double x, y;
        explicit Point(double x = 0, double y = 0) : x(x), y(y) {}
        // Addition, subtraction, multiply by constant, cross product.
        friend Point operator + (const Point& p, const Point& q){
            return Point(p.x + q.x, p.y + q.y);
        }
        friend Point operator - (const Point& p, const Point& q){
            return Point(p.x - q.x, p.y - q.y);
        }
        friend Point operator * (const Point& p, const double& k){
            return Point(p.x * k, p.y * k);
        }
        friend double cross(const Point& p, const Point& q){
            return p.x * q.y - p.y * q.x;
        }
    };
    // Basic half-plane struct.
    struct Halfplane{
        // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
        Point p, pq;
        double angle;
        Halfplane() {}
        Halfplane(const Point& a, const Point& b) : p(a), pq(b - a){
            angle = atan2(pq.y, pq.x);
        }
        // Check if point 'r' is outside this half-plane.
        // Every half-plane allows the region to the LEFT of its line.
        bool out(const Point& r){
            return cross(pq, r - p) < -eps;
        }
        // Comparator for sorting.
        // If the angle of both half-planes is equal, the leftmost one should go first.
        bool operator < (const Halfplane& e) const{
            if (fabs(angle - e.angle) < eps) return cross(pq, e.p - p) < 0;
            return angle < e.angle;
        }
    };
}

```



```

}
// We use equal comparator for std::unique to easily
// remove parallel half-planes.
bool operator==(const Halfplane& e) const{
    return fabs(angle - e.angle) < eps;
}
// Intersection point of the lines of two half-planes.
// It is assumed they're never parallel.
friend Point inter(const Halfplane& s, const Halfplane&
t){
    double alpha = cross((t.p - s.p), t.pq) /
        cross(s.pq, t.pq);
    return s.p + (s.pq * alpha);
}
};
static vector<Point> hp_intersect(vector<Halfplane>& H){
    Point box[4] = // Bounding box in CCW order{
        Point(-inf, inf),
        Point(inf, inf),
        Point(inf, -inf),
        Point(-inf, -inf)
    };
    for(int i = 0; i < 4; i++) // Add bounding box
        half-planes.{
            Halfplane aux(box[i], box[(i+1) % 4]);
            H.push_back(aux);
        }
    // Sort and remove duplicates
    sort(H.begin(), H.end());
    H.erase(unique(H.begin(), H.end()), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++){
        // Remove from the back of the deque while last
        // half-plane is redundant
        while(len > 1 && H[i].out(inter(dq[len-1],
            dq[len-2]))){
            dq.pop_back();
            --len;
        }
        // Remove from the front of the deque while first
        // half-plane is redundant
        while(len > 1 && H[i].out(inter(dq[0], dq[1]))){
            dq.pop_front();
            --len;
        }
        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }
    // Final cleanup: Check half-planes at the front
    // against the back and vice-versa
    while(len > 2 && dq[0].out(inter(dq[len-1],
        dq[len-2]))){
        dq.pop_back();
        --len;
    }
    while(len > 2 && dq[len-1].out(inter(dq[0], dq[1]))){
        dq.pop_front();
        --len;
    }
    // Report empty intersection if necessary
    if(len < 3) return vector<Point>();
    // Reconstruct the convex polygon from the remaining
    // half-planes.
    vector<Point> ret(len);
    for(int i = 0; i < len; i++){
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
};
double HalfPlaneIntersection::eps=1e-9;
double HalfPlaneIntersection::inf=1e9;

```

## 5.7 heart-of-geometry-2d

```
typedef double ftype;
```

```

const double EPS = 1E-9;
struct pt{
    ftype x, y;
    int id;
    pt() {}
    pt(ftype _x, ftype _y):x(_x), y(_y) {}
    pt operator+(const pt & p) const{
        return pt(x + p.x, y + p.y);
    }
    pt operator-(const pt & p) const{
        return pt(x - p.x, y - p.y);
    }
    ftype cross(const pt & p) const{
        return x * p.y - y * p.x;
    }
    ftype dot(const pt & p) const{
        return x * p.x + y * p.y;
    }
    ftype cross(const pt & a, const pt & b) const{
        return (a - *this).cross(b - *this);
    }
    ftype dot(const pt & a, const pt & b) const{
        return (a - *this).dot(b - *this);
    }
    ftype sqrLen() const{
        return this->dot(*this);
    }
    bool operator<(const pt& p) const{
        return x < p.x - EPS || (abs(x - p.x) < EPS && y < p.y
            - EPS);
    }
    bool operator==(const pt& p) const{
        return abs(x-p.x)<EPS && abs(y-p.y)<EPS;
    }
};
int sign(double x) { return (x > EPS) - (x < -EPS); }
inline int orientation(pt a, pt b, pt c) { return
    sign(a.cross(b,c)); }
bool is_point_on_seg(pt a, pt b, pt p) {
    if (fabs(b.cross(p,a)) < EPS) {
        if (p.x < min(a.x, b.x) - EPS || p.x > max(a.x, b.x) +
            EPS) return false;
        if (p.y < min(a.y, b.y) - EPS || p.y > max(a.y, b.y) +
            EPS) return false;
        return true;
    }
    return false;
}
bool is_point_on_polygon(vector<pt> &p, const pt& z) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (is_point_on_seg(p[i], p[(i + 1) % n], z)) return 1;
    }
    return 0;
}
int winding_number(vector<pt> &p, const pt& z) { // O(n)
    if (is_point_on_polygon(p, z)) return 1e9;
    int n = p.size(), ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        bool below = p[i].y < z.y;
        if (below != (p[j].y < z.y)) {
            auto orient = orientation(z, p[j], p[i]);
            if (orient == 0) return 0;
            if (below == (orient > 0)) ans += below ? -1 : 1;
        }
    }
    return ans;
}
double dist_sqr(pt a,pt b){
    return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
double dist(pt a, pt b){
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
double angle(pt a,pt b,pt c){
    if(b==a || b==c) return 0;
    double A2 = dist_sqr(b,c);

```

```

double C2 = dist_sqr(a,b);
double B2 = dist_sqr(c,a);
double A = sqrt(A2), C = sqrt(C2);
double ans = (A2 + C2 - B2)/(A*C*2);
if(ans<-1) ans=acos(-1);
else if(ans>1) ans=acos(1);
else ans = acos(ans);
return ans;
}
bool cmp(pt a, pt b){
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool ccw(pt a, pt b, pt c, bool include_collinear=false) {
    int o = orientation(a, b, c);
    return o > 0 || (include_collinear && o == 0);
}
bool cw(pt a, pt b, pt c, bool include_collinear=false) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c)
    == 0; }
double area(pt a, pt b, pt c){
    return (a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y))/2;
}
struct cmp_x{
    bool operator()(const pt & a, const pt & b) const{
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
};
struct cmp_y{
    bool operator()(const pt & a, const pt & b) const{
        return a.y < b.y || (a.y == b.y && a.x < b.x);
    }
};
struct circle : pt {
    ftype r;
};
bool insideCircle(circle c, pt p){
    return dist_sqr(c,p) <= c.r*c.r + EPS;
}
struct line {
    ftype a, b, c;
    line() {}
    line(pt p, pt q){
        a = p.y - q.y;
        b = q.x - p.x;
        c = -a * p.x - b * p.y;
        norm();
    }
    void norm(){
        double z = sqrt(a * a + b * b);
        if (abs(z) > EPS)
            a /= z, b /= z, c /= z;
    }
    line getParallel(pt p){
        line ans = *this;
        ans.c = -(ans.a*p.x+ans.b*p.y);
        return ans;
    }
    ftype getValue(pt p){
        return a*p.x+b*p.y+c;
    }
    line getPerpend(pt p){
        line ans;
        ans.a=this->b;
        ans.b=-(this->a);
        ans.c = -(ans.a*p.x+ans.b*p.y);
        return ans;
    }
    //dist formula is wrong but don't change
    double dist(pt p) const { return a * p.x + b * p.y + c; }
};
double sqr (double a) {
    return a * a;
}
double det(double a, double b, double c, double d) {
    return a*d - b*c;
}

```

```

bool intersect(line m, line n, pt & res) {
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS)
        return false;
    res.x = -det(m.c, m.b, n.c, n.b) / zn;
    res.y = -det(m.a, m.c, n.a, n.c) / zn;
    return true;
}

bool parallel(line m, line n) {
    return abs(det(m.a, m.b, n.a, n.b)) < EPS;
}

bool equivalent(line m, line n) {
    return abs(det(m.a, m.b, n.a, n.b)) < EPS
        && abs(det(m.a, m.c, n.a, n.c)) < EPS
        && abs(det(m.b, m.c, n.b, n.c)) < EPS;
}

double det(double a, double b, double c, double d){
    return a * d - b * c;
}

inline bool betw(double l, double r, double x){
    return min(l, r) <= x + EPS && x <= max(l, r) + EPS;
}

inline bool intersect_1d(double a, double b, double c, double d){
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d) + EPS;
}

bool intersect_segment(pt a, pt b, pt c, pt d, pt& left, pt& right){
    if (!intersect_1d(a.x, b.x, c.x, d.x) ||
        !intersect_1d(a.y, b.y, c.y, d.y))
        return false;
    line m(a, b);
    line n(c, d);
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS) {
        if (abs(m.dist(c)) > EPS || abs(n.dist(a)) > EPS)
            return false;
        if (b < a)
            swap(a, b);
        if (d < c)
            swap(c, d);
        left = max(a, c);
        right = min(b, d);
        return true;
    } else {
        left.x = right.x = -det(m.c, m.b, n.c, n.b) / zn;
        left.y = right.y = -det(m.a, m.c, n.a, n.c) / zn;
        return betw(a.x, b.x, left.x) && betw(a.y, b.y, left.y)
            && betw(c.x, d.x, left.x) && betw(c.y, d.y, left.y);
    }
}

void tangents (pt c, double r1, double r2, vector<line> & ans)
{
    double r = r2 - r1;
    double z = sqrt(c.x) + sqrt(c.y);
    double d = z - sqrt(r);
    if (d < -EPS) return;
    d = sqrt (abs (d));
    line l;
    l.a = (c.x * r + c.y * d) / z;
    l.b = (c.y * r - c.x * d) / z;
    l.c = r1;
    ans.push_back (l);
}

vector<line> tangents (circle a, circle b) {
    vector<line> ans;
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents (b-a, a.r*i, b.r*j, ans);
    for (size_t i=0; i<ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    return ans;
}

class pointLocationInPolygon{

```

```

    bool lexComp(const pt & l, const pt & r){
        return l.x < r.x || (l.x == r.x && l.y < r.y);
    }
    int sgn(ftype val){
        return val > 0 ? 1 : (val == 0 ? 0 : -1);
    }
    vector<pt> seq;
    int n;
    pt translate;
    bool pointInTriangle(pt a, pt b, pt c, pt point){
        ftype s1 = abs(a.cross(b, c));
        ftype s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs(point.cross(c, a));
        return s1 == s2;
    }
public:
    pointLocationInPolygon(){
    }
    pointLocationInPolygon(vector<pt> & points){
        prepare(points);
    }
    void prepare(vector<pt> & points){
        seq.clear();
        n = points.size();
        int pos = 0;
        for(int i = 1; i < n; i++){
            if(lexComp(points[i], points[pos]))
                pos = i;
        }
        translate.x=points[pos].x;
        translate.y=points[pos].y;
        rotate(points.begin(), points.begin() + pos, points.end());
        n--;
        seq.resize(n);
        for(int i = 0; i < n; i++)
            seq[i] = points[i + 1] - points[0];
    }
    bool pointInConvexPolygon(pt point){
        point.x-=translate.x;
        point.y-=translate.y;
        if(seq[0].cross(point) != 0 && sgn(seq[0].cross(point))
            != sgn(seq[0].cross(seq[n - 1])))
            return false;
        if(seq[n - 1].cross(point) != 0 && sgn(seq[n - 1].cross(point))
            != sgn(seq[n - 1].cross(seq[0])))
            return false;
        if(seq[0].cross(point) == 0)
            return seq[0].sqrLen() >= point.sqrLen();
        int l = 0, r = n - 1;
        while(r - l > 1){
            int mid = (l + r)/2;
            int pos = mid;
            if(seq[pos].cross(point) >= 0) l = mid;
            else r = mid;
        }
        int pos = l;
        return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
    }
    ~pointLocationInPolygon(){
        seq.clear();
    }
}

class Minkowski{
    static void reorder_polygon(vector<pt> & P){
        size_t pos = 0;
        for(size_t i = 1; i < P.size(); i++){
            if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x < P[pos].x))
                pos = i;
        }
        rotate(P.begin(), P.begin() + pos, P.end());
    }
public:
    static vector<pt> minkowski(vector<pt> P, vector<pt> Q){
        // the first vertex must be the lowest
        reorder_polygon(P);
        reorder_polygon(Q);
    }

```

```

        // we must ensure cyclic indexing
        P.push_back(P[0]);
        P.push_back(P[1]);
        Q.push_back(Q[0]);
        Q.push_back(Q[1]);
        // main part
        vector<pt> result;
        size_t i = 0, j = 0;
        while(i < P.size() - 2 || j < Q.size() - 2){
            result.push_back(P[i] + Q[j]);
            auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
            if(cross >= 0)
                ++i;
            if(cross <= 0)
                ++j;
        }
        return result;
    }
};

vector<pt> circle_line_intersections(circle cir, line l){
    double r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x + l.b*cir.y;
    vector<pt> ans;
    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS);
    else if (abs (c*c - r*r*(a*a+b*b)) < EPS){
        pt p;
        p.x=x0;
        p.y=y0;
        ans.push_back(p);
    }
    else{
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax, ay, bx, by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        pt p;
        p.x = ax;
        p.y = ay;
        ans.push_back(p);
        p.x = bx;
        p.y = by;
        ans.push_back(p);
    }
    for(int i=0; i<ans.size(); i++){
        ans[i] = ans[i] + cir;
    }
    return ans;
}

double circle_polygon_intersection(circle c, vector<pt> &V){
    int n = V.size();
    double ans = 0;
    for(int i=0; i<n; i++){
        line l(V[i], V[(i+1)%n]);
        vector<pt> lpts = circle_line_intersections(c, l);
        int sz=lpts.size();
        for(int j=sz-1; j>=0; j--){
            if(!is_point_on_seg(V[i], V[(i+1)%n], lpts[j])){
                swap(lpts.back(), lpts[j]);
                lpts.pop_back();
            }
        }
        lpts.push_back(V[i]);
        lpts.push_back(V[(i+1)%n]);
        sort(lpts.begin(), lpts.end());
        sz=lpts.size();
        if(V[(i+1)%n]<V[i])
            reverse(lpts.begin(), lpts.end());
        for(int j=1; j<sz; j++){
            if(insideCircle(c, lpts[j-1])
                && insideCircle(c, lpts[j]))
                ans = ans + area(lpts[j-1], lpts[j], c);
            else{
                double ang = angle(lpts[j-1], c, lpts[j]);
                double aa = c.r*c.r*ang/2;
                if(ccw(lpts[j-1], lpts[j], c))

```

```

        ans = ans+aa;
    else
        ans = ans-aa;
    }
}
ans = abs(ans);
return ans;
}

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }
    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw[st.size()-2], st.back(),
            a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }
    a = st;
    int m = a.size();
    for(int i = 0; i<m-1-i; i++){
        swap(a[i], a[m-1-i]);
    }
}

double mindist;
pair<int, pair<int, int>> best_pair;
void upd_ans(const pt & a, const pt & b, const pt & c){
    double distC = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
    double distA = sqrt((c.x - b.x)*(c.x - b.x) + (c.y - b.y)*(c.y - b.y));
    double distB = sqrt((a.x - c.x)*(a.x - c.x) + (a.y - c.y)*(a.y - c.y));
    if (distA + distB + distC < mindist){
        mindist = distA + distB + distC;
        best_pair = make_pair(a.id, make_pair(b.id, c.id));
    }
}

vector<pt> t;
//Min possible triplet distance
void rec(int l, int r){
    if (r - l <= 3 && r - l >= 2){
        for (int i = l; i < r; ++i){
            for (int j = i + 1; j < r; ++j){
                for (int k = j + 1; k < r; ++k){
                    upd_ans(a[i], a[j], a[k]);
                }
            }
        }
    }
    sort(a.begin() + l, a.begin() + r, cmp_y());
    return;
}

int m = (l + r) >> 1;
int midx = a[m-1].x;
/*
 * Got WA in a team contest
 * for putting midx = a[m].x;
 * Don't know why. Maybe due to
 * floating point numbers.
 */
rec(l, m);
rec(m, r);
merge(a.begin() + l, a.begin() + m, a.begin() + m,
    a.begin() + r, t.begin(), cmp_y());
copy(t.begin(), t.begin() + r - l, a.begin() + l);

```

```

int tsz = 0;
for (int i = l; i < r; ++i){
    if (abs(a[i].x - midx) < mindist/2){
        for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y <
            mindist/2; --j){
            if (i+1<r) upd_ans(a[i], a[i+1], t[j]);
            if (j>0) upd_ans(a[i], t[j-1], t[j]);
        }
        t[tsz++] = a[i];
    }
}
}
}

```

## 5.8 intersecting-segments-pair

```

const double EPS = 1E-9;
struct pt {
    double x, y;
};
struct seg {
    pt p, q;
    int id;
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};

bool intersectId(double l1, double r1, double l2, double r2) {
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}

int vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}

bool intersect(const seg& a, const seg& b){
    return intersectId(a.p.x, a.q.x, b.p.x, b.q.x) &&
        intersectId(a.p.y, a.q.y, b.p.y, b.q.y) &&
        vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
        vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}

bool operator<(const seg& a, const seg& b){
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}

struct event {
    double x;
    int tp, id;
    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};

set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}

set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}

pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());
    s.clear();
    where.resize(a.size());
}

```

```

for (size_t i = 0; i < e.size(); ++i) {
    int id = e[i].id;
    if (e[i].tp == +1) {
        set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
        if (nxt != s.end() && intersect(*nxt, a[id]))
            return make_pair(nxt->id, id);
        if (prv != s.end() && intersect(*prv, a[id]))
            return make_pair(prv->id, id);
        where[id] = s.insert(nxt, a[id]);
    } else {
        set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
        if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
            return make_pair(prv->id, nxt->id);
        s.erase(where[id]);
    }
    return make_pair(-1, -1);
}
}

```

## 5.9 radiant-geo

```

typedef double Tf; typedef double Ti;
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) {return abs(x)<EPS? 0 : (x<0?-1:1);}
struct PT {
    Ti x, y;
    bool operator == (const PT& u) const {
        return dcmp(x-u.x)==0 && dcmp(y-u.y)==0;
    }
    bool operator != (const PT& u) const {
        return !(*this == u);
    }
    friend istream &operator >> (istream &is, PT &p) {
        return is >> p.x >> p.y;
    }
    friend ostream &operator << (ostream &os, const PT &p) {return os<<p.x<<" "<<p.y; }
};

Ti dot(PT a, PT b) { return a.x*b.x + a.y*b.y; }
Ti cross(PT a, PT b) { return a.x*b.y - a.y*b.x; }
Tf length(PT a) { return sqrt(dot(a, a)); }
Ti sqLength(PT a) { return dot(a, a); }
Tf distance(PT a, PT b) {return length(a-b);}
Tf angle(PT u) { return atan2(u.y, u.x); }
Tf angleBetween(PT a, PT b) { //in range [-PI, PI]
    Tf ans = angle(b) - angle(a);
    return ans <= -PI ? ans + 2*PI :
        (ans > PI ? ans - 2*PI : ans);
}

PT rotate(PT a, Tf rad) {
    static_assert(is_same<Tf, Ti>::value);
    return PT(a.x * cos(rad) - a.y * sin(rad),
        a.x * sin(rad) + a.y * cos(rad));
}

// Rotate(a, rad) where cos(rad)=co, sin(rad)=si
PT rotatePrecise(PT a, Tf co, Tf si) {
    static_assert(is_same<Tf, Ti>::value);
    return PT(a.x*co - a.y*si, a.y*co + a.x*si);
}

PT rotate90(PT a) { return PT(-a.y, a.x); }
PT scale(PT a, Tf s) {
    static_assert(is_same<Tf, Ti>::value);
    return a / length(a) * s;
}

PT normal(PT a) {
    static_assert(is_same<Tf, Ti>::value);
    Tf l = length(a); return PT(-a.y / l, a.x / l);
}

// returns 1/0/-1 if c is left/on/right of ab
int orient(PT a, PT b, PT c) {
    return dcmp(cross(b - a, c - a));
}

//sort(v.begin(), v.end(), polarComp(0, dir))
struct polarComp {
    PT O, dir;
    polarComp(PT O = PT(0, 0), PT dir = PT(1, 0))
        : O(O), dir(dir) {}
    bool half(PT p) {

```

```

    return dcmp(cross(dir, p)) < 0 ||
    (dcmp(cross(dir, p)) == 0 && dcmp(dot(dir, p)) > 0);
}
bool operator()(PT p, PT q) {
    return make_tuple(half(p-0), 0) <
    make_tuple(half(q-0), cross(p-0, q-0));
}
};
struct Segment {
    PT a, b;
    Segment() {}
    Segment(PT aa, PT bb) : a(aa), b(bb) {}
};
typedef Segment Line;
struct Circle {
    PT o; Tf r;
    Circle(PT o = PT(0, 0), Tf r = 0) : o(o), r(r) {}
    bool contains(PT p) {
        return dcmp(sqrLength(p - o) - r * r) <= 0;
    }
    PT point(Tf rad) {
        static_assert(is_same<Tf, Ti>::value);
        return PT(o.x + cos(rad)*r, o.y + sin(rad)*r);
    }
    Tf area(Tf rad = PI + PI) { return rad * r * r / 2; }
    Tf sector(Tf alpha) {
        return r * r * 0.5 * (alpha - sin(alpha));
    }
};
#####
namespace Linear {
bool onSegment(PT p, Segment s) { ///Is p on S?
    return dcmp(cross(s.a - p, s.b - p)) == 0 &&
    dcmp(dot(s.a - p, s.b - p)) <= 0;
}
bool segmentsIntersect(Segment p, Segment q) {
    if(onSegment(p.a, q) || onSegment(p.b, q)) return 1;
    if(onSegment(q.a, p) || onSegment(q.b, p)) return 1;
    Ti c1 = cross(p.b - p.a, q.a - p.a);
    Ti c2 = cross(p.b - p.a, q.b - p.a);
    Ti c3 = cross(q.b - q.a, p.a - q.a);
    Ti c4 = cross(q.b - q.a, p.b - q.a);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
bool linesParallel(Line p, Line q) {
    return dcmp(cross(p.b - p.a, q.b - q.a)) == 0;
}
///returns if lines (p, p+v) && (q, q+w) intersect
bool lineLineIntersect(PT p, PT v, PT q, PT w, PT& o) {
    static_assert(is_same<Tf, Ti>::value);
    if(dcmp(cross(v, w)) == 0) return false;
    PT u = p - q; o = p + v * (cross(w, u) / cross(v, w));
    return true;
}
bool lineLineIntersect(Line p, Line q, PT& o) {
    return lineLineIntersect(p.a, p.b - p.a, q.a,
    q.b - q.a, o);
}
Tf distancePointLine(PT p, Line l) {
    return abs(cross(l.b-l.a, p-l.a)/length(l.b-l.a));
}
Tf distancePointSegment(PT p, Segment s) {
    if(s.a == s.b) return length(p - s.a);
    PT v1 = s.b - s.a, v2 = p - s.a, v3 = p - s.b;
    if(dcmp(dot(v1, v2)) < 0) return length(v2);
    else if(dcmp(dot(v1, v3)) > 0) return length(v3);
    else return abs(cross(v1, v2) / length(v1));
}
Tf distanceSegmentSegment(Segment p, Segment q) {
    if(segmentsIntersect(p, q)) return 0;
    Tf ans = distancePointSegment(p.a, q);
    ans = min(ans, distancePointSegment(p.b, q));
    ans = min(ans, distancePointSegment(q.a, p));
    ans = min(ans, distancePointSegment(q.b, p));
    return ans;
}
PT projectPointLine(PT p, Line l) {
    static_assert(is_same<Tf, Ti>::value);
    PT v = l.b - l.a;
    return l.a + v * ((Tf) dot(v, p-l.a)/dot(v, v));
}

```

```

} } // namespace Linear
#####
typedef vector<PT> Polygon;
namespace Polygonal {
/// cannot be all collinear
Polygon RemoveCollinear(const Polygon& poly) {
    Polygon ret;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        PT a = poly[i];
        PT b = poly[(i + 1) % n];
        PT c = poly[(i + 2) % n];
        if(dcmp(cross(b-a, c-b)) != 0 && (ret.empty() ||
        b != ret.back())) ret.push_back(b);
    }
    return ret;
}
Tf signedPolygonArea(const Polygon& p);
/// returns inside = -1, on = 0, outside = 1
int pointInPolygon(const Polygon& p, PT o);
/// returns (longest segment, total length)
pair<Tf, Tf> linePolygonIntersection(Line l,
    const Polygon& p) {
    using Linear::lineLineIntersect;
    int n = p.size(); vector<pair<Tf, int>> ev;
    for(int i=0; i<n; ++i) {
        PT a = p[i], b = p[(i+1)%n], z = p[(i-1+n)%n];
        int ora=orient(l.a,l.b,a), orb =
        orient(l.a,l.b,b), orz=orient(l.a,l.b,z);
        if(!ora) {
            Tf d = dot(a - l.a, l.b - l.a);
            if(orz && orb) {
                if(orz != orb) ev.emplace_back(d, 0);
                ///else // PT Touch
            } else if(orz) ev.emplace_back(d, orz);
            else if(orb) ev.emplace_back(d, orb);
        }
        else if(ora == -orb) {
            PT ins;
            lineLineIntersect(l, Line(a, b), ins);
            ev.emplace_back(dot(ins-l.a, l.b-l.a), 0);
        }
    }
    sort(ev.begin(), ev.end());
    Tf ans = 0, len = 0, last = 0, tot = 0;
    bool active = false; int sign = 0;
    for(auto &qq : ev) {
        int tp = qq.second;
        Tf d = qq.first; ///current Seg is (last, d)
        if(sign) {
            len+=d-last; tot+=d-last; ans=max(ans,len);
            if(tp != sign) active = !active;
            sign = 0;
        }
        else {
            if(active) { ///Strictly Inside
                len+=d-last; tot+=d-last; ans=max(ans,len);
            }
            if(tp == 0) active=!active; else sign = tp;
        }
        last = d; if(!active) len = 0;
    }
    ans /= length(l.b-l.a); tot /= length(l.b-l.a);
    return {ans, tot};
} } // namespace Polygonal
#####
namespace Convex {
///[min area, min perimeter] rectangle containing p
pair<Tf, Tf> rotatingCalipersBBBox(const Polygon& p) {
    using Linear::distancePointLine;
    static_assert(is_same<Tf, Ti>::value);
    int n = p.size(); int l = 1, r = 1, j = 1;
    Tf area = 1e100; Tf perimeter = 1e100;
    for(int i = 0; i < n; i++) {
        PT v = (p[(i+1)%n]-p[i])/length(p[(i+1)%n]-p[i]);
        while(dcmp(dot(v, p[r%n]-p[i]) -
        dot(v, p[(r+1)%n]-p[i])) < 0) r++;
        while(j < r || dcmp(cross(v, p[j%n]-p[i]) -

```

```

        cross(v, p[(j+1)%n]-p[i])) < 0) j++;
        while(l < j || dcmp(dot(v, p[l%n]-p[i]) -
        dot(v, p[(l+1)%n]-p[i])) > 0) l++;
        Tf w = dot(v, p[r%n]-p[i]) - dot(v, p[l%n]-p[i]);
        Tf h = distancePointLine(p[j%n],
        Line(p[i], p[(i+1)%n]));
        area = min(area, w * h);
        perimeter = min(perimeter, 2 * w + 2 * h);
    }
    return make_pair(area, perimeter);
}
/// returns the left half of u on left on ray ab
Polygon cutPolygon(Polygon u, PT a, PT b) {
    using Linear::lineLineIntersect;
    using Linear::onSegment;
    Polygon ret; int n = u.size();
    for(int i = 0; i < n; i++) {
        PT c = u[i], d = u[(i + 1) % n];
        if(dcmp(cross(b-a, c-a)) > 0) ret.push_back(c);
        if(dcmp(cross(b-a, d-c)) != 0) {
            PT t; lineLineIntersect(a, b-a, c, d-c, t);
            if(onSegment(t, Segment(c, d))) ret.push_back(t);
        }
    }
    return ret;
}
bool pointInTriangle(PT a, PT b, PT c, PT p) {
    return dcmp(cross(b - a, p - a)) >= 0
    && dcmp(cross(c - b, p - b)) >= 0
    && dcmp(cross(a - c, p - c)) >= 0;
}
int pointInConvexPolygon(const Polygon& p, PT p);
/// most extreme Point in the direction u
int extremePoint(const Polygon& poly, PT u) {
    int n = (int) poly.size();
    int a = 0, b = n;
    while(b - a > 1) {
        int c = (a + b) / 2;
        if(dcmp(dot(poly[c]-poly[(c+1)%n], u)) >= 0 &&
        dcmp(dot(poly[c]-poly[(c-1)%n], u)) >= 0) {
            return c;
        }
        bool a_up=dcmp(dot(poly[(a+1)%n]-poly[a], u)) >= 0;
        bool c_up=dcmp(dot(poly[(c+1)%n]-poly[c], u)) >= 0;
        bool a_above_c=dcmp(dot(poly[a]-poly[c], u)) >= 0;
        if(a_up && !c_up) b = c;
        else if(!a_up && c_up) a = c;
        else if(a_up && c_up) {
            if(a_above_c) b = c; else a = c;
        }
        else {
            if(!a_above_c) b = c; else a = c;
        }
    }
    if(dcmp(dot(poly[a]-poly[(a+1)%n], u)) > 0 &&
    dcmp(dot(poly[a]-poly[(a-1+n)%n], u)) > 0)
        return a;
    return b % n;
}
/// return list of segs of p that touch/intersect l
/// the i'th segment is [p[i], p[(i + 1)%p]]
/// #1 If a side is collinear, only that returned
/// #2 If l goes through p[i], ith segment is added
vector<int> lineConvexPolyIntersection(
    const Polygon& p, Line l) {
    assert((int) p.size() >= 3); assert(l.a != l.b);
    int n = p.size(); vector<int> ret;
    PT v = l.b - l.a;
    int lf = extremePoint(p, rotate90(v));
    int rt = extremePoint(p, rotate90(v) * Ti(-1));
    int olf = orient(l.a, l.b, p[lf]);
    int ort = orient(l.a, l.b, p[rt]);
    if(!olf || !ort) {
        int idx = (!olf ? lf : rt);
        if(orient(l.a, l.b, p[(idx - 1 + n) % n]) == 0)
            ret.push_back((idx - 1 + n) % n);
        else ret.push_back(idx);
        return ret;
    }
    if(olf == ort) return ret;

```



```

for(int i=0; i<2; ++i) {
    int lo = i ? rt : lf, hi = i ? lf : rt;
    int olo = i ? ort : olf;
    while(true) {
        int gap = (hi - lo + n) % n;
        if(gap < 2) break;
        int mid = (lo + gap / 2) % n;
        int omid = orient(1.a, 1.b, p[mid]);
        if(!omid) {lo = mid; break;}
        if(omid == olo) lo = mid;
        else hi = mid;
    } ret.push_back(lo);
} return ret;
}

// [ACW, CW] tangent pair from an external point
constexpr int CW = -1, ACW = 1;
bool isGood(PT u, PT v, PT Q, int dir) {
    return orient(Q, u, v) != -dir;
}
PT better(PT u, PT v, PT Q, int dir) {
    return orient(Q, u, v) == dir ? u : v;
}
PT pointPolyTangent(const Polygon &pt, PT Q,
                    int dir, int lo, int hi) {
    while(hi - lo > 1) {
        int mid = (lo + hi) / 2;
        bool pvs = isGood(pt[mid], pt[mid-1], Q, dir);
        bool nxt = isGood(pt[mid], pt[mid+1], Q, dir);
        if(pvs && nxt) return pt[mid];
        if(!(pvs || nxt)) {
            PT p1 = pointPolyTangent(pt, Q, dir, mid+1, hi);
            PT p2 = pointPolyTangent(pt, Q, dir, lo, mid-1);
            return better(p1, p2, Q, dir);
        }
        if(!pvs) {
            if(orient(Q, pt[mid], pt[lo]) == dir) hi = mid-1;
            else if(better(pt[lo], pt[hi], Q, dir) == pt[lo])
                hi = mid - 1; else lo = mid + 1;
        }
        if(!nxt) {
            if(orient(Q, pt[mid], pt[lo]) == dir) lo = mid+1;
            else if(better(pt[lo], pt[hi], Q, dir) == pt[lo])
                hi = mid - 1; else lo = mid + 1;
        }
    }
    PT ret = pt[lo];
    for(int i = lo + 1; i <= hi; i++)
        ret = better(ret, pt[i], Q, dir);
    return ret;
}

// [ACW, CW] Tangent
pair<PT, PT> pointPolyTangents(
    const Polygon &pt, PT Q) {
    int n = pt.size();
    PT acw_tan = pointPolyTangent(pt, Q, ACW, 0, n-1);
    PT cw_tan = pointPolyTangent(pt, Q, CW, 0, n-1);
    return make_pair(acw_tan, cw_tan);
}

namespace Circular {
// returns intersections in order of ray (1.a, 1.b)
vector<PT> circleLineIntersection(Circle c, Line l) {
    static_assert(is_same<Tf, Ti>::value);
    vector<PT> ret;
    PT b = 1.b - 1.a, a = 1.a - c.o;
    Tf A = dot(b, b), B = dot(a, b);
    Tf C = dot(a, a) - c.r * c.r, D = B*B - A*C;
    if(D < -EPS) return ret;
    ret.push_back(1.a + b * (-B - sqrt(D + EPS)) / A);
    if(D > EPS)
        ret.push_back(1.a + b * (-B + sqrt(D)) / A);
    return ret;
}

// circle(c.o, c.r) x triangle(c.o, s.a, s.b) (ccw)
Tf circleTriInterArea(Circle c, Segment s) {
    using Linear::distancePointSegment;
    Tf OA = length(c.o - s.a), OB = length(c.o - s.b);
    if(dcmp(distancePointSegment(c.o, s) - c.r) >= 0)
        return angleBetween(s.a - c.o, s.b - c.o) * c.r * c.r / 2;
    if(dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
        return cross(c.o - s.b, s.a - s.b) / 2.0;
}

vector<PT> Sect = circleLineIntersection(c, s);
return circleTriInterArea(c, Segment(s.a, Sect[0]))
    + circleTriInterArea(c, Segment(Sect[0], Sect[1]))
    + circleTriInterArea(c, Segment(Sect[1], s.b));
}

Tf circlePolyIntersectionArea(Circle c, Polygon p);
// locates circle c2 relative to c1: intersect = 0
// inside = -2, inside touch = -1,
// outside touch = 1, outside = 2
int circleCirclePosition(Circle c1, Circle c2) {
    Tf d = length(c1.o - c2.o);
    int in = dcmp(d - abs(c1.r - c2.r)),
        ex = dcmp(d + (c1.r + c2.r));
    return in<0?-2:in==0?-1:ex==0?1:ex>0?2:0;
}

vector<PT> circleCircleInter(Circle c1, Circle c2) {
    static_assert(is_same<Tf, Ti>::value);
    vector<PT> ret;
    Tf d = length(c1.o - c2.o);
    if(dcmp(d) == 0) return ret;
    if(dcmp(c1.r + c2.r - d) < 0) return ret;
    if(dcmp(abs(c1.r - c2.r) - d) > 0) return ret;
    PT v = c2.o - c1.o;
    Tf co = (c1.r * c1.r + sqLength(v) - c2.r*c2.r)
        / (2 * c1.r * length(v));
    Tf si = sqrt(abs(1.0 - co * co));
    PT p1 = scale(rotatePrecise(v, co, -si), c1.r) + c1.o;
    PT p2 = scale(rotatePrecise(v, co, si), c1.r) + c1.o;
    ret.push_back(p1);
    if(p1 != p2) ret.push_back(p2); return ret;
}

Tf circleCircleInterArea(Circle c1, Circle c2) {
    PT AB = c2.o - c1.o; Tf d = length(AB);
    if(d >= c1.r + c2.r) return 0;
    if(d + c1.r <= c2.r) return PI * c1.r * c1.r;
    if(d + c2.r <= c1.r) return PI * c2.r * c2.r;
    Tf alpha1 = acos((c1.r*c1.r + d*d - c2.r*c2.r)
        / (2.0 * c1.r * d));
    Tf alpha2 = acos((c2.r*c2.r + d*d - c1.r*c1.r)
        / (2.0 * c2.r * d));
    return c1.sector(2*alpha1) + c2.sector(2*alpha2);
}

// returns tangents from a point p to circle c
vector<PT> pointCircleTangents(PT p, Circle c) {
    static_assert(is_same<Tf, Ti>::value);
    vector<PT> ret; PT u = c.o - p; Tf d = length(u);
    if(d < c.r) return ret;
    else if(dcmp(d - c.r) == 0) {
        ret = { rotate(u, PI / 2) };
    }
    else {
        Tf ang = asin(c.r / d);
        ret = { rotate(u, -ang), rotate(u, ang) };
    }
    return ret;
}

// returns points on tangents that touches circle c
vector<PT> pointCircleTangencyPoints(PT p, Circle c) {
    static_assert(is_same<Tf, Ti>::value);
    PT u = p - c.o; Tf d = length(u);
    if(d < c.r) return {};
    else if(dcmp(d - c.r) == 0) return {c.o + u};
    else {
        Tf ang = acos(c.r / d); u = u/length(u) * c.r;
        return {c.o + rotate(u, -ang), c.o + rotate(u, ang)};
    }
}

// finds a, b st a[i] on c1, b[i] on c2, Segment
// a[i], b[i] touches c1, c2. if c1, c2 touch at x
// (x, x) is also returned, -1 returned if c1 = c2
int circleCircleTangencyPoints(Circle c1, Circle c2,
    vector<PT> &a, vector<PT> &b) {
    a.clear(), b.clear(); int cnt = 0;
    if(dcmp(c1.r - c2.r) < 0) {swap(c1, c2); swap(a, b);}
    Tf d2 = sqLength(c1.o - c2.o);
    Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;
    if(dcmp(d2 - rdif * rdif) < 0) return 0;
    if(dcmp(d2) == 0 && dcmp(c1.r - c2.r) == 0) return -1;
    Tf base = angle(c2.o - c1.o);
    if(dcmp(d2 - rdif * rdif) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(base));
        cnt++; return cnt;
    }
    Tf ang = acos((c1.r - c2.r) / sqrt(d2));
    a.push_back(c1.point(base + ang));
    b.push_back(c2.point(base + ang)); cnt++;
    a.push_back(c1.point(base - ang));
    b.push_back(c2.point(base - ang)); cnt++;
    if(dcmp(d2 - rsum * rsum) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(PI + base)); cnt++;
    }
    else if(dcmp(d2 - rsum * rsum) > 0) {
        Tf ang = acos((c1.r + c2.r) / sqrt(d2));
        a.push_back(c1.point(base + ang));
        b.push_back(c2.point(PI + base + ang)); cnt++;
        a.push_back(c1.point(base - ang));
        b.push_back(c2.point(PI + base - ang)); cnt++;
    }
    return cnt;
}

// namespace Circular
// returns false if points are collinear
bool inCircle(PT a, PT b, PT c, Circle &p) {
    using Linear::distancePointLine;
    static_assert(is_same<Tf, Ti>::value);
    if(orient(a, b, c) == 0) return false;
    Tf u=length(b-c), v=length(c-a), w=length(a-b);
    p.o = (a * u + b * v + c * w) / (u + v + w);
    p.r = distancePointLine(p.o, Line(a, b));
    return true;
}

// set of points A(x, y) st PA : QA = rp : rq
Circle apolloniusCircle(PT P, PT Q, Tf rp, Tf rq) {
    static_assert(is_same<Tf, Ti>::value);
    rq *= rq; rp *= rp; Tf a=rq-rp; assert(dcmp(a));
    Tf g = (rq*P.x-rp*Q.x)/a, h = (rq*P.y-rp*Q.y)/a;
    Tf c = (rq*P.x*P.x - rp*Q.x*Q.x +
        rq*P.y*P.y - rp*Q.y*Q.y)/a;
    PT o(g, h); Tf R = sqrt(g * g + h * h - c);
    return Circle(o, R);
}

// returns false if points are collinear
bool circumCircle(PT a, PT b, PT c, Circle &p) {
    using Linear::lineLineIntersect;
    if(orient(a, b, c) == 0) return false;
    PT d = (a + b) / 2, e = (a + c) / 2;
    PT vd = rotate90(b - a), ve = rotate90(a - c);
    bool f = lineLineIntersect(d, vd, e, ve, p.o);
    if(f) p.r = length(a - p.o);
    return f;
}

// finds a circle that goes all of p, |p| <= 3.
Circle boundary(const vector<PT> &p) {
    Circle ret; int sz = p.size();
    if(sz == 0) ret.r = 0;
    else if(sz == 1) ret.o = p[0], ret.r = 0;
    else if(sz == 2) ret.o = (p[0] + p[1]) / 2,
        ret.r = length(p[0] - p[1]) / 2;
    else if(!circumCircle(p[0], p[1], p[2], ret))
        ret.r = 0;
    return ret;
}

// Min circle enclosing p[fr....n-1],
// with points in b on the boundary, |b| <= 3.
Circle welzl(const vector<PT> &p,
    int fr, vector<PT> &b) {
    if(fr >= (int) p.size() || b.size() == 3)
        return boundary(b);
    Circle c = welzl(p, fr + 1, b);
    if(!c.contains(p[fr])) {
        b.push_back(p[fr]); c = welzl(p, fr + 1, b);
        b.pop_back();
    }
    return c;
}

// MEC of p, using welzl's algo. amortized O(n).
Circle MEC(vector<PT> p) {

```

```

    random_shuffle(p.begin(), p.end());
    vector<PT> q; return welzl(p, 0, q);
} }
#####
// Given list of segments v, finds a pair (i, j) st
// v[i],v[j] intersects. If none, returns {-1, -1}
namespace IntersectingSegments {
struct Event {
    Tf x; int tp, id;
    bool operator < (const Event &p) const {
        if(dcmp(x-p.x)) return x<p.x; return tp>p.tp;
    }
};
pair<int, int> anyInters(const vector<Segment> &v){
    using Linear::segmentsIntersect;
    static_assert(is_same<Tf, Ti>::value);
    vector<Event> ev;
    for(int i=0; i<v.size(); i++) {
        ev.push_back({min(v[i].a.x, v[i].b.x), +1, i});
        ev.push_back({max(v[i].a.x, v[i].b.x), -1, i});
    }
    sort(ev.begin(), ev.end());
    auto comp = [&v] (int i, int j) {
        Segment p = v[i], q = v[j];
        Tf x=max(min(p.a.x,p.b.x), min(q.a.x, q.b.x));
        auto yvalSegment = [&x](const Line &s) {
            if(dcmp(s.a.x - s.b.x) == 0) return s.a.y;
            return s.a.y + (s.b.y - s.a.y)
                * (x - s.a.x) / (s.b.x - s.a.x);
        };
        return dcmp(yvalSegment(p)-yvalSegment(q))<0;
    };
    multiset<int, decltype(comp)> st(comp);
    typedef decltype(st)::iterator iter;
    auto prev = [&st](iter it) {
        return it == st.begin() ? st.end() : --it;
    };
    auto next = [&st](iter it) {
        return it == st.end() ? st.end() : ++it;
    };
    vector<iter> pos(v.size());
    for(auto &cur : ev) {
        int id = cur.id;
        if(cur.tp == 1) {
            iter nxt = st.lower_bound(id), pre=prev(nxt);
            if(pre != st.end() && segmentsIntersect(
                v[*pre], v[id])) return {*pre, id};
            if(nxt != st.end() && segmentsIntersect(
                v[*nxt], v[id])) return {*nxt, id};
            pos[id] = st.insert(nxt, id);
        }
        else {
            iter nxt=next(pos[id]), pre=prev(pos[id]);
            if(pre != st.end() && nxt != st.end() &&
                segmentsIntersect(v[*pre], v[*nxt]))
                return {*pre, *nxt};
            st.erase(pos[id]);
        }
    }
    return {-1, -1};
} }
#####

```

## 5.10 trianlge-ear-clipping

```

///O(n^3) v bad brute force implementation,implement better
    algorithm later
template<class T>
int area(pair<T,T>& p1,pair<T,T>& p2,pair<T,T>& p3){
    return
        (p1.first*p2.second+p2.first*p3.second+p3.first*p1.second
        -p1.second*p2.first-p2.second*p3.first-p3.second*p1.first);
}
template<class T>
bool inside(pair<T,T>& a,pair<T,T>& b,pair<T,T>& c,pair<T,T>&
    p)
{
    int ar=abs(area(a,b,c));
    int t=abs(area(a,b,p))+abs(area(b,c,p))+abs(area(c,a,p));
}

```

```

    return ar==t;
}
template<class T>
void triangulate(vector<pair<T,T> > p,vector<pair<T,T> >&out)
{
    int pindx=0;
    if((int)p.size()<=3)
    {
        out.resize(p.size());
        copy(p.begin(),p.end(),out.begin());
        return;
    }
    while(p.size()>3)
    {
        int n=(int)p.size();
        int i,j,k;
        for(i=0;i<n;i++)
        {
            j=i+1;
            k=i+2;
            j=j>n?j-n:j;
            k=k>n?k-n:k;
            if(area(p[i],p[j],p[k])<0)
            {
                continue;
            }
            bool chk=true;
            for(int l=0;l<n;l++)
            {
                if(l==i||l==j||l==k)
                    continue;
                if(inside(p[i],p[j],p[k],p[l]))
                {
                    chk=false;
                    break;
                }
            }
            if(chk)
                break;
        }
        out[pindx++]=p[i];
        out[pindx++]=p[j];
        out[pindx++]=p[k];
        p.erase(p.begin()+j);
    }
    for(auto e:p)
        out[pindx++]=e;
}

```

## 5.11 vertical-decomposition

```

typedef double dbl;
const dbl eps = 1e-9;
inline bool eq(dbl x, dbl y){
    return fabs(x - y) < eps;
}
inline bool lt(dbl x, dbl y){
    return x < y - eps;
}
inline bool gt(dbl x, dbl y){
    return x > y + eps;
}
inline bool le(dbl x, dbl y){
    return x < y + eps;
}
inline bool ge(dbl x, dbl y){
    return x > y - eps;
}
struct pt{
    dbl x, y;
    inline pt operator - (const pt & p)const{
        return pt{x - p.x, y - p.y};
    }
    inline pt operator + (const pt & p)const{
        return pt{x + p.x, y + p.y};
    }
    inline pt operator * (dbl a)const{
        return pt{x * a, y * a};
    }
    inline dbl cross(const pt & p)const{

```

```

        return x * p.y - y * p.x;
    }
    inline dbl dot(const pt & p)const{
        return x * p.x + y * p.y;
    }
    inline bool operator == (const pt & p)const{
        return eq(x, p.x) && eq(y, p.y);
    }
};
struct Line{
    pt p[2];
    Line(){
        Line(pt a, pt b):p{a, b}{}
    }
    pt vec()const{
        return p[1] - p[0];
    }
    pt& operator [] (size_t i){
        return p[i];
    }
};
inline bool lexComp(const pt & l, const pt & r){
    if(fabs(l.x - r.x) > eps){
        return l.x < r.x;
    }
    else return l.y < r.y;
}
vector<pt> interSegSeg(Line l1, Line l2){
    if(eq(l1.vec().cross(l2.vec()), 0)){
        if(!eq(l1.vec().cross(l2[0] - l1[0]), 0))
            return {};
        if(!lexComp(l1[0], l1[1]))
            swap(l1[0], l1[1]);
        if(!lexComp(l2[0], l2[1]))
            swap(l2[0], l2[1]);
        pt l = lexComp(l1[0], l2[0]) ? l2[0] : l1[0];
        pt r = lexComp(l1[1], l2[1]) ? l1[1] : l2[1];
        if(l == r)
            return {l};
        else return lexComp(l, r) ? vector<pt>{l, r} :
            vector<pt>{};
    }
    else{
        dbl s = (l2[0] - l1[0]).cross(l2.vec()) /
            l1.vec().cross(l2.vec());
        pt inter = l1[0] + l1.vec() * s;
        if(ge(s, 0) && le(s, 1) && le((l2[0] - inter).dot(l2[1]
            - inter), 0))
            return {inter};
        else
            return {};
    }
}
inline char get_segtype(Line segment, pt other_point){
    if(eq(segment[0].x, segment[1].x))
        return 0;
    if(!lexComp(segment[0], segment[1]))
        swap(segment[0], segment[1]);
    return (segment[1] - segment[0]).cross(other_point -
        segment[0]) > 0 ? 1 : -1;
}
dbl union_area(vector<tuple<pt, pt, pt> > triangles){
    vector<Line> segments(3 * triangles.size());
    vector<char> segtype(segments.size());
    for(size_t i = 0; i < triangles.size(); i++){
        pt a, b, c;
        tie(a, b, c) = triangles[i];
        segments[3 * i] = lexComp(a, b) ? Line(a, b) : Line(b, a);
        segtype[3 * i] = get_segtype(segments[3 * i], c);
        segments[3 * i + 1] = lexComp(b, c) ? Line(b, c) : Line(c, b);
        segtype[3 * i + 1] = get_segtype(segments[3 * i + 1], a);
        segments[3 * i + 2] = lexComp(c, a) ? Line(c, a) : Line(a, c);
        segtype[3 * i + 2] = get_segtype(segments[3 * i + 2], b);
    }
    vector<dbl> k(segments.size(), b(segments.size()));
    for(size_t i = 0; i < segments.size(); i++){
        if(segtype[i]){
            k[i] = (segments[i][1].y - segments[i][0].y) /

```

```

        (segments[i][1].x - segments[i][0].x);
        b[i] = segments[i][0].y - k[i] * segments[i][0].x;
    }
    dbl ans = 0;
    for(size_t i = 0; i < segments.size(); i++){
        if(!segtype[i])
            continue;
        dbl l = segments[i][0].x, r = segments[i][1].x;
        vector<pair<dbl, int>> evts;
        for(size_t j = 0; j < segments.size(); j++){
            if(!segtype[j] || i == j)
                continue;
            dbl l1 = segments[j][0].x, r1 = segments[j][1].x;
            if(ge(l1, r) || ge(l, r1))
                continue;
            dbl common_l = max(l, l1), common_r = min(r, r1);
            auto pts = interSegSeg(segments[i], segments[j]);
            if(pts.empty()){
                dbl y1 = k[j] * common_l + b[j];
                dbl y2 = k[i] * common_l + b[i];
                if(1e-10 < y1 - y2){
                    int evt_type = -segtype[i] * segtype[j];
                    evts.emplace_back(common_l, evt_type);
                    evts.emplace_back(common_r, -evt_type);
                }
            }
            else if(pts.size() == 1u){
                dbl y1 = k[i] * common_l + b[i], y2 = k[j] * common_l + b[j];
                int evt_type = -segtype[i] * segtype[j];
                if(1e-10 < y1 - y2){
                    evts.emplace_back(common_l, evt_type);
                    evts.emplace_back(common_r, -evt_type);
                }
            }
            y1 = k[i] * common_r + b[i], y2 = k[j] * common_r + b[j];
            if(1e-10 < y1 - y2){
                evts.emplace_back(pts[0].x, evt_type);
                evts.emplace_back(common_r, -evt_type);
            }
        }
        else{
            if(segtype[j] != segtype[i] || j > i){
                evts.emplace_back(common_l, -2);
                evts.emplace_back(common_r, 2);
            }
        }
    }
    evts.emplace_back(1, 0); sort(evts.begin(), evts.end());
    size_t j = 0; int balance = 0;
    while(j < evts.size()){
        size_t ptr = j;
        while(ptr < evts.size() && eq(evts[j].first, evts[ptr].first)){
            balance += evts[ptr].second;
            ++ptr;
        }
        if(!balance && !eq(evts[j].first, r)){
            dbl next_x = ptr == evts.size() ? r : evts[ptr].first;
            ans -= segtype[i] * (k[i] * (next_x + evts[j].first) + 2 * b[i]) * (next_x - evts[j].first);
        }
        j = ptr;
    }
    return ans/2;
}
pair<dbl,dbl> union_perimeter(vector<tuple<pt, pt, pt>> triangles){
    //Same as before
    pair<dbl,dbl> ans = make_pair(0,0);
    for(size_t i = 0; i < segments.size(); i++){
        //Same as before
        double dist=(segments[i][1].x-segments[i][0].x)
            *(segments[i][1].x-segments[i][0].x)
            +(segments[i][1].y-segments[i][0].y)
            *(segments[i][1].y-segments[i][0].y);

```

```

        dist=sqrt(dist);
        while(j < evts.size()){
            size_t ptr = j;
            while(ptr < evts.size() && eq(evts[j].first, evts[ptr].first)){
                balance += evts[ptr].second; ++ptr;
            }
            if(!balance && !eq(evts[j].first, r)){
                dbl next_x = ptr == evts.size() ? r : evts[ptr].first;
                ans.first += dist * (next_x - evts[j].first) / (r-1);
                if(eq(segments[i][1].y, segments[i][0].y))
                    ans.second += (next_x - evts[j].first);
            }
            j = ptr;
        }
    }
    return ans;
}

```

## 6 Graph

### 6.1 DMST

```

// tested on https://lightoj.com/problem/teleport
const int inf = 1e9;
struct edge {
    int u, v, w;
    edge() {}
    edge(int a, int b, int c) : u(a), v(b), w(c) {}
    bool operator < (const edge& o) const {
        if (u == o.u)
            if (v == o.v) return w < o.w;
            else return v < o.v;
        return u < o.u;
    }
};
int dmst(vector<edge> &edges, int root, int n) {
    int ans = 0;
    int cur_nodes = n;
    while (true) {
        vector<int> lo(cur_nodes, inf), pi(cur_nodes, inf);
        for (int i = 0; i < edges.size(); ++i) {
            int u = edges[i].u, v = edges[i].v, w = edges[i].w;
            if (w < lo[v] and u != v) {
                lo[v] = w;
                pi[v] = u;
            }
        }
        lo[root] = 0;
        for (int i = 0; i < lo.size(); ++i) {
            if (i == root) continue;
            if (lo[i] == inf) return -1;
        }
        int cur_id = 0;
        vector<int> id(cur_nodes, -1), mark(cur_nodes, -1);
        for (int i = 0; i < cur_nodes; ++i) {
            ans += lo[i];
            int u = i;
            while (u != root and id[u] < 0 and mark[u] != i) {
                mark[u] = i;
                u = pi[u];
            }
            if (u != root and id[u] < 0) { // Cycle
                for (int v = pi[u]; v != u; v = pi[v]) id[v] = cur_id;
                id[u] = cur_id++;
            }
        }
        if (cur_id == 0) break;
        for (int i = 0; i < cur_nodes; ++i)
            if (id[i] < 0) id[i] = cur_id++;
        for (int i = 0; i < edges.size(); ++i) {
            int u = edges[i].u, v = edges[i].v, w = edges[i].w;
            edges[i].u = id[u];
            edges[i].v = id[v];
            if (id[u] != id[v]) edges[i].w -= lo[v];
        }
        cur_nodes = cur_id;
    }
}

```

```

        root = id[root];
    }
    return ans;
}

```

### 6.2 Flow With Demands

**Finding an arbitrary flow** Consider flow networks, where we additionally require the flow of each edge to have a certain amount, i.e. we bound the flow from below by a **demand** function  $d(e)$ :

$$d(e) \leq f(e) \leq c(e)$$

So next each edge has a minimal flow value, that we have to pass along the edge.

We make the following changes in the network. We add a new source  $s'$  and a new sink  $t'$ , a new edge from the source  $s'$  to every other vertex, a new edge for every vertex to the sink  $t'$ , and one edge from  $t$  to  $s$ . Additionally we define the new capacity function  $c'$  as:

- $c'((s', v)) = \sum_{u \in V} d((u, v))$  for each edge  $(s', v)$ .
- $c'((v, t')) = \sum_{w \in V} d((v, w))$  for each edge  $(v, t')$ .
- $c'((u, v)) = c((u, v)) - d((u, v))$  for each edge  $(u, v)$  in the old network.
- $c'((t, s)) = \infty$

If the new network has a saturating flow (a flow where each edge outgoing from  $s'$  is completely filled, which is equivalent to every edge incoming to  $t'$  is completely filled), then the network with demands has a valid flow, and the actual flow can be easily reconstructed from the new network. Otherwise there doesn't exist a flow that satisfies all conditions. Since a saturating flow has to be a maximum flow, it can be found by any maximum flow algorithm.

**Minimal flow** Note that along the edge  $(t, s)$  (from the old sink to the old source) with the capacity  $\infty$  flows the entire flow of the corresponding old network. I.e. the capacity of this edge effects the flow value of the old network. By giving this edge a sufficient large capacity (i.e.  $\infty$ ), the flow of the old network is unlimited. By limiting this edge by smaller capacities, the flow value will decrease. However if we limit this edge by a too small value, than the network will not have a saturated solution, e.g. the corresponding solution for the original network will not satisfy the demand of the edges. Obviously here can use a binary search to find the lowest value with which all constraints are still satisfied. This gives the minimal flow of the original network.

### 6.3 LCA

```

int n; //beware n is declared global
int bparent[MAXN][LOG], depth[MAXN];
bool vis[MAXN];
void dfs(int a){
    vis[a]=true;
    for(auto v: g[a]){
        if(!vis[v]){
            bparent[v][0]=a;

```

```

        depth[v]=1+depth[a];
        dfs(v);}}
void build_ancestor(){
    dfs(1);
    for(int i=1;(1<i)<n;i++){
        for(int j=1;j<=n;j++){
            bparent[j][i]=bparent[bparent[j][i-1]][i-1];}
int pth_ancestor(int a,int p){
    for(int i=0;(1<i)<=p;i++){
        if((1<i)&p) a=bparent[a][i];
    return a;}
int lca(int u,int v){
    if(depth[v]>depth[u])
        v=pth_ancestor(v,depth[v]-depth[u]);
    if(depth[u]>depth[v])
        u=pth_ancestor(u,depth[u]-depth[v]);
    if(u==v) return u;
    for(int i=log2(n-1);i>=0;i--){
        if(bparent[u][i]!=bparent[v][i]){
            u=bparent[u][i];
            v=bparent[v][i];}}
    return bparent[u][0];}

```

## 6.4 articulation-vertex

```

int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}

```

## 6.5 bellman-ford

```

struct Edge {
    int a, b, cost;
};
int n, m;
vector<Edge> edges;
const int INF = 1000000000;
void solve(){
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }
}

```

```

    }
    }
}
if (x == -1) {
    cout << "No negative cycle found.";
} else {
    for (int i = 0; i < n; ++i)
        x = p[x];
    vector<int> cycle;
    for (int v = x; v = p[v]) {
        cycle.push_back(v);
        if (v == x && cycle.size() > 1)
            break;
    }
    reverse(cycle.begin(), cycle.end());
    cout << "Negative cycle: ";
    for (int v : cycle)
        cout << v << ' ';
    cout << endl;
}
}
}

```

## 6.6 bridge

```

const int vmax = 2e5+10, emax = 2e5+10;
namespace Bridge {///edge, nodes, comps 1 indexed
    vector<int> adj[vmax]; /// edge-id
    pair<int, int> edges[emax]; /// (u, v)
    bool isBridge[emax];
    int visited[vmax]; ///0-unvis,1-vising,2-vis
    int st[vmax], low[vmax], clk = 0, edgeId = 0;
    /// For bridge tree components
    int who[vmax], compId = 0;
    vector<int> stk;
    /// For extra end time calc
    int en[vmax];
    void dfs(int u, int parEdge) {
        visited[u] = 1; low[u] = st[u] = ++clk;
        stk.push_back(u);
        for (auto e : adj[u]) {
            if (e == parEdge) continue;
            int v=edges[e].first^edges[e].second^u;
            if (visited[v] == 1) {
                low[u] = min(low[u], st[v]);
            } else if(visited[v] == 0){
                dfs(v, e); low[u] = min(low[u], low[v]);
            }
        }
        visited[u] = 2;
        if(st[u] == low[u]){/// found
            ++compId; int cur;
            do{
                cur = stk.back(); stk.pop_back();
                who[cur] = compId;
            }while(cur != u);
            if(parEdge != -1){isBridge[parEdge] = true;}
        }
        en[u] = clk;
    }
    void clearAll(int n){
        for(int i = 0; i<=n; i++) {
            adj[i].clear(); visited[i] = st[i] = 0; }
        for(int i = 0; i<=edgeId; i++) isBridge[i]=0;
        clk = compId = edgeId = 0;
    }
    void findBridges(int n){
        for(int i = 1; i<=n; i++){
            if(visited[i] == 0) dfs(i, -1); }
    }
    bool isReplacable(int eid, int u, int v){
        if(!isBridge[eid]) return true;
        int a=edges[eid].first,b=edges[eid].second;
        if(st[a] > st[b]) swap(a, b);
        return (st[b] <= st[u] && st[u] <= en[b])
            != (st[b] <= st[v] && st[v] <= en[b]);
    }
    void addEdge(int u, int v){
        edgeId++; edges[edgeId] = {u, v};
        adj[u].emplace_back(edgeId);
    }
}

```

```
adj[v].emplace_back(edgeId);
}
```

## 6.7 edmond-blossom

```

/**Copied from https://codeforces.com/blog/entry/49402***/
/*
GETS:
V->number of vertices
E->number of edges
pair of vertices as edges (vertices are 1..V)
GIVES:
output of edmonds() is the maximum matching
match[i] is matched pair of i (-1 if there isn't a matched pair)
*/
const int M=500;
struct struct_edge
{
    int v;
    struct_edge* n;
};
typedef struct_edge* edge;
struct struct_edge pool[M*M*2];
edge top=pool,adj[M];
int V,E,match[M],gh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
void add_edge(int u,int v)
{
    top->v=v,top->n=adj[u],adj[u]=top++;
    top->v=u,top->n=adj[v],adj[v]=top++;
}
int LCA(int root,int u,int v)
{
    static bool inp[M];
    memset(inp,0,sizeof(inp));
    while(1)
    {
        inp[u=base[u]]=true;
        if (u==root) break;
        u=father[match[u]];
    }
    while(1)
    {
        if (inp[v=base[v]]) return v;
        else v=father[match[v]];
    }
}
void mark_blossom(int lca,int u)
{
    while (base[u]!=lca)
    {
        int v=match[u];
        inb[base[u]]=inb[base[v]]=true;
        u=father[v];
        if (base[u]!=lca) father[u]=v;
    }
}
void blossom_contraction(int s,int u,int v)
{
    int lca=LCA(s,u,v);
    memset(inb,0,sizeof(inb));
    mark_blossom(lca,u);
    mark_blossom(lca,v);
    if (base[u]!=lca)
        father[u]=v;
    if (base[v]!=lca)
        father[v]=u;
    for (int u=0; u<V; u++)
        if (inb[base[u]])
        {
            base[u]=lca;
            if (!inq[u])
                inq[q[++qt]=u]=true;
        }
}
int find_augmenting_path(int s)
{
}

```



```

memset(inq,0,sizeof(inq));
memset(father,-1,sizeof(father));
for (int i=0; i<V; i++) base[i]=i;
inq[q[qh=qt=0]=s]=true;
while (qh<=qt)
{
    int u=q[qh++];
    for (edge e=adj[u]; e; e=e->n)
    {
        int v=e->v;
        if (base[u] != base[v] && match[u] != v)
            if ((v==s) || (match[v] != -1 &&
                father[match[v]] != -1))
                blossom_contraction(s,u,v);
            else if (father[v] == -1)
            {
                father[v]=u;
                if (match[v] == -1)
                    return v;
                else if (!inq[match[v]])
                    inq[q[++qt]=match[v]]=true;
            }
    }
}
return -1;
}
int augment_path(int s,int t)
{
    int u=t,v,w;
    while (u!=-1)
    {
        v=father[u];
        w=match[v];
        match[v]=u;
        match[u]=v;
        u=w;
    }
    return t!=-1;
}
int edmonds()//Gives number of matchings
{
    int matchc=0;
    memset(match,-1,sizeof(match));
    for (int u=0; u<V; u++)
        if (match[u]==-1)
            matchc+=augment_path(u,find_augmenting_path(u));
    return matchc;
}
//To add edge add_edge(u-1,v-1);
ed[u-1][v-1]=ed[v-1][u-1]=true;

```

## 6.8 euler-path

```

class EulerGraph{
public:
    struct Edge{
        int other;
        int oi;
        bool vis;
        Edge(int other, int oi): other(other), oi(oi){
            vis=false;
        };
    };
    vector<vector<Edge>> adj;
    int* cnt;
    bool* sig;
    EulerGraph(int n){
        adj.resize(n);
        cnt=new int[n];
        sig=new bool[n];
        memset(cnt, 0, sizeof(int)*n);
        memset(sig, 0, n);
    }
    void addEdge(int u, int v){
        int un=adj[u].size(), vn=adj[v].size();
        adj[u].eb(v, vn), adj[v].eb(u, un);
    }
    deque<int> getCircuit(int src){
        deque<int> circuit;
        circuit.push_back(src);
        int ecnt=adj[src].size();
        while(ecnt){
            int v=circuit.back();
            sig[v]=true;

```

```

        if(cnt[v]==adj[v].size()){
            circuit.pop_back();
            circuit.push_front(circuit.back());
        }
        else if(adj[v][cnt[v]].vis){
            cnt[v]++;
            ecnt--;
        }
        else {
            int other=adj[v][cnt[v]].other;
            int oi=adj[v][cnt[v]].oi;
            adj[other][oi].vis=true;
            circuit.push_back(other);
            if(!sig[other]) ecnt+=adj[other].size();

            cnt[v]++;
            ecnt--;
        }
        return circuit;
    };
}

```

## 6.9 hopcraft-karp

```

/** Source: https://iq.opengenus.org/hopcroft-karp-algorithm/
**/
// A class to represent Bipartite graph for
// Hopcroft Karp implementation
class BGraph{
    // m and n are number of vertices on left
    // and right sides of Bipartite Graph
    int m, n;
    // adj[u] stores adjacents of left side
    // vertex 'u'. The value of u ranges from 1 to m.
    // 0 is used for dummy vertex
    std::list<int> *adj;
    // pointers for hopcroftKarp()
    int *pair_u, *pair_v, *dist;
public:
    BGraph(int m, int n); // Constructor
    void addEdge(int u, int v); // To add edge
    // Returns true if there is an augmenting path
    bool bfs();
    // Adds augmenting path if there is one beginning
    // with u
    bool dfs(int u);
    // Returns size of maximum matching
    int hopcroftKarpAlgorithm();
};
// Returns size of maximum matching
int BGraph::hopcroftKarpAlgorithm(){
    // pair_u[u] stores pair of u in matching on left side of
    // Bipartite Graph.
    // If u doesn't have any pair, then pair_u[u] is NIL
    pair_u = new int[m + 1];
    // pair_v[v] stores pair of v in matching on right side of
    // Bipartite Graph.
    // If v doesn't have any pair, then pair_u[v] is NIL
    pair_v = new int[n + 1];
    // dist[u] stores distance of left side vertices
    dist = new int[m + 1];
    // Initialize NIL as pair of all vertices
    for (int u = 0; u <= m; u++)
        pair_u[u] = NIL;
    for (int v = 0; v <= n; v++)
        pair_v[v] = NIL;
    // Initialize result
    int result = 0;
    // Keep updating the result while there is an
    // augmenting path possible.
    while (bfs()){
        // Find a free vertex to check for a matching
        for (int u = 1; u <= m; u++)
            // If current vertex is free and there is
            // an augmenting path from current vertex
            // then increment the result
            if (pair_u[u] == NIL && dfs(u))
                result++;
    }
    return result;
}
// Returns true if there is an augmenting path available, else
// returns false

```

```

bool BGraph::bfs(){
    std::queue<int> q; //an integer queue for bfs
    // First layer of vertices (set distance as 0)
    for (int u = 1; u <= m; u++){
        // If this is a free vertex, add it to queue
        if (pair_u[u] == NIL){
            // u is not matched so distance is 0
            dist[u] = 0;
            q.push(u);
        }
        // Else set distance as infinite so that this vertex is
        // considered next time for availability
        else
            dist[u] = INF;
    }
    // Initialize distance to NIL as infinite
    dist[NIL] = INF;
    // q is going to contain vertices of left side only.
    while (!q.empty()){
        // dequeue a vertex
        int u = q.front();
        q.pop();
        // If this node is not NIL and can provide a shorter
        // path to NIL then
        if (dist[u] < dist[NIL]){
            // Get all the adjacent vertices of the dequeued
            // vertex u
            std::list<int>::iterator it;
            for (it = adj[u].begin(); it != adj[u].end(); ++it){
                int v = *it;
                // If pair of v is not considered so far
                // i.e. (v, pair_v[v]) is not yet explored edge.
                if (dist[pair_v[v]] == INF){
                    // Consider the pair and push it to queue
                    dist[pair_v[v]] = dist[u] + 1;
                    q.push(pair_v[v]);
                }
            }
        }
        // If we could come back to NIL using alternating path of
        // distinct
        // vertices then there is an augmenting path available
        return (dist[NIL] != INF);
    }
    // Returns true if there is an augmenting path beginning with
    // free vertex u
    bool BGraph::dfs(int u){
        if (u != NIL){
            std::list<int>::iterator it;
            for (it = adj[u].begin(); it != adj[u].end(); ++it){
                // Adjacent vertex of u
                int v = *it;
                // Follow the distances set by BFS search
                if (dist[pair_v[v]] == dist[u] + 1){
                    // If dfs for pair of v also return true then
                    // if (dfs(pair_v[v]) == true){ // new matching
                    //     possible, store the matching
                    //     pair_v[v] = u;
                    //     pair_u[u] = v;
                    //     return true;
                    // }
                }
            }
            // If there is no augmenting path beginning with u then.
            dist[u] = INF;
            return false;
        }
        return true;
    }
    // Constructor for initialization
    BGraph::BGraph(int m, int n){
        this->m = m;
        this->n = n;
        adj = new std::list<int>[m + 1];
    }
    // function to add edge from u to v
    void BGraph::addEdge(int u, int v){
        adj[u].push_back(v); // Add v to us list.
    }
}

```

}

## 6.10 hungarian-algorithm

```

class HungarianAlgorithm{
    int N,inf,n,max_match;
    int *lx,*ly,*xy,*yx,*slack,*slackx,*prev;
    int **cost;
    bool *S,*T;
    void init_labels(){
        for(int x=0;x<n;x++) lx[x]=0;
        for(int y=0;y<n;y++) ly[y]=0;
        for (int x = 0; x < n; x++){
            for (int y = 0; y < n; y++){
                lx[x] = max(lx[x], cost[x][y]);
            }
        }
        void update_labels(){
            int x, y, delta = inf; //init delta as infinity
            for (y = 0; y < n; y++) //calculate delta using slack
                if (!T[y])
                    delta = min(delta, slack[y]);
            for (x = 0; x < n; x++) //update X labels
                if (S[x]) lx[x] -= delta;
            for (y = 0; y < n; y++) //update Y labels
                if (T[y]) ly[y] += delta;
            for (y = 0; y < n; y++) //update slack array
                if (!T[y])
                    slack[y] -= delta;
        }
        void add_to_tree(int x, int prevx)
        //x - current vertex,prevx - vertex from X before x in the
        //alternating path,
        //so we add edges (prevx, xy[x]), (xy[x], x){
            S[x] = true; //add x to S
            prev[x] = prevx; //we need this when augmenting
            for (int y = 0; y < n; y++) //update slacks, because we
            //add new vertex to S
                if (lx[x] + ly[y] - cost[x][y] < slack[y]){
                    slack[y] = lx[x] + ly[y] - cost[x][y];
                    slackx[y] = x;
                }
        }
        void augment() //main function of the algorithm{
            if (max_match == n) return; //check wether matching is
            //already perfect
            int x, y, root; //just counters and root vertex
            int q[N], wr = 0, rd = 0; //q - queue for bfs, wr,rd -
            //write and read
            //pos in queue
            //memset(S, false, sizeof(S)); //init set S
            for(int i=0;i<n;i++) S[i]=false;
            //memset(T, false, sizeof(T)); //init set T
            for(int i=0;i<n;i++) T[i]=false;
            //memset(prev, -1, sizeof(prev)); //init set prev - for
            //the alternating tree
            for(int i=0;i<n;i++) prev[i]=-1;
            for (x = 0; x < n; x++) //finding root of the tree{
                if (xy[x] == -1){
                    q[wr++] = root = x;
                    prev[x] = -2;
                    S[x] = true;
                    break;
                }
            }
            for (y = 0; y < n; y++) //initializing slack array{
                slack[y] = lx[root] + ly[y] - cost[root][y];
                slackx[y] = root;
            }
            while (true) //main cycle{
                while (rd < wr) //building tree with bfs cycle{
                    x = q[rd++]; //current vertex from X part
                    for (y = 0; y < n; y++) //iterate through all
                    //edges in equality graph{
                        if (cost[x][y] == lx[x] + ly[y] && !T[y]){
                            if (yx[y] == -1) break; //an exposed
                            //vertex in Y found, so
                            //augmenting path exists!
                            T[y] = true; //else just add y to T,
                }
            }
        }
    }

```

```

        q[wr++] = yx[y]; //add vertex yx[y],
        //which is matched
        //with y, to the queue
        add_to_tree(yx[y], x); //add edges (x,y)
        //and (y,yx[y]) to the tree
    }
    }
    if (y < n) break; //augmenting path found!
}
if (y < n) break; //augmenting path found!
update_labels(); //augmenting path not found, so
//improve labeling
wr = rd = 0;
for (y = 0; y < n; y++){
    //in this cycle we add edges that were added to
    //the equality graph as a
    //result of improving the labeling, we add edge (slackx[y], y)
    //to the tree if
    //and only if !T[y] && slack[y] == 0, also with this edge we
    //add another one
    //(y, yx[y]) or augment the matching, if y was exposed
    if (!T[y] && slack[y] == 0){
        if (yx[y] == -1) //exposed vertex in Y found
            - augmenting path exists!{
                x = slackx[y];
                break;
            }
        else{
            T[y] = true; //else just add y to T,
            if (!S[yx[y]]){
                q[wr++] = yx[y]; //add vertex yx[y],
                //which is matched with
                //y, to the queue
                add_to_tree(yx[y], slackx[y]); //and
                //add edges (x,y) and (y,
                //yx[y]) to the tree
            }
        }
    }
    if (y < n) break; //augmenting path found!
}
if (y < n) //we found augmenting path!{
    max_match++; //increment matching
    //in this cycle we inverse edges along augmenting path
    for (int cx = x, cy = y, ty; cx != -2; cx =
        prev[cx], cy = ty){
        ty = xy[cx];
        yx[cy] = cx;
        xy[cx] = cy;
    }
    augment(); //recall function, go to step 1 of the
    //algorithm
}
} //end of augment() function
public:
HungarianAlgorithm(int vv,int inf=1000000000){
    N=vv;
    n=N;
    max_match=0;
    this->inf=inf;
    lx=new int[N];
    ly=new int[N]; //labels of X and Y parts
    xy=new int[N]; //xy[x] - vertex that is matched with x,
    yx=new int[N]; //yx[y] - vertex that is matched with y
    slack=new int[N]; //as in the algorithm description
    slackx=new int[N]; //slackx[y] such a vertex, that
    //l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
    prev=new int[N]; //array for memorizing alternating paths
    S=new bool[N];
    T=new bool[N]; //sets S and T in algorithm
    cost=new int*[N]; //cost matrix
    for(int i=0; i<N; i++){
        cost[i]=new int[N];
    }
}
HungarianAlgorithm(){
    delete []lx;

```

```

    delete []ly;
    delete []xy;
    delete []yx;
    delete []slack;
    delete []slackx;
    delete []prev;
    delete []S;
    delete []T;
    int i;
    for(i=0; i<N; i++){
        delete []cost[i];
    }
    delete []cost;
}
void setCost(int i,int j,int c){
    cost[i][j]=c;
}
int* matching(bool first=true){
    int *ans;
    ans=new int[N];
    for(int i=0;i<N;i++){
        if(first) ans[i]=xy[i];
        else ans[i]=yx[i];
    }
    return ans;
}
int hungarian(){
    int ret = 0; //weight of the optimal matching
    max_match = 0; //number of vertices in current matching
    for(int x=0;x<n;x++) xy[x]=-1;
    for(int y=0;y<n;y++) yx[y]=-1;
    init_labels(); //step 0
    augment(); //steps 1-3
    for (int x = 0; x < n; x++) //forming answer there
        ret += cost[x][xy[x]];
    return ret;
}
};

```

## 6.11 max-flow-dinic

```

namespace Dinic {
    typedef long long LL;
    const int N = 2e4+100, K = 60; const LL INF = 1e18;
    struct Edge { int frm, to; LL cap, flow; };
    int s, t, n, level[N], ptr[N];
    vector<Edge> edges; vector<int> adj[N];
    void init(int nodes) {
        n = nodes; edges.clear();
        for (int i=0; i<n; i++) adj[i].clear();
        //adding undirected edge call addEdge(u,v,c,c);
    }
    int addEdge(int a, int b, LL cap, LL revcap = 0) {
        edges.push_back({a, b, cap, 0});
        edges.push_back({b, a, revcap, 0});
        adj[a].push_back(edges.size()-2);
        adj[b].push_back(edges.size()-1);
        return edges.size()-2;
    }
    bool bfs(LL lim) {
        fill(level, level+n, -1); level[s] = 0;
        queue<int> q; q.push(s);
        while (!q.empty() && level[t] == -1) {
            int v = q.front(); q.pop();
            for (int id: adj[v]) {
                Edge e = edges[id];
                if (level[e.to] == -1 && e.cap-e.flow>=lim) {
                    q.push(e.to); level[e.to] = level[v] + 1;
                }
            }
        }
        return level[t] != -1;
    }
    LL dfs(int v, LL flow) {
        if (v == t || !flow) return flow;
        for (; ptr[v] < adj[v].size(); ptr[v]++) {
            int eid = adj[v][ptr[v]];
            Edge &e = edges[eid];
            if (level[e.to] != level[v] + 1) continue;
            if (LL pushed=dfs(e.to,min(flow,e.cap-e.flow))) {
                e.flow+=pushed; edges[eid^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
    LL maxFlow(int source,int sink,bool SCALING=false){
        s = source, t = sink;
    }
}

```

```

long long flow = 0;
for (LL lim=SCALING?(1LL<<K):1; lim>0; lim>=1){
    while (bfs(lim)) {
        fill(ptr, ptr+n, 0);
        while (LL pushed = dfs(s,INF)) flow+=pushed;
    }
}
return flow;

```

## 6.12 min-cost-max-flow

```

namespace MCMF {
    typedef long long F; typedef long long C;
    const F infF = 1e18; const C infC = 1e18;
    const int N = 5005;
    typedef pair<C, F> PCF;
    struct Edge {int frm, to; C cost; F cap, flow;};
    int n, s, t;
    vector<Edge> edges;
    vector<int> adj[N];
    C pi[N], dis[N];
    F fl[N];
    int prv[N], vis[N];
    void init(int nodes, int source, int sink) {
        n = nodes, s = source, t = sink;
        for (int i=0; i<n; i++) pi[i] = 0, adj[i].clear();
        edges.clear();
    }
    void addEdge(int u, int v, F cap, C cost) {
        edges.push_back({u, v, cost, cap, 0});
        edges.push_back({v, u, -cost, 0, 0});
        adj[u].push_back(edges.size()-2);
        adj[v].push_back(edges.size()-1);
    }
    bool SPFA() {
        for (int i=0; i<n; i++) {
            dis[i] = infC; fl[i] = 0;
            vis[i] = 0; prv[i] = -1;
        }
        queue<int> q;
        q.push(s);
        dis[s] = 0; fl[s] = infF; vis[s] = 1;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            vis[u] = 0;
            for (int eid : adj[u]) {
                Edge &e = edges[eid];
                if (e.cap == e.flow) continue;
                if (dis[u] + e.cost < dis[e.to]) {
                    dis[e.to] = dis[u] + e.cost;
                    fl[e.to] = min(fl[u], e.cap - e.flow);
                    prv[e.to] = eid+1;
                    if (!vis[e.to]) q.push(e.to);
                }
            }
        }
        return fl[t] > 0;
    }
    PCF solveSPFA() {
        C cost = 0; F flow = 0;
        while (SPFA()) {
            C pathcost = dis[t];
            cost += pathcost*fl[t]; flow += fl[t];
            for (int u=t, e=prv[u]; e!=-1; u=edges[e].to, e=prv[u]) {
                edges[e].flow -= fl[t];
                edges[e^1].flow += fl[t];
            }
        }
        return {cost, flow};
    }
    void normalize() {
        SPFA();
        for (int i=0; i<n; i++) pi[i] = dis[i];
    }
    bool Dijkstra() {
        for (int i=0; i<n; i++) {
            dis[i] = infC; fl[i] = 0;
            vis[i] = 0; prv[i] = -1;
        }
        priority_queue<pair<C, int>> pq;
        pq.emplace(0, s);
        dis[s] = 0; fl[s] = infF;
        while (!pq.empty()) {
            int u = pq.top().second; pq.pop();
            if (vis[u]) continue;
            vis[u] = 1;
            for (int eid : adj[u]) {
                Edge &e = edges[eid];
                if (vis[e.to] || e.cap == e.flow) continue;
                C nw = dis[u] + e.cost - pi[e.to] + pi[u];
                if (nw < dis[e.to]) {
                    dis[e.to] = nw;
                    fl[e.to] = min(fl[u], e.cap - e.flow);
                    prv[e.to] = eid+1;
                    pq.emplace(-dis[e.to], e.to);
                }
            }
        }
        return fl[t] > 0;
    }
    PCF solveDijkstra() {
        normalize();
        C cost = 0; F flow = 0;
        while (Dijkstra()) {
            for (int i=0; i<n; i++)
                if (fl[i]) pi[i] += dis[i];
            C pathcost = pi[t]-pi[s];
            cost += pathcost*fl[t]; flow += fl[t];
            for (int u=t, e=prv[u]; e!=-1; u=edges[e].to, e=prv[u]) {
                edges[e].flow -= fl[t];
                edges[e^1].flow += fl[t];
            }
        }
        return {cost, flow};
    }
}

```

```

}
priority_queue<pair<C, int>> pq;
pq.emplace(0, s);
dis[s] = 0; fl[s] = infF;
while (!pq.empty()) {
    int u = pq.top().second; pq.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (int eid : adj[u]) {
        Edge &e = edges[eid];
        if (vis[e.to] || e.cap == e.flow) continue;
        C nw = dis[u] + e.cost - pi[e.to] + pi[u];
        if (nw < dis[e.to]) {
            dis[e.to] = nw;
            fl[e.to] = min(fl[u], e.cap - e.flow);
            prv[e.to] = eid+1;
            pq.emplace(-dis[e.to], e.to);
        }
    }
}
return fl[t] > 0;
}
PCF solveDijkstra() {
    normalize();
    C cost = 0; F flow = 0;
    while (Dijkstra()) {
        for (int i=0; i<n; i++)
            if (fl[i]) pi[i] += dis[i];
        C pathcost = pi[t]-pi[s];
        cost += pathcost*fl[t]; flow += fl[t];
        for (int u=t, e=prv[u]; e!=-1; u=edges[e].to, e=prv[u]) {
            edges[e].flow -= fl[t];
            edges[e^1].flow += fl[t];
        }
    }
    return {cost, flow};
}

```

## 6.13 online-bridge

```

vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges; int lca_iteration;
vector<int> last_visit;
void init(int n) {
    par.resize(n); dsu_2ecc.resize(n); dsu_cc.resize(n);
    dsu_cc_size.resize(n); lca_iteration=0; last_visit.assign(n,0);
    for (int i=0; i<n; i++) {
        dsu_2ecc[i] = i; dsu_cc[i] = i; dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}
int find_2ecc(int v) {
    if (v == -1) return -1;
    return dsu_2ecc[v]==v?v:dsu_2ecc[v]=find_2ecc(dsu_2ecc[v]);
}
int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v]==v?v : dsu_cc[v] = find_cc(dsu_cc[v]);
}
void make_root(int v) {
    v = find_2ecc(v); int root = v; int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]); par[v] = child;
        dsu_cc[v] = root; child = v; v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}
void merge_path (int a, int b) {
    ++lca_iteration; vector<int> path_a, path_b; int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a); path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a; break;
            }
        }
        last_visit[a] = lca_iteration; a = par[a];
    }
}

```

```

}
if (b != -1) {
    b = find_2ecc(b); path_b.push_back(b);
    if (last_visit[b] == lca_iteration){
        lca = b; break;
    }
    last_visit[b] = lca_iteration; b = par[b];
}
for (int v : path_a) {
    dsu_2ecc[v] = lca; if (v == lca) break; --bridges;
}
for (int v : path_b) {
    dsu_2ecc[v] = lca; if (v == lca) break; --bridges;
}
}
void add_edge(int a, int b) {
    a = find_2ecc(a); b = find_2ecc(b);
    if (a == b) return;
    int ca = find_cc(a); int cb = find_cc(b);
    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b); swap(ca, cb);
        }
        make_root(a); par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
}

```

## 6.14 scc + 2 Sat

```

namespace SCC { //Everything 0-indexed.
    const int N = 2e6+7; int which[N], vis[N], cc;
    vector<int> adj[N], adjr[N]; vector<int> order;
    void addEdge(int u, int v) {
        adj[u].push_back(v); adjr[v].push_back(u);
    }
    void dfs1(int u){
        if (vis[u]) return; vis[u] = true;
        for(int v: adj[u]) dfs1(v); order.push_back(u);
    }
    void dfs2(int u, int id) {
        if(vis[u]) return; vis[u] = true;
        for(int v: adjr[u]) dfs2(v, id); which[u] = id;
    }
    int last = 0;
    void findSCC(int n) {
        cc=0;last=n; order.clear(); fill(vis, vis+n, 0);
        for(int i=0; i<n; i++) if(!vis[i]) dfs1(i);
        reverse(order.begin(), order.end());
        fill(vis, vis+n, 0);
        for (int u: order) {
            if (vis[u]) continue; dfs2(u, cc); ++cc;
        }
    }
    void clear() {
        for (int i=0; i<last; i++)
            adj[i].clear(), adjr[i].clear();
    }
}
struct TwoSat {
    int n; int vars = 0; vector<bool> ans;
    TwoSat(int n) : n(n), ans(n) {
        SCC::clear(); vars = 2*n;
    }
    void implies(int x, int y) {
        SCC::addEdge(x, y); SCC::addEdge(y^1, x^1);
    }
    void OR(int x, int y) {
        SCC::addEdge(x^1, y); SCC::addEdge(y^1, x);
    }
    void XOR(int x, int y) {
        implies(x, y^1); implies(x^1, y);
    }
    void atmostOne(vector<int> v) {
        int k = v.size();
        for (int i=0; i<k; i++) {
            if (i+1<k) implies(vars+2*i, vars+2*i+2);
            implies(v[i], vars+2*i);
        }
    }
}

```

```

    if (i>0)    implies(v[i], vars+2*i-1);
}
vars += 2*k;
}
bool solve() {
    SCC::findSCC(vars); ans.resize(vars/2);
    for (int i=0; i<vars; i+=2) {
        if (SCC::which[i]==SCC::which[i+1])return 0;
        if (i<2*n)
            ans[i/2] = SCC::which[i]>SCC::which[i+1];
    }
    return true;
}
};

```

## 7 Math

### 7.1 BerleKampMassey

```

const int MOD = 998244353;
vector<LL> berlekampMassey(vector<LL> s) {
    if (s.empty()) return {};
    int n = s.size(), L = 0, m = 0;
    vector<LL> C(n), B(n), T;
    C[0] = B[0] = 1; LL b = 1;
    for (int i = 0; i < n; ++i) {
        ++m; LL d = s[i] % MOD;
        for (int j = 1; j <= L; ++j) d = (d + C[j] * s[i - j]) % MOD;
        if (!d) continue;
        T = C; LL coeff = d * bigMod(b, -1) % MOD;
        for (int j = m; j < n; ++j) C[j] = (C[j] - coeff * B[j - m]) % MOD;
        if (2*L > i) continue;
        L = i+1-L, B = T, b = d, m = 0;
    }
    C.resize(L + 1), C.erase(C.begin());
    for (LL &x : C) x = (MOD - x) % MOD;
    return C;
}

```

### 7.2 FloorSum

```

LL mod(LL a, LL m) {
    LL ans = a%m;
    return ans < 0 ? ans+m : ans;
}
//Sum(floor((ax+b)/m)) for i=0 to n-1, (n,m >= 0)
LL floorSum(LL n, LL m, LL a, LL b) {
    LL ra = mod(a, m), rb = mod(b, m), k = (ra*n+rb);
    LL ans = ((a-ra)/m) * n*(n-1)/2 + ((b-rb)/m) * n;
    if (k < m) return ans;
    return ans + floorSum(k/m, ra, m, k%m);
}

```

### 7.3 Stern Brocot Tree

```

//finds x/y with min y st: L <= (x/y) < R
pair<LL,LL> solve(LD L, LD R){
    pair<LL, LL> l(0, 1), r(1, 1);
    if(L==0.0) return l; // corner case
    while(true) {
        pair<int, int> m(l.x+r.x, l.y+r.y);
        if(m.x<L*m.y){ // move to the right
            LL kl=1, kr=1;
            while(l.x+kr*r.x <= L*(l.y+kr*r.y)) kr*=2;
            while(kl!=kr){
                LL km = (kl+kr)/2;
                if(l.x+km*r.x < L*(l.y+km*r.y)) kl=km+1;
                else kr=km;
            }
            l={l.x+(kl-1)*r.x,l.y+(kl-1)*r.y};
        }
        else if(m.x>R*m.y){ //move to left
            LL kl=1, kr=1;
            while(r.x+kr*l.x>R*(r.y+kr*l.y)) kr*=2;
            while(kl!=kr){
                LL km = (kl+kr)/2;
                if(r.x+km*l.x>R*(r.y+km*l.y)) kl = km+1;
                else kr = km;
            }
        }
    }
}

```

```

    r={r.x+(kl-1)*l.x,r.y+(kl-1)*l.y};
}
else return m;
}
}

```

### 7.4 Sum Of Kth Power

```

LL mod; LL S[105][105];
// Find 1^k+2^k+...+n^k % mod
void solve() {
    LL n, k;
    /* x^k = sum (i=1 to k) Stirling2(k, i) * i! * ncr(x, i)
    sum (x = 0 to n) x^k
    = sum (i=0 to k) Stirling2(k, i) * i! * sum (x=0 to n) ncr(x, i)
    = sum (i=0 to k) Stirling2(k, i) * i! * ncr(n + 1, i + 1)
    = sum (i=0 to k) Stirling2(k, i) * i! * (n + 1)! / (i+1)! / (n-i)!
    = sum (i=0 to k) Stirling2(k, i) * (n - i + 1) *
        (n - i + 2) * ... (n + 1) / (i + 1) */
    S[0][0] = 1 % mod;
    for (int i = 1; i <= k; i++) {
        for (int j = 1; j <= i; j++) {
            if (i == j) S[i][j] = 1 % mod;
            else S[i][j] = (j * S[i - 1][j] + S[i - 1][j - 1]) % mod;
        }
    }
    LL ans = 0;
    for (int i = 0; i <= k; i++) {
        LL fact = 1, z = i + 1;
        for (LL j = n - i + 1; j <= n + 1; j++) {
            LL mul = j;
            if (mul % z == 0) {
                mul /= z; z /= z;
            }
            fact = (fact * mul) % mod;
        }
        ans = (ans + S[k][i] * fact) % mod;
    }
}

```

### 7.5 combination-generator

```

bool next_combination(vector<int>& a, int n) {
    int k = (int)a.size();
    for (int i = k - 1; i >= 0; i--) {
        if (a[i] < n - k + i + 1) {
            a[i]++;
            for (int j = i + 1; j < k; j++)
                a[j] = a[j - 1] + 1;
            return true;
        }
    }
    return false;
}
vector<int> ans;
void gen(int n, int k, int idx, bool rev) {
    if (k > n || k < 0)
        return;
    if (!n) {
        for (int i = 0; i < idx; ++i) {
            if (ans[i])
                cout << i + 1;
        }
        cout << "\n";
        return;
    }
    ans[idx] = rev;
    gen(n - 1, k - rev, idx + 1, false);
    ans[idx] = !rev;
    gen(n - 1, k - rev, idx + 1, true);
}
void all_combinations(int n, int k) {
    ans.resize(n);
    gen(n, k, 0, false);
}

```

### 7.6 continued-fractions

```

auto fraction(int p, int q) {
    vector<int> a;

```

```

    while(q) {
        a.push_back(p / q);
        tie(p, q) = make_pair(q, p % q);
    }
    return a;
}
auto convergents(vector<int> a) {
    vector<int> p = {0, 1};
    vector<int> q = {1, 0};
    for(auto it: a) {
        p.push_back(p[p.size() - 1] * it + p[p.size() - 2]);
        q.push_back(q[q.size() - 1] * it + q[q.size() - 2]);
    }
    return make_pair(p, q);
}

```

### 7.7 convolution

```

//zeta transform or sos dp
void zeta(vll& d,int m){
    int n=1<<m;
    for(int len=2;len<=n;len*=2){
        for(int i=0;i<n;i+=len){
            int l2=len>>1;
            for(int j=i;j<i+l2;j++){
                d[j+l2]+=d[j];
            }
        }
    }
//zeta_inverse or mobius transform
void zinv(vll &d,int m){
    int n=1<<m;
    for(int len=2;len<=n;len*=2){
        for(int i=0;i<n;i+=len){
            int l2=len>>1;
            for(int j=i;j<i+l2;j++){
                d[j+l2]-=d[j];
            }
        }
    }
//subset sum convolution
//not fully tested,got some error if used with mod
#define MAX_SIZE 1<<20
ll f[MAX_SIZE];
ll g[MAX_SIZE];
ll res[MAX_SIZE];
ll fhat[20][MAX_SIZE];
ll ghat[20][MAX_SIZE];
ll h[20][MAX_SIZE];
void subsetConvolution(int m){
    int n=1<<m;
    memset(fhat,0,sizeof(fhat));
    memset(ghat,0,sizeof(ghat));
    memset(res,0,sizeof(res));
    memset(h,0,sizeof(h));
    for(int i=0;i<n;i++){
        fhat[__builtin_popcount(i)][i]=f[i];
        ghat[__builtin_popcount(i)][i]=g[i];
    }
    for (int i=0; i<m; i++) {
        zeta(fhat[i],m);
        zeta(ghat[i],m);
        for (int j=0; j<=i; j++){
            for (int mask = 0; mask < n; mask++){
                h[i][mask] += fhat[j][mask]*ghat[i-j][mask];
            }
        }
        zinv(h[i],m);
    }
    for(int i=0;i<n;i++)
        res[i]=h[__builtin_popcount(i)][i];
}

```

### 7.8 crt anachor

```

// Chinese remainder theorem (special case): find z st z%m1 = r1, z%m2 = r2.
// z is unique modulo M = lcm(m1, m2). Returns (z, M). On failure, M = -1.
PLL CRT(LL m1, LL r1, LL m2, LL r2) {
    LL s, t;
    LL g = egcd(m1, m2, s, t);
    if (r1%g != r2%g) return PLL(0, -1);
    LL M = m1*m2;
    LL ss = ((s*r2)%m2)*m1;
    LL tt = ((t*r1)%m1)*m2;
    LL ans = ((ss+tt)%M+M)%M;
    return PLL(ans/g, M/g);
}
// expected: 23 105

```



```
//      11 12
PLL ans = CRT({3,5,7}, {2,3,2});
cout << ans.first << " " << ans.second << endl;
ans = CRT({4,6}, {3,5});
cout << ans.first << " " << ans.second << endl;
```

## 7.9 discrete-root

```
#define MAX 100000
int prime[MAX+1], Phi[MAX+1];
vector<int> pr;
void sieve(){
    for (int i=2; i <= N; ++i) {
        if (prime[i] == 0) {
            prime[i] = i;
            pr.push_back(i);
        }
        for (int j=0; j < (int)pr.size() && pr[j] <= prime[i]
            && i*pr[j] <= N; ++j) {
            prime[i * pr[j]] = pr[j];
        }
    }
}
void PhiWithSieve(){
    int i;
    for(i=2; i<=MAX; i++){
        if(prime[i]==i){
            Phi[i]=i-1;
        }
        else if((i/prime[i])%prime[i]==0){
            Phi[i]=Phi[i/prime[i]]*prime[i];
        }
        else{
            Phi[i]=Phi[i/prime[i]]*(prime[i]-1);
        }
    }
}
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p), --b;
        else
            a = int (a * 1ll * a % p), b >>= 1;
    return res;
}
int PrimitiveRoot(int p){
    vector<int> fact;
    int phi=Phi[p];
    int n=phi;
    while(n>1){
        if(prime[n]==n){
            fact.push_back(n);
            n=1;
        }
        else{
            int f=prime[n];
            while(n%f==0){
                n=n/f;
            }
            fact.push_back(f);
        }
    }
    int res;
    for(res=p-1; res>1; --res){
        for(n=0; n<fact.size(); n++){
            if(powmod(res,phi/fact[n],p)==1){
                break;
            }
        }
        if(n==fact.size()) return res;
    }
    return -1;
}
ll DiscreteLog(ll a,ll b,ll mod){
    ll n,p,q;
    n=(ll)sqrt((double)mod)+1ll;
    ll an=powmod(a,n,mod);
    map<ll,ll> map_;
    ll cur=an;
```

```
for(p=1;p<=n;p++){
    if(map_.find(cur)==map_.end()){
        map_.insert({cur,p});
    }
    cur=(cur*an)%mod;
}
cur=b;
ll mn=LLONG_MAX;
for(q=0;q<=n;q++){
    if(map_.find(cur)!=map_.end()){
        mn=min((n*map_[cur])-q,mn);
    }
    cur=(cur*a)%mod;
}
if(mn==LLONG_MAX)return -1;
else return mn;
}
vector<int> DiscreteRoot(int n,int a,int k){
    int g = PrimitiveRoot(n);
    vector<int> ans;
    int any_ans = DiscreteLog(powmod(g,k,n),a,n);
    if (any_ans == -1){
        return ans;
    }
    int delta = (n-1) / gcd(k, n-1);
    for (int cur = any_ans % delta; cur < n-1; cur += delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```

## 7.10 fast-walsh-hadamard

```
void FWHT(vector<LL> &p, bool inv) {
    int n = p.size(); assert((n&(n-1))==0);
    for (int len=1; 2*len<=n; len<= 1) {
        for (int i = 0; i < n; i += len+len){
            for (int j = 0; j < len; j++) {
                LL u = p[i+j], v = p[i+len+j];
                ///XOR p[i+j]=u+v; p[i+len+j]=u-v;
                ///OR if(!inv) p[i+j]=v, p[i+len+j]=u+v;
                ///OR else p[i+j]=-u+v, p[i+len+j]=u;
                ///AND if(!inv) p[i+j]=u+v, p[i+len+j]=u;
                ///AND else p[i+j]=v, p[i+len+j]=u-v;
            }
        }
    }
    ///XOR if(inv) for(int i=0;i<n;i++) p[i]/=n;
}
vector<LL> convo(vector<LL> a, vector<LL> b) {
    int n = 1, sz = max(a.size(), b.size());
    while(n<sz) n*=2;
    a.resize(n); b.resize(n); vector<LL> res(n, 0);
    FWHT(a, 0); FWHT(b, 0);
    for(int i=0;i<n;i++) res[i] = a[i] * b[i];
    FWHT(res, 1);
    return res;
}
```

## 7.11 fft

```
struct CD {
    double x, y;
    CD(double x=0, double y=0) :x(x), y(y) {}
    CD operator+(const CD& o) { return {x+o.x, y+o.y};}
    CD operator-(const CD& o) { return {x-o.x, y-o.y};}
    CD operator*(const CD& o) { return {x*o.x-y*o.y,
        x*o.y+o.x*y};}
    void operator /= (double d) { x/=d; y/=d;}
    double real() {return x;}
    double imag() {return y;}
}
CD conj(const CD &c) {return CD(c.x, -c.y);}
const double PI = acos(-1.0L);
namespace FFT {
    int N;
    vector<int> perm;
    vector<CD> wp[2];
    void precalculate(int n) {
        assert((n & (n-1)) == 0);
```

```
N = n;
perm = vector<int> (N, 0);
for (int k=1; k<N; k<=1) {
    for (int i=0; i<k; i++) {
        perm[i] <= 1;
        perm[i+k] = 1 + perm[i];}
}
wp[0] = wp[1] = vector<CD>(N);
for (int i=0; i<N; i++) {
    wp[0][i] = CD( cos(2*PI*i/N), sin(2*PI*i/N) );
    wp[1][i] = CD( cos(2*PI*i/N), -sin(2*PI*i/N) );}
void fft(vector<CD> &v, bool invert = false) {
    if (v.size() != perm.size()) precalculate(v.size());
    for (int i=0; i<N; i++)
        if (i < perm[i])
            swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len *= 2) {
        for (int i=0, d = N/len; i<N; i+=len) {
            for (int j=0, idx=0; j<len/2; j++, idx += d) {
                CD x = v[i+j];
                CD y = wp[invert][idx]*v[i+j+len/2];
                v[i+j] = x+y;
                v[i+j+len/2] = x-y;}}
    }
    if (invert) {
        for (int i=0; i<N; i++) v[i]/=N;}}
void pairfft(vector<CD> &a, vector<CD> &b, bool invert =
    false) {
    int N = a.size();
    vector<CD> p(N);
    for (int i=0; i<N; i++) p[i] = a[i] + b[i] * CD(0, 1);
    fft(p, invert);
    p.push_back(p[0]);
    for (int i=0; i<N; i++) {
        if (invert) {
            a[i] = CD(p[i].real(), 0);
            b[i] = CD(p[i].imag(), 0);}
        else {
            a[i] = (p[i]+conj(p[N-i]))*CD(0.5, 0);
            b[i] = (p[i]-conj(p[N-i]))*CD(0, -0.5);}}
}
vector<ll> multiply(const vector<ll> &a, const vector<ll>
    &b) {
    int n = 1;
    while (n < a.size()+ b.size()) n<=1;
    vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    fa.resize(n); fb.resize(n);
    // fft(fa); fft(fb);
    pairfft(fa, fb);
    for (int i=0; i<n; i++) fa[i] = fa[i] * fb[i];
    fft(fa, true);
    vector<ll> ans(n);
    for (int i=0; i<n; i++) ans[i] = round(fa[i].real());
    return ans;
}
const int M = 1e9+7, B = sqrt(M)+1;
vector<ll> anyMod(const vector<ll> &a, const vector<ll> &b) {
    int n = 1;
    while (n < a.size()+ b.size()) n<=1;
    vector<CD> al(n), ar(n), bl(n), br(n);
    for (int i=0; i<a.size(); i++) al[i] = a[i]%M/B, ar[i] =
        a[i]%M/B;
    for (int i=0; i<b.size(); i++) bl[i] = b[i]%M/B, br[i] =
        b[i]%M/B;
    pairfft(al, ar); pairfft(bl, br);
    // fft(al); fft(ar); fft(bl); fft(br);
    for (int i=0; i<n; i++) {
        CD l1 = (al[i] * bl[i]), lr = (al[i] * br[i]);
        CD r1 = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
        al[i] = l1; ar[i] = lr;
        bl[i] = r1; br[i] = rr;
    }
    pairfft(al, ar, true); pairfft(bl, br, true);
    // fft(al, true); fft(ar, true); fft(bl, true); fft(br,
        true);
    vector<ll> ans(n);
    for (int i=0; i<n; i++) {
        ll right = round(br[i].real()), left =
            round(al[i].real());
        ll mid = round(round(bl[i].real()) + round(ar[i].real()));
        ans[i] = ((left%M)*B*B + (mid%M)*B + right)%M;
    }
    return ans;}
```

## 7.12 formulas

## Binomial Coefficient List

- $\sum_{k=0}^m \binom{n+k}{k} = \binom{n+m+1}{m}.$
- $\binom{n}{0} + \binom{n-1}{1} + \dots + \binom{n-k}{k} + \dots + \binom{0}{n} = F_{n+1}$

## Catalan's Triangle

- $C(n, 0) = 1, n \geq 0.$
- $C(n, 1) = n, n \geq 1.$
- $C(n+1, k) = C(n+1, k-1) + C(n, k), 1 < k < n+1.$
- $C(n, k) = \binom{n+k}{k} - \binom{n+k}{k-1} = \frac{n-k+1}{n+1} \binom{n+k}{k}.$

## Fibonacci Numbers

- $F_{n-1}F_{n+1} - F_n^2 = (-1)^n.$
- $F_{n+k} = F_kF_{n+1} + F_{k-1}F_n.$

## 7.13 integer-factorization

```
typedef long long LL;
typedef unsigned long long ULL;

namespace Rho {
ULL mult(ULL a, ULL b, ULL mod) {
    LL ret = a * b - mod * (ULL)(1.0L/mod*a*b);
    return ret+mod*(ret<0) - mod*(ret>=(LL) mod);
}
ULL power(ULL x, ULL p, ULL mod){
    ULL s=1, m=x;
    while(p) {
        if(p&1) s = mult(s, m, mod);
        p>>=1; m = mult(m, m, mod);
    } return s;
}
vector<LL> bases =
{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,187,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,473,479,487,491,499,503,509,521,523,527,531,539,547,557,563,569,571,577,587,593,599,601,607,613,617,619,623,629,631,637,641,643,647,653,659,661,667,671,673,677,683,687,691,697,701,709,713,719,727,733,739,743,751,757,761,769,773,787,797,809,811,823,827,833,839,847,853,857,863,869,877,881,883,887,893,897,907,911,913,917,923,929,937,941,947,953,959,967,971,973,977,983,989,991,993,997,1009,1013,1017,1021,1031,1033,1037,1043,1047,1051,1057,1063,1067,1073,1079,1087,1093,1097,1103,1107,1117,1123,1127,1133,1137,1147,1153,1157,1163,1169,1177,1181,1183,1187,1193,1197,1201,1207,1213,1217,1223,1229,1231,1237,1243,1247,1253,1259,1267,1271,1277,1283,1287,1291,1297,1301,1303,1307,1313,1317,1321,1327,1331,1337,1343,1347,1351,1357,1363,1367,1373,1379,1381,1387,1393,1397,1403,1407,1411,1417,1423,1427,1433,1439,1443,1447,1453,1457,1463,1469,1471,1477,1483,1487,1493,1497,1501,1507,1511,1517,1523,1527,1531,1537,1543,1547,1553,1559,1567,1571,1573,1577,1583,1587,1591,1597,1601,1603,1607,1613,1617,1621,1627,1631,1637,1643,1647,1651,1657,1663,1667,1673,1679,1681,1687,1693,1697,1703,1707,1711,1717,1723,1727,1733,1737,1743,1747,1753,1757,1763,1769,1771,1777,1783,1787,1793,1797,1801,1807,1811,1813,1817,1823,1827,1831,1837,1843,1847,1853,1857,1863,1867,1873,1879,1881,1887,1893,1897,1903,1907,1913,1917,1923,1927,1931,1937,1943,1947,1953,1957,1963,1967,1973,1979,1981,1987,1993,1997,2003,2007,2011,2017,2023,2027,2033,2037,2043,2047,2053,2057,2063,2069,2071,2077,2083,2087,2093,2097,2101,2107,2111,2113,2117,2123,2127,2131,2137,2143,2147,2153,2157,2163,2167,2173,2179,2181,2187,2193,2197,2203,2207,2213,2217,2223,2229,2231,2237,2243,2247,2253,2257,2263,2269,2271,2277,2283,2287,2293,2297,2303,2307,2311,2317,2323,2327,2333,2337,2343,2347,2353,2357,2363,2369,2371,2377,2383,2387,2393,2397,2403,2407,2411,2417,2423,2427,2431,2437,2443,2447,2453,2457,2463,2469,2471,2477,2483,2487,2493,2497,2503,2507,2511,2517,2523,2527,2533,2537,2543,2547,2553,2557,2563,2569,2571,2577,2583,2587,2593,2597,2603,2607,2611,2617,2623,2627,2633,2637,2643,2647,2653,2657,2663,2669,2671,2677,2683,2687,2693,2697,2703,2707,2711,2717,2723,2727,2731,2737,2743,2747,2753,2757,2763,2769,2771,2777,2783,2787,2793,2797,2803,2807,2811,2817,2823,2827,2833,2837,2843,2847,2853,2857,2863,2869,2871,2877,2883,2887,2893,2897,2903,2907,2911,2917,2923,2927,2933,2937,2943,2947,2953,2957,2963,2969,2971,2977,2983,2987,2993,2997,3003,3007,3011,3017,3023,3027,3033,3037,3043,3047,3053,3057,3063,3069,3071,3077,3083,3087,3093,3097,3103,3107,3111,3117,3123,3127,3133,3137,3143,3147,3153,3157,3163,3169,3171,3177,3183,3187,3193,3197,3203,3207,3211,3217,3223,3227,3233,3237,3243,3247,3253,3257,3263,3269,3271,3277,3283,3287,3293,3297,3303,3307,3311,3317,3323,3327,3333,3337,3343,3347,3353,3357,3363,3369,3371,3377,3383,3387,3393,3397,3403,3407,3411,3417,3423,3427,3433,3437,3443,3447,3453,3457,3463,3469,3471,3477,3483,3487,3493,3497,3503,3507,3511,3517,3523,3527,3533,3537,3543,3547,3553,3557,3563,3569,3571,3577,3583,3587,3593,3597,3603,3607,3611,3617,3623,3627,3633,3637,3643,3647,3653,3657,3663,3669,3671,3677,3683,3687,3693,3697,3703,3707,3711,3717,3723,3727,3733,3737,3743,3747,3753,3757,3763,3769,3771,3777,3783,3787,3793,3797,3803,3807,3811,3817,3823,3827,3833,3837,3843,3847,3853,3857,3863,3869,3871,3877,3883,3887,3893,3897,3903,3907,3911,3917,3923,3927,3933,3937,3943,3947,3953,3957,3963,3969,3971,3977,3983,3987,3993,3997,4003,4007,4011,4017,4023,4027,4033,4037,4043,4047,4053,4057,4063,4069,4071,4077,4083,4087,4093,4097,4103,4107,4111,4117,4123,4127,4133,4137,4143,4147,4153,4157,4163,4169,4171,4177,4183,4187,4193,4197,4203,4207,4211,4217,4223,4227,4233,4237,4243,4247,4253,4257,4263,4269,4271,4277,4283,4287,4293,4297,4303,4307,4311,4317,4323,4327,4333,4337,4343,4347,4353,4357,4363,4369,4371,4377,4383,4387,4393,4397,4403,4407,4411,4417,4423,4427,4433,4437,4443,4447,4453,4457,4463,4469,4471,4477,4483,4487,4493,4497,4503,4507,4511,4517,4523,4527,4533,4537,4543,4547,4553,4557,4563,4569,4571,4577,4583,4587,4593,4597,4603,4607,4611,4617,4623,4627,4633,4637,4643,4647,4653,4657,4663,4669,4671,4677,4683,4687,4693,4697,4703,4707,4711,4717,4723,4727,4733,4737,4743,4747,4753,4757,4763,4769,4771,4777,4783,4787,4793,4797,4803,4807,4811,4817,4823,4827,4833,4837,4843,4847,4853,4857,4863,4869,4871,4877,4883,4887,4893,4897,4903,4907,4911,4917,4923,4927,4933,4937,4943,4947,4953,4957,4963,4969,4971,4977,4983,4987,4993,4997,5003,5007,5011,5017,5023,5027,5033,5037,5043,5047,5053,5057,5063,5069,5071,5077,5083,5087,5093,5097,5103,5107,5111,5117,5123,5127,5133,5137,5143,5147,5153,5157,5163,5169,5171,5177,5183,5187,5193,5197,5203,5207,5211,5217,5223,5227,5233,5237,5243,5247,5253,5257,5263,5269,5271,5277,5283,5287,5293,5297,5303,5307,5311,5317,5323,5327,5333,5337,5343,5347,5353,5357,5363,5369,5371,5377,5383,5387,5393,5397,5403,5407,5411,5417,5423,5427,5433,5437,5443,5447,5453,5457,5463,5469,5471,5477,5483,5487,5493,5497,5503,5507,5511,5517,5523,5527,5533,5537,5543,5547,5553,5557,5563,5569,5571,5577,5583,5587,5593,5597,5603,5607,5611,5617,5623,5627,5633,5637,5643,5647,5653,5657,5663,5669,5671,5677,5683,5687,5693,5697,5703,5707,5711,5717,5723,5727,5733,5737,5743,5747,5753,5757,5763,5769,5771,5777,5783,5787,5793,5797,5803,5807,5811,5817,5823,5827,5833,5837,5843,5847,5853,5857,5863,5869,5871,5877,5883,5887,5893,5897,5903,5907,5911,5917,5923,5927,5933,5937,5943,5947,5953,5957,5963,5969,5971,5977,5983,5987,5993,5997,6003,6007,6011,6017,6023,6027,6033,6037,6043,6047,6053,6057,6063,6069,6071,6077,6083,6087,6093,6097,6103,6107,6111,6117,6123,6127,6133,6137,6143,6147,6153,6157,6163,6169,6171,6177,6183,6187,6193,6197,6203,6207,6211,6217,6223,6227,6233,6237,6243,6247,6253,6257,6263,6269,6271,6277,6283,6287,6293,6297,6303,6307,6311,6317,6323,6327,6333,6337,6343,6347,6353,6357,6363,6369,6371,6377,6383,6387,6393,6397,6403,6407,6411,6417,6423,6427,6433,6437,6443,6447,6453,6457,6463,6469,6471,6477,6483,6487,6493,6497,6503,6507,6511,6517,6523,6527,6533,6537,6543,6547,6553,6557,6563,6569,6571,6577,6583,6587,6593,6597,6603,6607,6611,6617,6623,6627,6633,6637,6643,6647,6653,6657,6663,6669,6671,6677,6683,6687,6693,6697,6703,6707,6711,6717,6723,6727,6733,6737,6743,6747,6753,6757,6763,6769,6771,6777,6783,6787,6793,6797,6803,6807,6811,6817,6823,6827,6833,6837,6843,6847,6853,6857,6863,6869,6871,6877,6883,6887,6893,6897,6903,6907,6911,6917,6923,6927,6933,6937,6943,6947,6953,6957,6963,6969,6971,6977,6983,6987,6993,6997,7003,7007,7011,7017,7023,7027,7033,7037,7043,7047,7053,7057,7063,7069,7071,7077,7083,7087,7093,7097,7103,7107,7111,7117,7123,7127,7133,7137,7143,7147,7153,7157,7163,7169,7171,7177,7183,7187,7193,7197,7203,7207,7211,7217,7223,7227,7233,7237,7243,7247,7253,7257,7263,7269,7271,7277,7283,7287,7293,7297,7303,7307,7311,7317,7323,7327,7333,7337,7343,7347,7353,7357,7363,7369,7371,7377,7383,7387,7393,7397,7403,7407,7411,7417,7423,7427,7433,7437,7443,7447,7453,7457,7463,7469,7471,7477,7483,7487,7493,7497,7503,7507,7511,7517,7523,7527,7533,7537,7543,7547,7553,7557,7563,7569,7571,7577,7583,7587,7593,7597,7603,7607,7611,7617,7623,7627,7633,7637,7643,7647,7653,7657,7663,7669,7671,7677,7683,7687,7693,7697,7703,7707,7711,7717,7723,7727,7733,7737,7743,7747,7753,7757,7763,7769,7771,7777,7783,7787,7793,7797,7803,7807,7811,7817,7823,7827,7833,7837,7843,7847,7853,7857,7863,7869,7871,7877,7883,7887,7893,7897,7903,7907,7911,7917,7923,7927,7933,7937,7943,7947,7953,7957,7963,7969,7971,7977,7983,7987,7993,7997,8003,8007,8011,8017,8023,8027,8033,8037,8043,8047,8053,8057,8063,8069,8071,8077,8083,8087,8093,8097,8103,8107,8111,8117,8123,8127,8133,8137,8143,8147,8153,8157,8163,8169,8171,8177,8183,8187,8193,8197,8203,8207,8211,8217,8223,8227,8233,8237,8243,8247,8253,8257,8263,8269,8271,8277,8283,8287,8293,8297,8303,8307,8311,8317,8323,8327,8333,8337,8343,8347,8353,8357,8363,8369,8371,8377,8383,8387,8393,8397,8403,8407,8411,8417,8423,8427,8433,8437,8443,8447,8453,8457,8463,8469,8471,8477,8483,8487,8493,8497,8503,8507,8511,8517,8523,8527,8533,8537,8543,8547,8553,8557,8563,8569,8571,8577,8583,8587,8593,8597,8603,8607,8611,8617,8623,8627,8633,8637,8643,8647,8653,8657,8663,8669,8671,8677,8683,8687,8693,8697,8703,8707,8711,8717,8723,8727,8733,8737,8743,8747,8753,8757,8763,8769,8771,8777,8783,8787,8793,8797,8803,8807,8811,8817,8823,8827,8833,8837,8843,8847,8853,8857,8863,8869,8871,8877,8883,8887,8893,8897,8903,8907,8911,8917,8923,8927,8933,8937,8943,8947,8953,8957,8963,8969,8971,8977,8983,8987,8993,8997,9003,9007,9011,9017,9023,9027,9033,9037,9043,9047,9053,9057,9063,9069,9071,9077,9083,9087,9093,9097,9103,9107,9111,9117,9123,9127,9133,9137,9143,9147,9153,9157,9163,9169,9171,9177,9183,9187,9193,9197,9203,9207,9211,9217,9223,9227,9233,9237,9243,9247,9253,9257,9263,9269,9271,9277,9283,9287,9293,9297,9303,9307,9311,9317,9323,9327,9333,9337,9343,9347,9353,9357,9363,9369,9371,9377,9383,9387,9393,9397,9403,9407,9411,9417,9423,9427,9433,9437,9443,9447,9453,9457,9463,9469,9471,9477,9483,9487,9493,9497,9503,9507,9511,9517,9523,9527,9533,9537,9543,9547,9553,9557,9563,9569,9571,9577,9583,9587,9593,9597,9603,9607,9611,9617,9623,9627,9633,9637,9643,9647,9653,9657,9663,9669,9671,9677,9683,9687,9693,9697,9703,9707,9711,9717,9723,9727,9733,9737,9743,9747,9753,9757,9763,9769,9771,9777,9783,9787,9793,9797,9803,9807,9811,9817,9823,9827,9833,9837,9843,9847,9853,9857,9863,9869,9871,9877,9883,9887,9893,9897,9903,9907,9911,9917,9923,9927,9933,9937,9943,9947,9953,9957,9963,9969,9971,9977,9983,9987,9993,9997,10003,10007,10011,10017,10023,10027,10033,10037,10043,10047,10053,10057,10063,10069,10071,10077,10083,10087,10093,10097,10103,10107,10111,10117,10123,10127,10133,10137,10143,10147,10153,10157,10163,10169,10171,10177,10183,10187,10193,10197,10203,10207,10211,10217,10223,10227,10233,10237,10243,10247,10253,10257,10263,10269,10271,10277,10283,10287,10293,10297,10303,10307,10311,10317,10323,10327,10333,10337,10343,10347,10353,10357,10363,10369,10371,10377,10383,10387,10393,10397,10403,10407,10411,10417,10423,10427,10433,10437,10443,10447,10453,10457,10463,10469,10471,10477,10483,10487,10493,10497,10503,10507,10511,10517,10523,10527,10533,10537,10543,10547,10553,10557,10563,10569,10571,10577,10583,10587,10593,10597,10603,10607,10611,10617,10623,10627,10633,10637,10643,10647,10653,10657,10663,10669,10671,10677,10683,10687,10693,10697,10703,10707,10711,10717,10723,10727,10733,10737,10743,10747,10753,10757,10763,10769,10771,10777,10783,10787,10793,10797,10803,10807,10811,10817,10823,10827,10833,10837,10843,10847,10853,10857,10863,10869,10871,10877,10883,10887,10893,10897,10903,10907,10911,10917,10923,10927,10933,10937,10943,10947,10953,10957,10963,10969,10971,10977,10983,10987,10993,10997,11003,11007,11011,11017,11023,11027,11033,11037,11043,11047,11053,11057,11063,11069,11071,11077,11083,11087,11093,11097,11103,11107,11111,11117,11123,11127,11133,11137,11143,11147,11153,11157,11163,11169,11171,11177,11183,11187,11193,11197,11203,11207,11211,11217,11223,11227,11233,11237,11243,11247,11253,11257,11263,11269,11271,11277,11283,11287,11293,11297,11303,11307,11311,11317,11323,11327,11333,11337,11343,11347,11353,11357,11363,11369,11371,11377,11383,11387,11393,11397,11403,11407,11411,11417,11423,11427,11433,11437,11443,11447,11453,11457,11463,11469,11471,11477,11483,11487,11493,11497,11503
```

```

    pw++;}
    pf[i]=j;
    pfp[i]=pw;
    i++;}
return i;}
int get_all_factor(vector<int>& pf,vector<int>& pfp,int
sz,vector<int> &vct){
    vct[0]=1;
    int i,j,k,l,r,s=1;
    for(i=0;i<sz;i++){
        l=0;
        for(j=0;j<pfp[i];j++){
            r=s;
            for(k=1;k<r;k++,s++){
                vct[s]=(vct[k]*pf[i]);}
            l=r;}}
    return s;}
/*****General multiplicative function*****/
int *mf;
int base_case(int p,int k){//base case for p^k
    return k+1;}
void comp_mult_func(int n){
    mf=new int[n+1];
    memset(mf,-1,sizeof(int)*(n+1));
    int i,k,k2;ll l;
    mf[1]=1;
    for(i=2;i<=n;i++){
        if(mf[i]==-1){
            for(l=i+1;l<=n;l+=i)
                mf[l]=-1;
            l=i;k=1;
            while(l<=(ll)n){
                mf[l]=base_case(i,k);
                k++;
                l*=i;}}
        for(i=2;i<=n;i++){
            if(mf[i]<0){
                mf[i]=-mf[i];
                k=i;
                while(!(k%mf[i])){
                    k/=mf[i];
                }
                mf[i]=mf[k]*mf[i/k];}}
    return;}
int mu[N];
void mobius(){
    memset(mu,-1,sizeof mu); mu[1] = 1;
    for(int i = 2; i < N; i++)
        for(int j = (i << 1) ; j < N; j += i)mu[j] -= mu[i];
}

```

## 7.18 nCr mod $p^a$

```

LL F[1000009];
void pre(LL mod,LL pp){ // mod is pp^a, pp is prime
    F[0] = 1;
    REPL(i,1,mod){ // we keep in F factorial with
        // the terms which are coprime with pp
        if(i%pp!= 0) F[i]=(F[i-1]*i)%mod;
        else F[i]=F[i-1];
    }
}
LL fact2(LL nn,LL mod){
    LL cycle = nn/mod;
    LL n2=nn%mod;
    return (bigmod(F[mod],cycle,mod)*F[n2])%mod;
}
// returns highest power of pp that divides N and the coprime
// with pp part of N! %mod
PLL fact(LL N,LL pp,LL mod){
    LL nn = N; LL ord = 0;
    while(nn > 0){nn /= pp;ord += nn;}
    LL ans = 1; nn = N;
    while(nn > 0){ ans =(ans*fact2(nn,mod))%mod;
        nn/=pp;}
    return MP(ord,ans);
}
LL ncrp(ULL n,ULL r,LL prm,LL pr){ //ncr mod prm^pr
    LL mod=bigmod(prm,pr,INF),temp;
    pre(mod,prm);
}

```

```

PLL x=fact(n,prm,mod),y=fact(r,prm,mod),z=fact(n-r,prm,mod);
LL guun=x.second*modInverse(y.second,mod,prm);
guun%=mod;guun*=modInverse(z.second,mod,prm);
guun%=mod;
LL guun2=x.first-y.first-z.first;
guun*=bigmod(prm,guun2,mod);
guun%=mod;
return guun;
}

```

## 7.19 ntt

```

7340033 = 7 * 2^20, G = 3
645922817 = 77 * 2^23, G = 3
897581057 = 107 * 2^23, G = 3
998244353 = 119 * 2^23, G = 3
namespace NTT {
    vector<int> perm, wp[2];
    const int mod = 998244353, G = 3; ///G is the primitive root
    of M
    int root, inv, N, invN;
    int power(int a, int p) {
        int ans = 1;
        while (p) {
            if (p & 1) ans = (1LL*ans*a)%mod;
            a = (1LL*a*a)%mod;
            p >>= 1;
        }
        return ans;}
    void precalculate(int n) {
        assert( (n&(n-1)) == 0 && (mod-1)%n==0);
        N = n;
        invN = power(N, mod-2);
        perm = wp[0] = wp[1] = vector<int>(N);

        perm[0] = 0;
        for (int k=1; k<N; k<=1)
            for (int i=0; i<k; i++) {
                perm[i] <= 1;
                perm[i+k] = 1 + perm[i];}
        root = power(G, (mod-1)/N);
        inv = power(root, mod-2);
        wp[0][0]=wp[1][0]=1;
        for (int i=1; i<N; i++) {
            wp[0][i] = (wp[0][i-1]*1LL*root)%mod;
            wp[1][i] = (wp[1][i-1]*1LL*inv)%mod;}}
    void fft(vector<int> &v, bool invert = false) {
        if (v.size() != perm.size()) precalculate(v.size());
        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);
        for (int len = 2; len <= N; len *= 2) {
            for (int i=0, d = N/len; i<N; i+=len) {
                for (int j=0, idx=0; j<len/2; j++, idx += d) {
                    int x = v[i+j];
                    int y = (wp[invert][idx]* 1LL*v[i+j+len/2])%mod;
                    v[i+j] = (x+y)%mod ? x+y-mod : x+y;
                    v[i+j+len/2] = (x-y)%mod ? x-y : x-y+mod;}}
            if (invert) {
                for (int &x : v) x = (x*1LL*invN)%mod;}}
        vector<int> multiply(vector<int> a, vector<int> b) {
            int n = 1;
            while (n < a.size()+ b.size()) n<=1;
            a.resize(n);
            b.resize(n);
            fft(a);
            fft(b);
            for (int i=0; i<n; i++) a[i] = (a[i] * 1LL * b[i])%mod;
            fft(a, true);
            return a;}};
}

```

## 7.20 primality-test

```

using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1; base %= mod;
    while (e) {
        if (e & 1) result = (u128)result * base % mod;
        base = (u128)base * base % mod; e >>= 1;
    }
}

```

```

}
return result;
}
bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1) return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1) return false;
    }
    return true;
};
// returns true if n is prime, else returns false.
bool MillerRabin(u64 n) {
    if (n < 2) return false;
    int r = 0; u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1; r++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
        if (n == a) return true;
        if (check_composite(n, a, d, r)) return false;
    }
    return true;
}
}

```

## 7.21 prime counting function

```

#define MAXN 500
#define MAXM 100010
#define MAXP 666666
#define MAX 10000010
#define ll long long int
#define chkbit(ar,i) (((ar[(i)>>6])&(1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i)>>1)&31))))
#define isprime(x) (((x)&&((x)&1)&&(!chkbit(ar,(x))))||((x)==2))
namespace pcf{
    long long dp[MAXN][MAXM];
    unsigned int ar[(MAX>>6)+5] = {0};
    int len=0, primes[MAXP], counter[MAX];
    void Sieve(){ setbit(ar,0), setbit(ar,1);
        for (int i=3;(i*i)<MAX;i++,i++){
            if(!chkbit(ar, i)){ int k=i<<1;
                for(int j=(i*i);j<MAX;j+=k) setbit(ar,j);
            }
        }
        for(int i=1;i<MAX;i++){ counter[i]=counter[i - 1];
            if(isprime(i)) primes[len++]=i, counter[i]++;
        }
    }
    void init(){
        Sieve();
        for(int n=0;n<MAXN;n++){
            for(int m=0;m<MAXM;m++){
                if(!n) dp[n][m]=m;
                else dp[n][m]=dp[n-1][m]-dp[n-1][m/primes[n-1]];
            }
        }
    }
    ll phi(ll m,int n){
        if(n==0) return m; if(primes[n-1]>=m) return 1;
        if(m<MAXM && n<MAXN) return dp[n][m];
        return phi(m,n-1) - phi(m/primes[n-1],n-1);
    }
    ll Lehmer(long long m){
        if(m<MAX) return counter[m];
        ll w,res=0; int i,a,s,c,x,y;
        s=sqrt(0.9+m), y=cbrt(0.9+m);
        a=counter[y], res=phi(m,a)+a-1;
        for(i=a;primes[i]<=s;i++)
            res=res-Lehmer(m/primes[i])+Lehmer(primes[i])-1;
        return res;
    }
}
}

```

## 8 String

### 8.1 Hashing

```

11 fmod(11 a, 11 b, int md=mods[0]) {
    unsigned long long x = (long long) a * b;
    unsigned xh = (unsigned) (x >> 32), xl = (unsigned) x, d, m;
    asm(
        "div %4; \n\t"
        : "=a" (d), "=d" (m)
        : "d" (xh), "a" (xl), "r" (md)
    );
    return m;
}
void Build1(const string &str) {
    for(11 i = str.size() - 1; i >= 0; i--){
        hsh[i] = fmod(hsh[i + 1], bases[j], mods[j]) + str[i];
        if (hsh[i] > mods[j]) hsh[i] -= mods[j];
    }
}
11 getSingleHash(11 i, 11 j) {
    assert(i <= j);
    11 tmp1 = (hsh[i] - fmod(hsh[j+1], pwbase[0][j-i+1]));
    if(tmp1 < 0) tmp1 += mods[0]; return tmp1;
}

```

## 8.2 KMP

```

int failure[1000001];
void build_failure(string &str)
{
    int i, j, k;
    int cur;
    memset(failure, 0, sizeof failure);
    failure[0] = 0;
    failure[1] = 0;
    for(i=2; i<=str.length(); i++){
        cur=i-1;
        while(cur!=0){
            if(str[failure[cur]]==str[i-1]){
                failure[i]=failure[cur]+1;
                break;
            }
            cur=failure[cur];
        }
    }
}
int kmp(string &text, string &pat)
{
    int i, j, k;
    int cur=0;
    int ocur=0;
    for(i=0; i<text.length(); i++){
        if(cur==pat.length()){
            ocur++;
        }
        while(cur and text[i]!=pat[cur]){
            cur=failure[cur];
        }
        if(text[i]==pat[cur])
            cur++;
    }
    if(cur==pat.length())
        ocur++;
    return ocur;
}

```

## 8.3 aho-corasick

```

const int K = 26;
struct Vertex {
    int next[K]; bool leaf = false; int p = -1; char pch;
    int link = -1; int go[K];
    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};
vector<Vertex> t(1);
void add_string(string const& s) { int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size(); t.emplace_back(v, ch);
        }
    }
}

```

```

        v = t[v].next[c];
    }
    t[v].leaf = true;
}
int go(int v, char ch);
int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0) t[v].link = 0;
        else t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}
int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1) t[v].go[c] = t[v].next[c];
        else t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}

```

## 8.4 manacher

```

char s[MAX]; vector<int> d1(n); vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++){
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
    while (0 <= i - k && i + k < n && s[i - k] == s[i + k])
        { k++; }
    d1[i] = k--;
    if (i + k > r) { l = i - k; r = i + k; }
}
for (int i = 0, l = 0, r = -1; i < n; i++){
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k])
        { k++; }
    d2[i] = k--;
    if (i + k > r) { l = i - k - 1; r = i + k; }
}

```

## 8.5 palindromic tree

```

struct node {
    int next[26]; int len; int sufflink; int num; };
int len; char s[MAXN]; node tree[MAXN];
int num; // node 1 - root with len -1, node 2 - root with len 0
int suff; // max suffix palindrome
bool addLetter(int pos) {
    int cur = suff, curlen = 0; int let = s[pos] - 'a';
    while (true) {
        curlen = tree[cur].len;
        if (pos-1-curlen >= 0 && s[pos-1-curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let]; return false;
    }
    num++; suff = num; tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;
    if (tree[num].len == 1) { tree[num].sufflink = 2;
        tree[num].num = 1; return true;
    }
    while (true) {
        cur = tree[cur].sufflink; curlen = tree[cur].len;
        if (pos-1-curlen >= 0 && s[pos-1-curlen] == s[pos]) {
            tree[num].sufflink = tree[cur].next[let]; break;
        }
    }
    tree[num].num = 1 + tree[tree[num].sufflink].num; return true;
}
void initTree() {
    num = 2; suff = 2; // memset tree must
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}
int main() { gets(s); len = strlen(s); initTree();
    for (int i = 0; i < len; i++) { addLetter(i);
        ans += tree[suff].num; }
    cout << ans << endl; return 0;
}

```

## 8.6 suffix array da

```

/* sa => ith smallest suffix of the string
rak => rak[i] indicates the position of suffix(i) in the suffix
array; height => height[i] indicates the LCP of i-1 and i th
suffix; LCP of suffix(i) & suffix(j) = { L = rak[i], R = rak[j]
, min(height[L+1, R])};*/
const int maxn = 5e5+5;
int wa[maxn], wb[maxn], wv[maxn], wc[maxn];
int r[maxn], sa[maxn], rak[maxn], height[maxn], dp[maxn][22],
jump[maxn], SIGMA = 0;
int cmp(int *r, int a, int b, int l)
{ return r[a]==r[b] && r[a+l]==r[b+l]; }
void da(int *r, int *sa, int n, int m) {
    int i, j, p, *x=wa, *y=wb, *t;
    for(i=0; i<m; i++) wc[i]=0;
    for(i=0; i<n; i++) wc[x[i]=r[i]]++;
    for(i=1; i<m; i++) wc[i] += wc[i-1];
    for(i=n-1; i>=0; i--) sa[wc[i]] = i;
    for(j=1, p=1; p<n; j*=2, m=p){
        for(p=0, i=n-j; i<n; i++) y[p++] = i;
        for(i=0; i<n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
        for(i=0; i<n; i++) wv[i] = x[y[i]];
        for(i=0; i<n; i++) wc[i]=0;
        for(i=0; i<n; i++) wc[wv[i]]++;
        for(i=1; i<m; i++) wc[i] += wc[i-1];
        for(i=n-1; i>=0; i--) sa[wc[i]] = y[i];
        for(t=x, x=y, y=t, p=1, x[sa[0]] = 0, i=1; i<n; i++)
            x[sa[i]] = cmp(y, sa[i-1], sa[i], j) ? p-1 : p++;
    }
}
void calheight(int *r, int *sa, int n) {
    int i, j, k=0;
    for(i=1; i<n; i++) rak[sa[i]] = i;
    for(i=0; i<n; height[rak[i++]] = k) {
        for(k?k--:0, j=sa[rak[i]-1]; r[i+k] == r[j+k]; k++);
    }
}
void initRMQ(int n) {
    for(int i=0; i<=n; i++) dp[i][0] = height[i];
    for(int j=1; (1<<j) <= n; j++) {
        for(int i=0; i+(1<<j)-1 <= n; i++) {
            dp[i][j] = min(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
        }
    }
    for(int i=1; i<=n; i++) {
        int k=0; while((1<<(k+1)) <= i) k++; jump[i] = k;
    }
}
int askRMQ(int L, int R) {
    int k = jump[R-L+1];
    return min(dp[L][k], dp[R - (1<<k) + 1][k]);
}
int main() {
    scanf("%s", s); int n = strlen(s);
    for(int i=0; i<n; i++) {
        r[i] = s[i] - 'a' + 1; SIGMA = max(SIGMA, r[i]);
    }
    r[n] = 0; da(r, sa, n+1, SIGMA + 1);
    calheight(r, sa, n);
    /* don't forget SIGMA + 1. It will ruin you.*/
}

```

## 8.7 suffix-automaton

```

class SuffixAutomaton {
    bool complete; int last;
    set<char> alphabet;
    struct state {
        int len, link, endpos, first_pos, snas, height;
        long long substrings, sublen;
        bool is_clone;
        map<char, int> next;
        vector<int> inv_link;
        state(int leng=0, int li=0) {
            len=leng; link=li;
            first_pos=-1; substrings=0;
            sublen=0; // length of all substrings
            snas=0; // shortest_non_appearing_string
            endpos=1; is_clone=false; height=0;
        }
    };
};

```



```

    }
};
vector<state> st;
void process(int node){
    map<char, int> ::iterator mit;
    st[node].substrings=1;
    st[node].snas=st.size();
    if((int) st[node].next.size()<(int) alphabet.size())
        st[node].snas=1;
    for(mit=st[node].next.begin(); mit!=st[node].next.end();++mit){
        if(st[mit->second].substrings==0) process(mit->second);
    }
    st[node].height=max(st[node].height,1+st[mit->second].height);
    st[node].substrings=
        st[node].substrings+st[mit->second].substrings;
    st[node].sublen=st[node].sublen
        +st[mit->second].sublen+st[mit->second].substrings;
    st[node].snas=min(st[node].snas,
        1+st[mit->second].snas);
    }
    if(st[node].link!=-1)
        st[st[node].link].inv_link.push_back(node);
}
void set_suffix_links(int node){
    int i;
    for(i=0; i<st[node].inv_link.size(); i++){
        set_suffix_links(st[node].inv_link[i]);
        st[node].endpos=
            st[node].endpos+st[st[node].inv_link[i]].endpos;
    }
}
void output_all_occurrences(int v,int
    P_length,vector<int>&pos){
    if (!st[v].is_clone)
        pos.push_back(st[v].first_pos - P_length + 1);
    for (int u : st[v].inv_link)
        output_all_occurrences(u, P_length, pos);
}
void kth_smallest(int node,int k,vector<char> &str){
    if(k==0) return;
    map<char, int> ::iterator mit;
    for(mit=st[node].next.begin(); mit!=st[node].next.end();++mit){
        if(st[mit->second].substrings<k)k=k-st[mit->second].substrings;
        else{
            str.push_back(mit->first);
            kth_smallest(mit->second,k-1,str);
            return;
        }
    }
}
int find_occurrence_index(int node,int index,vector<char>&str){
    if(index==str.size()) return node;
    if(!st[node].next.count(str[index])) return -1;
    else return find_occurrence_index(st[node].next[str[index]]
        ,index+1,str);
}
}
void klen_smallest(int node,int k,vector<char> &str){
    if(k==0) return;
    map<char, int> ::iterator mit;
    for(mit=st[node].next.begin(); mit!=st[node].next.end();++mit){
        if(st[mit->second].height>=k-1){
            str.push_back(mit->first);
            klen_smallest(mit->second,k-1,str);
            return;
        }
    }
}
}
void minimum_non_existing_string(int node,vector<char> &str){
    map<char, int> ::iterator mit;
    set<char>::iterator sit;

```

```

    for(mit=st[node].next.begin(),sit=alphabet.begin();
        sit!=alphabet.end(); ++sit,++mit){
        if(mit==st[node].next.end()||mit->first!=(*sit)){
            str.push_back(*sit);
            return;
        }
        else if(st[node].snas==1+st[mit->second].snas){
            str.push_back(*sit);
            minimum_non_existing_string(mit->second,str);
            return;
        }
    }
}
void find_substrings(int node,int index,vector<char> &str,
    vector<pair<long long,long long> > &sub_info){
    sub_info.push_back(make_pair(st[node].substrings,
        st[node].sublen+st[node].substrings*index));
    if(index==str.size()) return;
    if(st[node].next.count(str[index])){ find_substrings(
        st[node].next[str[index]],index+1,str,sub_info);return;
    }
    else{
        sub_info.push_back(make_pair(0,0));
    }
}
void check(){
    if(!complete){
        process(0);
        set_suffix_links(0);
        int i;
        complete=true;
    }
}
public:
    SuffixAutomaton(set<char> &alpha){
        st.push_back(state(0,-1));
        last=0;
        complete=false;
        set<char>::iterator sit;
        for(sit=alpha.begin(); sit!=alpha.end(); sit++){
            alphabet.insert(*sit);
            st[0].endpos=0;
        }
    }
    void sa_extend(char c){
        int cur = st.size();
        st.push_back(state(st[last].len + 1));
        st[cur].first_pos=st[cur].len-1;
        int p = last;
        while (p != -1 && !st[p].next.count(c)){
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1){
            st[cur].link = 0;
        }
        else{
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len){
                st[cur].link = q;
            }
            else{
                int clone = st.size();
                st.push_back(state(st[p].len + 1,st[q].link));
                st[clone].next = st[q].next;
                st[clone].is_clone=true;
                st[clone].endpos=0;
                st[clone].first_pos=st[q].first_pos;
            }
        }
    }

```

```

        while (p != -1 && st[p].next[c] == q){
            st[p].next[c] = clone; p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
    }
    }
    last = cur;
    complete=false;
}
~SuffixAutomaton(){
    int i;
    for(i=0; i<st.size(); i++){
        st[i].next.clear();
        st[i].inv_link.clear();
    }
    st.clear();
    alphabet.clear();
}
void kth_smallest(int k,vector<char> &str){
    check();
    kth_smallest(0,k,str);
}
int FindFirstOccurrenceIndex(vector<char> &str){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind==0) return -1;
    else if(ind==-1) return st.size();
    else return st[ind].first_pos+1-(int) str.size();
}
void FindAllOccurrenceIndex(vector<char> &str,vector<int>&pos){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind!=-1) output_all_occurrences(ind,str.size(),pos);
}
int Occurrences(vector<char> &str){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind==0) return 1;
    else if(ind==-1) return 0;
    else return st[ind].endpos;
}
void klen_smallest(int k,vector<char> &str){
    check();
    if(st[0].height>=k) klen_smallest(0,k,str);
}
void minimum_non_existing_string(vector<char> &str){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind!=-1) minimum_non_existing_string(ind,str);
}
}
};

```

## 8.8 z-algorithm

```

vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}

```