

Contents

1 Basic	1	9 Else	23
1.1 Shell script	1	9.1 Mo's Alogrithm(With modification)	23
1.2 Default code	1	9.2 Mo's Alogrithm On Tree	23
1.3 vimrc	1	9.3 DynamicConvexTrick	24
1.4 readchar	1		
1.5 Black Magic	1	10 JAVA	24
1.6 Texas hold'em	1	10.1Big number	24
2 Graph	2	1 Basic	
2.1 BCC Vertex	2	1.1 Shell script	
2.2 Bridge	2	g++ -O2 -std=c++14 -Dbbq -Wall -Wextra -Wshadow -o \$1	
2.3 Strongly Connected Components	2	\$1.cpp	
2.4 MinimumMeanCycle	2	chmod +x compile.sh	
2.5 Virtual Tree	2	1.2 Default code	
2.6 Maximum Clique	3	#include<bits/stdc++.h>	
2.7 MinimumSteinerTree	3	#include<ext/pb_ds/assoc_container.hpp>	
2.8 Dominator Tree	3	using namespace std;	
2.9 Minimum Arborescence	4	using namespace __gnu_pbds;	
2.10Theory	4	typedef long long ll;	
3 Data Structure	4	typedef pair<int,int> pii;	
3.1 Treap	4	typedef pair<ll,ll> pll;	
3.2 Leftist Tree	4	#define X first	
3.3 Heavy light Decomposition	5	#define Y second	
3.4 Smart Pointer	5	#define ET cout << "\n"	
3.5 LiChaoST	5	#define SZ(a) ((int)a.size())	
3.6 link cut tree	6	#define ALL(v) v.begin(),v.end()	
3.7 KDTree	6	#define pb push_back	
4 Flow/Matching	8	#define IOS() ios_base::sync_with_stdio(0);cin.tie(0);	
4.1 Dinic	8	1.3 vimrc	
4.2 Kuhn Munkres	8	"This file should be placed at ~/.vimrc"	
4.3 MincostMaxflow	8	se nu ai hls et ru ic is sc cul	
4.4 Maximum Simple Graph Matching	9	se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a	
4.5 Minimum Weight Matching (Clique version)	9	hi cursorline cterm=none ctermbg=89	
4.6 SW-mincut	10	syntax on	
4.7 BoundedFlow	10	nnoremap <c-l> :nohl<cr>	
4.8 Gomory Hu tree	10	inoremap {<ENTER> {}<LEFT><ENTER><ENTER><UP><TAB>	
4.9 NumberofMaximalClique	11	1.4 readchar	
4.10isap	11	inline char readchar(){	
5 String	11	static const size_t bufsize = 65536;	
5.1 KMP	11	static char buf[bufsize];	
5.2 Z-value	11	static char *p = buf, *end = buf;	
5.3 Manacher	11	if (p == end) end = buf + fread_unlocked(buf, 1,	
5.4 Suffix Array	11	bufsize, stdin), p = buf;	
5.5 SAIS	12	return *p++;	
5.6 Aho-Corasick Automatan	12	}	
5.7 Smallest Rotation	13	1.5 Black Magic	
5.8 De Bruijn sequence	13	//paring heap	
5.9 SAM	13	#include <bits/stdc++.h>	
5.10PalTree	13	using namespace std;	
5.11cyclicLCS	14	#include <ext/pb_ds/priority_queue.hpp>	
6 Math	14	typedef __gnu_pbds::priority_queue<int> heap;	
6.1 ax+by=gcd	14	int main(){	
6.2 floor and ceil	14	heap h1,h2;	
6.3 Miller Rabin	14	h1.push(1); h1.push(3);	
6.4 Big number	14	h2.push(2); h2.push(4);	
6.5 Fraction	15	h1.join(h2);	
6.6 Simultaneous Equations	15	cout<<h1.size()<< '\n';	
6.7 Pollard Rho	16	cout<<h2.size()<< '\n';	
6.8 Simplex Algorithm	16	}	
6.9 chineseRemainder	16	//rb_tree	
6.10QuadraticResidue	16	#include <bits/stdc++.h>	
6.11PiCount	17		
6.12Algorithms about Primes	17		
7 Polynomial	17		
7.1 Fast Fourier Transform	17		
7.2 Number Theory Transform	17		
7.3 Fast Walsh Transform	18		
7.4 Polynomial Operation	18		
8 Geometry	18		
8.1 Default Code	18		
8.2 Convex hull	18		
8.3 External bisector	19		
8.4 Heart	19		
8.5 Polar Angle Sort	19		
8.6 Intersection of two circles	19		
8.7 Intersection of polygon and circle	20		
8.8 Intersection of line and circle	20		
8.9 Half plane intersection	20		
8.10Convexhull3D	20		
8.11CircleCover	21		
8.12DelaunayTriangulation	21		
8.13Tangent line of two circles	22		
8.14minMaxEnclosingRectangle	22		
8.15Minkowski Sum	23		

```
#include<ext/pb_ds/assoc_container.hpp>
using namespace std;
typedef long long ll;
using namespace std;
using namespace __gnu_pbds;
int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);
    tree<ll,null_type,less<ll>,rb_tree_tag,
        tree_order_statistics_node_update> st;
    tree<ll,ll,less<ll>,rb_tree_tag,
        tree_order_statistics_node_update> mp;

    st.insert(0);
    st.insert(2);
    st.insert(3);
    st.insert(4);
    cout<<*st.find_by_order(2)<<endl;
    cout<<st.order_of_key(1)<<endl;
}

//__int128_t
__int128_t,__float128_t
```

1.6 Texas hold'em

```
char suit[4]={'C','D','H','Y'},ranks[13]={'2','3','4','5',
    '6','7','8','9','T','J','Q','K','A'};
int rk[256];
/*
for(int i=0;i<13;++i)
    rk[ranks[i]]=i;
for(int i=0;i<4;++i)
    rk[suit[i]]=i;
*/
struct cards{
    vector<pii> v;
    int suit_count[4],hands;
    void reset(){v.clear(),FILL(suit_count,0),hands=-1;}
    void insert(char a,char b){//suit,rank
        ++suit_count[rk[a]];
        int flag=0;
        for(auto &i:v)
            if(i.Y==rk[b])
            {
                ++i.X,flag=1;
                break;
            }
        if(!flag) v.pb(pii(1,rk[b]));
    }
    void insert(string s){insert(s[0],s[1]);}
    void ready(){
        int Straight=0,Flush=(max_element(suit_count,
            suit_count+4))==5;
        sort(ALL(v),[](ii a,ii b){return a>b;});
        if(SZ(v)==5&&v[0].Y==v[1].Y+1&&v[1].Y==v[2].Y+1&&v[2].Y==v[3].Y+1&&v[3].Y==v[4].Y+1)
            Straight=1;
        else if(SZ(v)==5&&v[0].Y==12&&v[1].Y==3&&v[2].Y==2&&v[3].Y==1&&v[4].Y==0)
            v[0].Y=3,v[1].Y=2,v[2].Y=1,v[3].Y=0,v[4].Y=-1,
            Straight=1;
        if(Straight&&Flush) hands=1;
        else if(v[0].X==4) hands=2;
        else if(v[0].X==3&&v[1].X==2) hands=3;
        else if(Flush) hands=4;
        else if(Straight) hands=5;
        else if(v[0].X==3) hands=6;
        else if(v[0].X==2&&v[1].X==2) hands=7;
        else if(v[0].X==2) hands=8;
        else hands=9;
    }
    bool operator>(const cards &a)const{
        if(hands==a.hands) return v>a.v;
        return hands<a.hands;
    }
};
```

2 Graph

2.1 BCC Vertex

```
vector<int> G[N];// 1-base
vector<int> bcc[N];
int low[N],vis[N],Time;
int bcc_id[N],bcc_cnt;// 1-base
bool is_cut[N];//whether is av
int st[N],top;
void dfs(int u,int pa=-1){
    int child=0;
    low[u]=vis[u]=++Time;
    st[top++]=u;
    for(int v:G[u])
        if(!vis[v]){
            dfs(v,u),++child;
            low[u]=min(low[u],low[v]);
            if(vis[u]<=low[v]){
                is_cut[u]=1;
                bcc[++bcc_cnt].clear();
                int t;
                do{
                    bcc_id[t=st[--top]]=bcc_cnt;
                    bcc[bcc_cnt].push_back(t);
                }while(t!=v);
                bcc_id[u]=bcc_cnt;
                bcc[bcc_cnt].push_back(u);
            }
        }
    else if(vis[v]<vis[u]&&v!=pa)
        low[u]=min(low[u],vis[v]);
    if(pa!=-1&&child<2)is_cut[u]=0;
}
inline void bcc_init(int n){
    Time=bcc_cnt=top=0;
    for(int i=1;i<=n;++i)
        G[i].clear(),vis[i]=0,is_cut[i]=0,bcc_id[i]=0;
}
```

2.2 Bridge

```
struct Bridge{//1-base
    int n,low[MAXN],dfn[MAXN],t;
    vector<pii> G[MAXN],edge;
    vector<bool> bri;
    void init(int n){n=n;
        for(int i=1;i<=n;++i) G[i].clear();
    }
    void add_edge(int a,int b){
        int x=edge.size();
        G[a].pb(pii(b,x)),G[b].pb(pii(a,x)),edge.pb(pii(a,b));
    }
    void dfs(int x,int f){
        dfn[x]=low[x]=++t;
        for(auto i:G[x])
            if(!dfn[i.X])
                dfs(i.X,i.Y),low[x]=min(low[x],low[i.X]);
            else if(i.Y!=f) low[x]=min(low[x],low[i.X]);
        if(low[x]==dfn[x] && f!=-1) bri[f]=1;
    }
    void get_edge(){
        bri.clear(),bri.resize(edge.size(),0);
        FILL(low,0),FILL(dfn,0),t=0;
        for(int i=1;i<=n;++i)
            if(!dfn[i]) dfs(i,-1);
    }
};
```

2.3 Strongly Connected Components

```
struct Strongly_CC{//1-base
    int low[MAXN],vis[MAXN],bln[MAXN],sz[MAXN],n,t,nScc;
    bitset<MAXN> instack;
    stack<int> st;
```

```

vector<int> G[MAXN], SCC[MAXN];
void init(int _n){n=_n;
    for(int i=1; i<=n; ++i)
        G[i].clear();
}
void add_edge(int a, int b){
    G[a].pb(b);
}
void dfs(int u){
    vis[u]=low[u]=++t;
    instack[u]=1, st.push(u);
    for(int i:G[u])
        if(!vis[i]) dfs(i), low[u]=min(low[i], low[u]);
        else if(instack[i]&&vis[i]<vis[u])
            low[u]=min(low[u], vis[i]);
    if(low[u]==vis[u]){
        int tmp;
        do{
            tmp=st.top(), st.pop();
            instack[tmp]=0, bln[tmp]=nScc;
        }while(tmp!=u);
        ++nScc;
    }
}
void solve(){
    FILL(low, 0), FILL(vis, 0), FILL(bln, 0), FILL(sz, 0), t=
        nScc=0;
    for(int i=1; i<=n; ++i) SG[i].clear();
    for(int i=1; i<=n; ++i)
        if(!vis[i]) dfs(i);
    for(int i=1; i<=n; ++i){
        ++sz[bln[i]], SCC[bln[i]].pb(i);
    }
};

```

2.4 MinimumMeanCycle

```

ll road[MAXN][MAXN]; //input here
struct MinimumMeanCycle{//0-base
    ll dp[MAXN+5][MAXN], n;
    pll solve(){//watch out overflow
        ll k=1, a=-1, b=-1, ta, tb, L=n+1;
        for(ll i=2; i<=L; ++i)
            for(ll k=0; k<n; ++k)
                for(ll j=0; j<n; ++j)
                    dp[i][j]=min(dp[i-1][k]+road[k][j], dp[i][j]);
        for(ll i=0; i<n; ++i)
        {
            if(dp[L][i]>=INF) continue;
            ta=0, tb=1;
            for(ll j=1; j<n; ++j)
                if(dp[j][i]<INF&&ta*(L-j)<(dp[L][i]-dp[j][i])*
                    tb)
                    ta=dp[L][i]-dp[j][i], tb=L-j;
            if(ta==0) continue;
            if(a==-1) a=ta, b=tb;
            else if(a*tb>ta*b) a=ta, b=tb;
        }
        if(a!=-1) return k=__gcd(a, b), MP(a/k, b/k);
        return MP(-1LL, -1LL);
    }
    void init(int _n){n=_n;
        for(ll i=0; i<n; ++i)
            for(ll j=0; j<n; ++j)
                dp[i+2][j]=INF;
    }
};

```

2.5 Virtual Tree

```

void insert(int x){
    if(top==-1)
        return st[++top]=x, void();
    int p=LCA(st[top], x);
    if(p==st[top])
        return st[++top]=x, void();
    while(dep[st[top-1]]>dep[p])
        vG[st[top-1]].pb(st[top]), --top;

```

```

    vG[p].pb(st[top]), --top;
    if(st[top]!=p) st[++top]=p;
    st[++top]=x;
}
void ending(){
    while(top>0)
        vG[st[top-1]].pb(st[top]), --top;
}
void reset(int x){
    for(int i:vG[x])
        reset(i);
    vG[x].clear();
}

```

2.6 Maximum Clique

```

struct Maximum_Clique{
    typedef bitset<MAXN> bst;
    bst N[MAXN], empty;
    int p[MAXN], n, ans;
    void BronKerbosch2(bst R, bst P, bst X){
        if(P==empty&&X==empty)
            return ans=max(ans, (int)R.count()), void();
        bst tmp=P|X;
        int u;
        if((R|P|X).count()<=ans) return;
        for(int uu=0; uu<n; ++uu){
            u=p[uu];
            if(tmp[u]==1) break;
        }
        //if (double(cLock())/CLOCKS_PER_SEC > .999) return
        ;
        bst now2=P&~N[u];
        for(int vv=0; vv<n; ++vv){
            int v=p[vv];
            if(now2[v]==1){
                R[v]=1;
                BronKerbosch2(R, P&N[v], X&N[v]);
                R[v]=0, P[v]=0, X[v]=1;
            }
        }
    }
    void init(int _n){n=_n;
        for(int i=0; i<n; ++i) N[i].reset();
    }
    void add_edge(int u, int v){N[u][v]=N[v][u]=1;}
    int solve(){//remember srand
        bst R, P, X;
        ans=0, P.flip();
        for(int i=0; i<n; ++i) p[i]=i;
        random_shuffle(p, p+n), BronKerbosch2(R, P, X);
        return ans;
    }
};

```

2.7 MinimumSteinerTree

```

// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree{//0-base
    static const int T=9, MAXN=70, INF=1e9;
    int n, dst[MAXN][MAXN], dp[1<T][MAXN], tdst[MAXN]
        ;
    int vcost[MAXN]; //the cost of vertexs
    void init(int _n){n=_n;
        for(int i=0; i<n; ++i){
            for(int j=0; j<n; ++j)
                dst[i][j]=INF;
            dst[i][i] = vcost[i] = 0;
        }
    }
    void add_edge(int ui, int vi, int wi){
        dst[ui][vi]=min(dst[ui][vi], wi);
        //dst[vi][ui]=min(dst[vi][ui], wi);
    }
    void shortest_path(){
        for(int k=0; k<n; ++k)
            for(int i=0; i<n; ++i)

```

```

        for(int j=0; j<n; ++j)
            dst[i][j]=min(dst[i][j],dst[i][k]+
                dst[k][j]);
    }
    int solve(const vector<int>& ter){
        shortest_path();
        int t=SZ(ter);
        for(int i=0; i<(1<t); ++i)
            for(int j=0; j<n; ++j)
                dp[i][j]=INF;
        for(int i=0; i<n; ++i)
            dp[0][i] = vcost[i];
        for(int msk=1; msk<(1<t); ++msk){
            if(msk==(msk&(-msk))){
                int who=__lg( msk );
                for(int i=0; i<n; ++i)
                    dp[msk][i] = vcost[ter[who]] + dst[
                        ter[who]][i];
                continue;
            }
            for(int i=0; i<n; ++i)
                for(int submsk=(msk-1)&msk; submsk; submsk=(
                    submsk-1)&msk)
                    dp[msk][i] = min(dp[msk][i], dp[submsk
                        ][i] + dp[msk^submsk][i] - vcost[i
                            ]);
            for(int i=0; i<n; ++i){
                tdst[i]=INF;
                for(int j=0; j<n; ++j)
                    tdst[i]=min(tdst[i],dp[msk][j]+dst[
                        j][i]);
            }
            for(int i=0; i<n; ++i)
                dp[msk][i]=tdst[i];
        }
        int ans=INF;
        for(int i=0; i<n; ++i)
            ans=min(ans,dp[(1<t)-1][i]);
        return ans;
    }
};

```

2.8 Dominator Tree

```

struct dominator_tree{//1-base
    int n;
    vector<int> G[MAXN],rG[MAXN];
    int pa[MAXN],dfn[MAXN],id[MAXN],dfnCnt;
    int semi[MAXN],idom[MAXN],best[MAXN];
    vector<int> tree[MAXN];//dominator_tree
    void init(int _n){
        n=_n;
        for(int i=1;i<n;++i)G[i].clear(),rG[i].clear();
    }
    void add_edge(int u,int v){
        G[u].push_back(v);
        rG[v].push_back(u);
    }
    void dfs(int u){
        id[dfn[u]]=++dfnCnt;u;
        for(auto v:G[u]) if(!dfn[v]){
            dfs(v),pa[dfn[v]]=dfn[u];
        }
    }
    int find(int y,int x){
        if(y<x)return y;
        int tmp=find(pa[y],x);
        if(semi[best[y]]>semi[best[pa[y]]])
            best[y]=best[pa[y]];
        return pa[y]=tmp;
    }
    void tarjan(int root){
        dfnCnt=0;
        for(int i=1;i<n;++i){
            dfn[i]=idom[i]=0;
            tree[i].clear();
            best[i]=semi[i]=i;
        }
        dfs(root);
        for(int i=dfnCnt;i>1;--i){

```

```

            int u=id[i];
            for(auto v:rG[u]) if(v=dfn[v]){
                find(v,i);
                semi[i]=min(semi[i],semi[best[v]]);
            }
            tree[semi[i]].push_back(i);
            for(auto v:tree[pa[i]]){
                find(v,pa[i]);
                idom[v] = semi[best[v]]==pa[i] ? pa[i] : best[v
                    ];
            }
            tree[pa[i]].clear();
        }
        for(int i=2; i<=dfnCnt; ++i){
            if(idom[i]!=semi[i]) idom[i]=idom[idom[i]];
            tree[id[idom[i]]].push_back(id[i]);
        }
    }
};

```

2.9 Minimum Arborescence

```

#define INF 0x3f3f3f3f
template<typename T>
struct zhu_liu{//0(VE)
    static const int MAXN=110;
    struct edge{
        int u,v;
        T w;
        edge(int u=0,int v=0,T w=0):u(u),v(v),w(w){}
    };
    vector<edge>E;// 0-base
    int pe[MAXN],id[MAXN],vis[MAXN];
    T in[MAXN];
    void init(){E.clear();}
    void add_edge(int u,int v,T w){
        if(u!=v)E.push_back(edge(u,v,w));
    }
    T build(int root,int n){
        T ans=0;
        for(;;){
            for(int u=0;u<n;++u)in[u]=INF;
            for(size_t i=0;i<E.size();++i)
                if(E[i].u!=E[i].v&&E[i].w<in[E[i].v])
                    pe[E[i].v]=i,in[E[i].v]=E[i].w;
            for(int u=0;u<n;++u)//no solution
                if(u!=root&&in[u]==INF)return -INF;
            int cntnode=0;
            memset(id,-1,sizeof(int)*n);
            memset(vis,-1,sizeof(int)*n);
            for(int u=0;u<n;++u){
                if(u!=root)ans+=in[u];
                int v=u;
                for(;vis[v]!=u&&id[v]==-1&&v!=root;v=E[pe[v]].u)
                    vis[v]=u;
                if(v!=root&&id[v]==-1){
                    for(int x=E[pe[v]].u;x!=v;x=E[pe[x]].u)
                        id[x]=cntnode;
                    id[v]=cntnode++;
                }
            }
            if(!cntnode)break;//no cycle
            for(int u=0;u<n;++u)if(id[u]==-1)id[u]=cntnode++;
            for(size_t i=0;i<E.size();++i){
                int v=E[i].v;
                E[i].u=id[E[i].u];
                E[i].v=id[E[i].v];
                if(E[i].u!=E[i].v)E[i].w-=in[v];
            }
            n=cntnode;
            root=id[root];
        }
        return ans;
    }
};

```

2.10 Theory

```

/*
/Maximum independent edge set/=|V|-/Minimum edge cover/
/Maximum independent set/=|V|-/Minimum vertex cover/
*/

```

3 Data Structure

3.1 Treap

```

struct node{
    int data,sz;
    node *l,*r;
    node(int k):data(k),sz(1),l(0),r(0){}
    void up(){
        sz=1;
        if(l) sz+=l->sz;
        if(r) sz+=r->sz;
    }
    void down(){
    };
    int sz(node *a){
        return a ? a->sz : 0;
    }
    node *merge(node *a,node *b){
        if(!a || !b) return a ? a : b;
        if(rand()%(sz(a)+sz(b))<sz(a))
            return a->down(),a->r=merge(a->r,b),a->up(),a;
        return b->down(),b->l=merge(a,b->l),b->up(),b;
    }
    void split(node *o,node *&a,node *&b,int k){
        if(!o) return a=b=0,void();
        o->down();
        if(o->data<=k) a=o,split(o->r,a->r,b,k),a->up();
        else b=o,split(o->l,a,b->l,k),b->up();
    }
    void split2(node *o,node *&a,node *&b,int k){
        if(sz(o)<=k) return a=o,b=0,void();
        o->down();
        if(sz(o->l)+1<=k) a=o,split2(o->r,a->r,b,k-sz(o->l)-1);
        else b=o,split2(o->l,a,b->l,k);
        o->up();
    }
    node *kth(node *o,int k){
        if(k<=sz(o->l)) return kth(o->l,k);
        if(k==sz(o->l)+1) return o;
        return kth(o->r,k-sz(o->l)-1);
    }
    int Rank(node *o,int key){
        if(o->data<key) return sz(o->l)+1+Rank(o->r,key);
        else return Rank(o->l,key);
    }
    bool erase(node *&o,int k){
        if(!o) return 0;
        if(o->data==k){
            node *t=o;
            o->down(),o=merge(o->l,o->r);
            delete t;
            return 1;
        }
        node *&t=k<o->data?o->l:o->r;
        return erase(t,k)?o->up(),1:0;
    }
    void insert(node *&o,int k){
        node *a,*b;
        split(o,a,b,k),o=merge(a,merge(new node(k),b));
    }
    void interval(node *&o,int l,int r){
        node *a,*b,*c;
        split2(o,a,b,l-1),split2(b,b,c,r);
        //operate
        o=merge(a,merge(b,c));
    }
}

```

3.2 Leftist Tree

```

struct node{
    ll v,data,sz,sum;
    node *l,*r;
    node(ll k):v(0),data(k),sz(1),l(0),r(0),sum(k){}
};
ll sz(node *p){return p ? p->sz : 0;}
ll V(node *p){return p ? p->v : -1;}
ll sum(node *p){return p ? p->sum : 0;}
node* merge(node *a,node *b){
    if(!a || !b) return a ? a : b;
    if(a->data<b->data) swap(a,b);
    a->r=merge(a->r,b);
    if(V(a->r)>V(a->l)) swap(a->r,a->l);
    a->v=V(a->r)+1,a->sz=sz(a->l)+sz(a->r)+1;
    a->sum=sum(a->l)+sum(a->r)+a->data;
    return a;
}
void pop(node *&o){
    node *tmp=o;
    o=merge(o->l,o->r);
    delete tmp;
}

```

3.3 Heavy light Decomposition

```

struct Heavy_light_Decomposition{//1-base
    int n,ulink[10005],deep[10005],mxson[10005],w[10005],
        pa[10005];
    int t,pl[10005],data[10005],dt[10005],bln[10005],edge
        [10005],et;
    vector<pii> G[10005];
    void init(int _n){n=_n,t=0,et=1;
        for(int i=1;i<=n;++i) G[i].clear(),mxson[i]=0;
    }
    void add_edge(int a,int b,int w){
        G[a].pb(pii(b,et)),G[b].pb(pii(a,et)),edge[et++]=w;
    }
    void dfs(int u,int f,int d){
        w[u]=1,pa[u]=f,deep[u]=d++;
        for(auto &i:G[u])
            if(i.X!=f){
                dfs(i.X,u,d),w[u]+=w[i.X];
                if(w[mxson[u]]<w[i.X])
                    mxson[u]=i.X;
            }
        else
            bln[i.Y]=u,dt[u]=edge[i.Y];
    }
    void cut(int u,int link){
        data[pl[u]=t++]=dt[u],ulink[u]=link;
        if(!mxson[u]) return ;
        cut(mxson[u],link);
        for(auto i:G[u])
            if(i.X!=pa[u]&&i.X!=mxson[u])
                cut(i.X,i.X);
    }
    void build(){
        dfs(1,1,1),cut(1,1),/*build*/;
    }
    int query(int a,int b){
        int ta=ulink[a],tb=ulink[b],re=0;
        while(ta!=tb)
            if(deep[ta]<deep[tb])
                /*query*/,tb=ulink[b=pa[tb]];
            else
                /*query*/,ta=ulink[a=pa[ta]];
        if(a==b) return re;
        if(pl[a]>pl[b]) swap(a,b);
        /*query*/
        return re;
    }
};

```

3.4 Smart Pointer

```

#ifndef REFERENCE_POINTER
#define REFERENCE_POINTER
template<typename T>

```

```

struct _RefCounter{
    T data;
    int ref;
    _RefCounter(const T&d=0):data(d),ref(0){}
};
template<typename T>
struct reference_pointer{
    _RefCounter<T> *p;
    T *operator->(){return &p->data;}
    T &operator*(){return p->data;}
    operator _RefCounter<T>*(){return p;}
    reference_pointer &operator=(const reference_pointer &t)
    ){
        if(p&&!--p->ref)delete p;
        p=t.p;
        p&&++p->ref;
        return *this;
    }
    reference_pointer(_RefCounter<T> *t=0):p(t){
        p&&++p->ref;
    }
    reference_pointer(const reference_pointer &t):p(t.p){
        p&&++p->ref;
    }
    ~reference_pointer(){
        if(p&&!--p->ref)delete p;
    }
};
template<typename T>
inline reference_pointer<T> new_reference(const T&nd){
    return reference_pointer<T>(new _RefCounter<T>(nd));
}
#endif
//note:
reference_pointer<int> a;
a = new_reference(5);
a = new_reference<int>(5);
a = new_reference((int)5);
reference_pointer<int> b = a;

struct P{
    int a,b;
    P(int _a,int _b):a(_a),b(_b){}
}p(2,3);
reference_pointer<P> a;
c = new_reference(P(1,2));
c = new_reference<P>(P(1,2));
c = new_reference(p);

```

3.5 LiChaoST

```

struct LiChao_min{
    struct line{
        LL m, c;
        line(LL _m=0, LL _c=0) { m = _m; c = _c; }
        LL eval(LL x) { return m * x + c; }
    };
    struct node{
        node *l, *r; line f;
        node(line v) { f = v; l = r = NULL; }
    };
    typedef node* pnode;
    pnode root; int sz;
#define mid ((l+r)>>1)
    void insert(line &v, int l, int r, pnode &nd){
        if(!nd) { nd = new node(v); return; }
        LL trl = nd->f.eval(l), trr = nd->f.eval(r);
        LL vl = v.eval(l), vr = v.eval(r);
        if(trl <= vl && trr <= vr) return;
        if(trl > vl && trr > vr) { nd->f = v; return; }
        if(trl > vl) swap(nd->f, v);
        if(nd->f.eval(mid) < v.eval(mid)) insert(v, mid + 1, r, nd->r);
        else swap(nd->f, v), insert(v, l, mid, nd->l);
    }
    LL query(int x, int l, int r, pnode &nd){
        if(!nd) return LLONG_MAX;
        if(l == r) return nd->f.eval(x);
        if(mid >= x) return min(nd->f.eval(x), query(x, l, mid, nd->l));
        else return min(nd->f.eval(x), query(x, mid + 1, r, nd->r));
    }
};

```

```

return min(nd->f.eval(x), query(x, mid + 1, r, nd->r));
}
/* -sz <= query_x <= sz */
void init(int _sz){ sz = _sz + 1; root = NULL; }
void add_line(LL m, LL c){ line v(m, c); insert(v, -sz, sz, root); }
LL query(LL x) { return query(x, -sz, sz, root); }
};

```

3.6 link cut tree

```

const int MXN = 100005;
const int MEM = 100005;
struct Splay {
    static Splay nil, mem[MEM], *pmem;
    Splay *ch[2], *f;
    int val, rev, size;
    Splay(int _val=-1) : val(_val), rev(0), size(1) {
        f = ch[0] = ch[1] = &nil;
    }
    bool isr() { return f->ch[0] != this && f->ch[1] != this; }
    int dir() { return f->ch[0] == this ? 0 : 1; }
    void setCh(Splay *c, int d){
        ch[d] = c;
        if (c != &nil) c->f = this;
        pull();
    }
    void push(){
        if (!rev) return;
        swap(ch[0], ch[1]);
        if (ch[0] != &nil) ch[0]->rev ^= 1;
        if (ch[1] != &nil) ch[1]->rev ^= 1;
        rev=0;
    }
    void pull(){
        size = ch[0]->size + ch[1]->size + 1;
        if (ch[0] != &nil) ch[0]->f = this;
        if (ch[1] != &nil) ch[1]->f = this;
    }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
Splay *nil = &Splay::nil;
void rotate(Splay *x){
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr()) p->f->setCh(x, p->dir());
    else x->f = p->f;
    p->setCh(x->ch[!d], d);
    x->setCh(p, !d);
    p->pull(); x->pull();
}
vector<Splay*> splayVec;
void splay(Splay *x){
    splayVec.clear();
    for (Splay *q=x; q=q->f){
        splayVec.push_back(q);
        if (q->isr()) break;
    }
    reverse(begin(splayVec), end(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
        else if (x->dir()==x->f->dir()) rotate(x->f), rotate(x);
        else rotate(x), rotate(x);
    }
}
int id(Splay *x) { return x - Splay::mem + 1; }
Splay* access(Splay *x){
    Splay *q = nil;
    for (;x!=nil;x=x->f){
        splay(x);
        x->setCh(q, 1);
        q = x;
    }
    return q;
}
void chroot(Splay *x){

```



```

    access(x);
    splay(x);
    x->rev ^= 1;
    x->push(); x->pull();
}
void link(Splay *x, Splay *y){
    access(x);
    splay(x);
    chroot(y);
    x->setCh(y, 1);
}
void cut_p(Splay *y) {
    access(y);
    splay(y);
    y->push();
    y->ch[0] = y->ch[0]->f = nil;
}
void cut(Splay *x, Splay *y){
    chroot(x);
    cut_p(y);
}
Splay* get_root(Splay *x) {
    access(x);
    splay(x);
    for(; x->ch[0] != nil; x = x->ch[0])
        x->push();
    splay(x);
    return x;
}
bool conn(Splay *x, Splay *y) {
    x = get_root(x);
    y = get_root(y);
    return x == y;
}
Splay* lca(Splay *x, Splay *y) {
    access(x);
    access(y);
    splay(x);
    if (x->f == nil) return x;
    else return x->f;
}
}

```

3.7 KDTree

```

template<typename T, size_t kd> //kd??????
class kd_tree{
public:
    struct point{
        T d[kd];
        inline T dist(const point &x) const{
            T ret=0;
            for(size_t i=0; i<kd; ++i) ret+=std::abs(d[i]-x.d[i]);
            return ret;
        }
        inline bool operator==(const point &p){
            for(size_t i=0; i<kd; ++i){
                if(d[i]!=p.d[i]) return 0;
            }
            return 1;
        }
        inline bool operator<(const point &b) const{
            return d[0]<b.d[0];
        }
    };
private:
    struct node{
        node *l,*r;
        point pid;
        int s;
        node(const point &p):l(0),r(0),pid(p),s(1){}
        inline void up(){
            s=(l?l->s:0)+1+(r?r->s:0);
        }
    };
    *root;
    const double alpha, loga;
    const T INF; //?????INF, ?????
    int maxn;
    struct __cmp{
        int sort_id;

```

```

        inline bool operator()(const node *x, const node *y) const{
            return operator()(x->pid, y->pid);
        }
        inline bool operator()(const point &x, const point &y) const{
            if(x.d[sort_id]!=y.d[sort_id])
                return x.d[sort_id]<y.d[sort_id];
            for(size_t i=0; i<kd; ++i){
                if(x.d[i]!=y.d[i]) return x.d[i]<y.d[i];
            }
            return 0;
        }
    } cmp;
    void clear(node *o){
        if(!o) return;
        clear(o->l);
        clear(o->r);
        delete o;
    }
    inline int size(node *o){
        return o?o->s:0;
    }
    std::vector<node*> A;
    node* build(int k, int l, int r){
        if(l>r) return 0;
        if(k==kd) k=0;
        int mid=(l+r)/2;
        cmp.sort_id=k;
        std::nth_element(A.begin()+l, A.begin()+mid, A.begin()+r+1, cmp);
        node *ret=A[mid];
        ret->l=build(k+1, l, mid-1);
        ret->r=build(k+1, mid+1, r);
        ret->up();
        return ret;
    }
    inline bool isbad(node *o){
        return size(o->l)>alpha*o->s || size(o->r)>alpha*o->s;
    }
    void flatten(node *u, typename std::vector<node*>::iterator &it){
        if(!u) return;
        flatten(u->l, it);
        *it=u;
        flatten(u->r, ++it);
    }
    inline void rebuild(node *u, int k){
        if((int)A.size()<u->s) A.resize(u->s);
        typename std::vector<node*>::iterator it=A.begin();
        flatten(u, it);
        u=build(k, 0, u->s-1);
    }
    bool insert(node *u, int k, const point &x, int dep){
        if(!u){
            u=new node(x);
            return dep<=0;
        }
        ++u->s;
        cmp.sort_id=k;
        if(insert(cmp(x, u->pid)?u->l:u->r, (k+1)%kd, x, dep-1)){
            if(!isbad(u)) return 1;
            rebuild(u, k);
        }
        return 0;
    }
    node *findmin(node *o, int k){
        if(!o) return 0;
        if(cmp.sort_id==k) return o->l?findmin(o->l, (k+1)%kd):o;
        node *l=findmin(o->l, (k+1)%kd);
        node *r=findmin(o->r, (k+1)%kd);
        if(l&&!r) return cmp(l,o)?l:o;
        if(!l&&r) return cmp(r,o)?r:o;
        if(!l&&!r) return 0;
        if(cmp(l,r)) return cmp(l,o)?l:o;
        return cmp(r,o)?r:o;
    }
    bool erase(node *u, int k, const point &x){

```

```

if(!u)return 0;
if(u->pid==x){
    if(u->r);
    else if(u->l){
        u->r=u->l;
        u->l=0;
    }else{
        delete u;
        u=0;
        return 1;
    }
    --u->s;
    cmp.sort_id=k;
    u->pid=findmin(u->r,(k+1)%kd)->pid;
    return erase(u->r,(k+1)%kd,u->pid);
}
cmp.sort_id=k;
if(erase(cmp(x,u->pid)?u->l:u->r,(k+1)%kd,x)){
    --u->s;return 1;
}return 0;
}
inline T heuristic(const T h[])const{
    T ret=0;
    for(size_t i=0;i<kd;++i)ret+=h[i];
    return ret;
}
int qM;
std::priority_queue<std::pair<T,point>>pQ;
void nearest(node *u,int k,const point &x,T *h,T &
    mndist){
    if(u==0||heuristic(h)>=mndist)return;
    T dist=u->pid.dist(x),old=h[k];
    /*mndist=std::min(mndist,dist);*/
    if(dist<mndist){
        pQ.push(std::make_pair(dist,u->pid));
        if((int)pQ.size()==qM+1){
            mndist=pQ.top().first,pQ.pop();
        }
    }
    if(x.d[k]<u->pid.d[k]){
        nearest(u->l,(k+1)%kd,x,h,mndist);
        h[k]=std::abs(x.d[k]-u->pid.d[k]);
        nearest(u->r,(k+1)%kd,x,h,mndist);
    }else{
        nearest(u->r,(k+1)%kd,x,h,mndist);
        h[k]=std::abs(x.d[k]-u->pid.d[k]);
        nearest(u->l,(k+1)%kd,x,h,mndist);
    }
    h[k]=old;
}
std::vector<point>in_range;
void range(node *u,int k,const point&mi,const point
    &ma){
    if(!u)return;
    bool is=1;
    for(int i=0;i<kd;++i)
        if(u->pid.d[i]<mi.d[i]||ma.d[i]<u->pid.d[i]){
            is=0;break;
        }
    if(is)in_range.push_back(u->pid);
    if(mi.d[k]<u->pid.d[k])range(u->l,(k+1)%kd,mi,ma);
    if(ma.d[k]>u->pid.d[k])range(u->r,(k+1)%kd,mi,ma);
}
public:
kd_tree(const T &INF,double a=0.75):root(0),alpha(a),
    loga(log2(1.0/a)),INF(INF),maxn(1){}
inline void clear(){
    clear(root),root=0,maxn=1;
}
inline void build(int n,const point *p){
    clear(root),A.resize(maxn=n);
    for(int i=0;i<n;++i)A[i]=new node(p[i]);
    root=build(0,0,n-1);
}
inline void insert(const point &x){
    insert(root,0,x,std::__lg(size(root))/loga);
    if(root->s>maxn)maxn=root->s;
}
inline bool erase(const point &p){
    bool d=erase(root,0,p);

```

```

    if(root&&root->s<alpha*maxn)rebuild();
    return d;
}
inline void rebuild(){
    if(root)rebuild(root,0);
    maxn=root->s;
}
inline T nearest(const point &x,int k){
    qM=k;
    T mndist=INF,h[kd]={};
    nearest(root,0,x,h,mndist);
    mndist=pQ.top().first;
    pQ=std::priority_queue<std::pair<T,point>>();
    return mndist;/*???x?k????*/
}
inline const std::vector<point> &range(const point&
    mi,const point&ma){
    in_range.clear();
    range(root,0,mi,ma);
    return in_range;/*???mi?ma???vector*/
}
inline int size(){return root?root->s:0;}
};

```

4 Flow/Matching

4.1 Dinic

```

struct MaxFlow{/*0-base
struct edge{
    int to,cap,flow,rev;
};
vector<edge> G[MAXN];
int s,t,dis[MAXN],cur[MAXN],n;
int dfs(int u,int cap){
    if(u==t || !cap) return cap;
    for(int &i=cur[u];i<(int)G[u].size();++i){
        edge &e=G[u][i];
        if(dis[e.to]==dis[u]+1 && e.flow!=e.cap){
            int df=dfs(e.to,min(e.cap-e.flow,cap));
            if(df){
                e.flow+=df;
                G[e.to][e.rev].flow-=df;
                return df;
            }
        }
    }
    dis[u]=-1;
    return 0;
}
bool bfs(){
    FILL(dis,-1);
    queue<int> q;
    q.push(s),dis[s]=0;
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        for(auto &u:G[tmp])
            if(!dis[u.to] && u.flow!=u.cap){
                q.push(u.to);
                dis[u.to]=dis[tmp]+1;
            }
    }
    return dis[t]!=-1;
}
int maxflow(int _s,int _t){
    s=_s,t=_t;
    int flow=0,df;
    while(bfs()){
        FILL(cur,0);
        while(df=dfs(s,INF)) flow+=df;
    }
    return flow;
}
void init(int _n){n=_n;
    for(int i=0;i<n;++i) G[i].clear();
}
void reset(){

```



```

for(int i=0;i<n;++i)
    for(auto &j:G[i])
        j.flow=0;
}
void add_edge(int u,int v,int cap){
    G[u].pb(edge{v,cap,0,(int)G[v].size()});
    G[v].pb(edge{u,0,0,(int)G[u].size()-1});
}
};

```

4.2 Kuhn Munkres

```

struct KM{// 0-base
    int w[MAXN][MAXN],hl[MAXN],hr[MAXN],slk[MAXN],n;
    int fl[MAXN],fr[MAXN],pre[MAXN],qu[MAXN],ql,q,r;
    bool vl[MAXN],vr[MAXN];
    void init(int _n){n=_n;
        for(int i=0;i<n;++i)
            for(int j=0;j<n;++j)
                w[i][j]=-INF;
    }
    void add_edge(int a,int b,int wei){
        w[a][b]=wei;
    }
    bool Check(int x){
        if(vl[x]=1,~fl[x]) return vr[qu[qr++]]=fl[x]=1;
        while(~x) swap(x,fr[fl[x]=pre[x]]);
        return 0;
    }
    void Bfs(int s){
        fill(slk,slk+n,INF);
        fill(vl,vl+n,0),fill(vr,vr+n,0);
        ql=qr=0,qu[qr++]=s,vr[s]=1;
        while(1){
            int d;
            while(ql<qr)
                for(int x=0,y=qu[ql++];x<n;++x)
                    if(!vl[x]&&slk[x]>=(d=hl[x]+hr[y]-w[x][y]))
                        if(pre[x]=y,d) slk[x]=d;
                        else if(!Check(x)) return;
            d=INF;
            for(int x=0;x<n;++x)
                if(!vl[x]&&d>slk[x]) d=slk[x];
            for(int x=0;x<n;++x){
                if(vl[x]) hl[x]+=d;
                else slk[x]-=d;
                if(vr[x]) hr[x]-=d;
            }
            for(int x=0;x<n;++x)
                if(!vl[x]&&slk[x]&&!Check(x)) return;
        }
    }
    int Solve(){
        fill(fl,fl+n,-1),fill(fr,fr+n,-1),fill(hr,hr+n,0);
        for(int i=0;i<n;++i) hl[i]=*max_element(w[i],w[i]+n);
        for(int i=0;i<n;++i) Bfs(i);
        int res=0;
        for(int i=0;i<n;++i) res += w[i][fl[i]];
        return res;
    }
};

```

4.3 MincostMaxflow

```

struct MCMF{//0-base
    struct edge{
        ll from,to,cap,flow,cost,rev;
    }*past[MAXN];
    vector<edge> G[MAXN];
    bitset<MAXN> inq;
    ll dis[MAXN],up[MAXN],s,t,mx,n;
    bool BellmanFord(ll &flow,ll &cost){
        fill(dis,dis+n,INF);
        queue<ll> q;
        q.push(s),inq.reset(),inq[s]=1;

```

```

        up[s]=mx-flow,past[s]=0,dis[s]=0;
        while(!q.empty()){
            ll u=q.front();
            q.pop(),inq[u]=0;
            if(!up[u]) continue;
            for(auto &e:G[u])
                if(e.flow!=e.cap&&dis[e.to]>dis[u]+e.cost){
                    dis[e.to]=dis[u]+e.cost,past[e.to]=&e;
                    up[e.to]=min(up[u],e.cap-e.flow);
                    if(!inq[e.to]) inq[e.to]=1,q.push(e.to);
                }
        }
        if(dis[t]==INF) return 0;
        flow+=up[t],cost+=up[t]*dis[t];
        for(ll i=t;past[i];i=past[i]->from){
            auto &e=*past[i];
            e.flow+=up[t],G[e.to][e.rev].flow-=up[t];
        }
        return 1;
    }
    ll MinCostMaxFlow(ll _s,ll _t,ll &cost){
        s=_s,t=_t,cost=0;ll flow=0;
        while(BellmanFord(flow,cost));
        return flow;
    }
    void init(ll _n,ll _mx){n=_n,mx=_mx;
        for(int i=0;i<n;++i) G[i].clear();
    }
    void add_edge(ll a,ll b,ll cap,ll cost){
        G[a].pb(edge{a,b,cap,0,cost,G[b].size()});
        G[b].pb(edge{b,a,0,0,-cost,G[a].size()-1});
    }
};

```

4.4 Maximum Simple Graph Matching

```

struct GenMatch{//1-base
    int V,pr[MAXN];
    bool el[MAXN][MAXN];
    bool inq[MAXN],inp[MAXN],inb[MAXN];
    queue<int> qe;
    int st,ed,nb,bk[MAXN],djs[MAXN],ans;
    void init(int _V){V=_V;
        for(int i=0;i<=V;++i){
            for(int j=0;j<=V;++j)
                el[i][j]=0;
            pr[i]=bk[i]=djs[i]=0;
            inq[i]=inp[i]=inb[i]=0;
        }
        ans=0;
    }
    void add_edge(int u,int v){
        el[u][v]=el[v][u]=1;
    }
    int lca(int u,int v){
        for(int i=0;i<=V;++i) inp[i]=0;
        while(1)
            if(u=djs[u],inp[u]=true,u==st) break;
            else u=bk[pr[u]];
        while(1)
            if(v=djs[v],inp[v]) return v;
            else v=bk[pr[v]];
        return v;
    }
    void upd(int u){
        int v;
        while(djs[u]!=nb){
            v=pr[u],inb[djs[u]]=inb[djs[v]]=true;
            u=bk[v];
            if(djs[u]!=nb) bk[u]=v;
        }
    }
    void blo(int u,int v){
        nb=lca(u,v);
        for(int i=0;i<=V;++i) inb[i]=0;
        upd(u),upd(v);
        if(djs[u]!=nb) bk[u]=v;
        if(djs[v]!=nb) bk[v]=u;
        for(int tu=1;tu<=V;++tu)
            if(inb[djs[tu]])

```

```

        if(djs[tu]=nb,!inq[tu])
            qe.push(tu),inq[tu]=1;
    }
    void flow(){
        for(int i=1;i<=V;++i)
            inq[i]=0,bk[i]=0,djs[i]=i;
        while(qe.size()) qe.pop();
        qe.push(st),inq[st]=1,ed=0;
        while(qe.size()){
            int u = qe.front();
            qe.pop();
            for(int v=1;v<=V;++v)
                if(e1[u][v]&&djs[u]!=djs[v]&&pr[u]!=v)
                    if((v==st)||pr[v]>0&&bk[pr[v]]>0)
                        blo(u,v);
                    else if(!bk[v])
                        if(bk[v]=u,pr[v]>0)
                            if(!inq[pr[v]])
                                qe.push(pr[v]);
                        else;
                    else
                        return ed=v,void();
        }
    }
    void aug(){
        for(int u=ed,v=w;u>0;)
            v=bk[u],w=pr[v],pr[v]=u,pr[u]=v,u=w;
    }
    int solve(){
        for(int i=0;i<=V;++i) pr[i]=0;
        for(int u=1;u<=V;++u)
            if(!pr[u])
                if(st=u,flow(),ed>0)
                    aug(),++ans;
        return ans;
    }
};

```

4.5 Minimum Weight Matching (Clique version)

```

struct Graph{//0-base (Perfect Match)
    int n,edge[MXN][MXN];
    int match[MXN],dis[MXN],onstk[MXN];
    vector<int> stk;
    void init(int _n){n=_n;
        for(int i=0;i<n;++i)
            for(int j=0;j<n;++j)
                edge[i][j]=0;
    }
    void add_edge(int u,int v,int w){
        edge[u][v]=edge[v][u]=w;
    }
    bool SPFA(int u){
        if(onstk[u]) return 1;
        stk.pb(u),onstk[u]=1;
        for(int v=0;v<n;++v)
            if(u!=v&&match[u]!=v&&!onstk[v]){
                int m=match[v];
                if(dis[m]>dis[u]-edge[v][m]+edge[u][v])
                    {
                        dis[m]=dis[u]-edge[v][m]+edge[u][v];
                        onstk[v]=1,stk.pb(v);
                        if(SPFA(m)) return 1;
                        stk.pop_back(),onstk[v]=0;
                    }
            }
        onstk[u]=0,stk.pop_back();
        return 0;
    }
    int solve(){// find a match
        for(int i=0;i<n;i+=2)
            match[i]=i+1,match[i+1]=i;
        while(1){
            int found=0;
            for(int i=0;i<n;++i) dis[i]=onstk[i]=0;
            for(int i=0;i<n;++i)
                if(stk.clear(),!onstk[i]&&SPFA(i))

```

```

                for(found=1;stk.size()>=2;){
                    int u=stk.back();
                    stk.pop_back();
                    int v=stk.back();
                    stk.pop_back();
                    match[u]=v,match[v]=u;
                }
            if(!found) break;
        }
        int ret=0;
        for(int i=0;i<n;++i) ret+=edge[i][match[i]];
        return ret>>1;
    }
};

```

4.6 SW-mincut

```

// global min cut
struct SW{ // O(V^3)
    static const int MXN = 514;
    int n,vst[MXN],del[MXN];
    int edge[MXN][MXN],wei[MXN];
    void init(int _n){
        n=_n,MEM(edge,0),MEM(del,0);
    }
    void addEdge(int u,int v,int w){
        edge[u][v]+=w,edge[v][u]+=w;
    }
    void search(int &s,int &t){
        MEM(vst,0),MEM(wei,0),s=t=-1;
        while(1){
            int mx=-1,cur=0;
            for(int i=0;i<n;++i)
                if(!del[i]&&!vst[i]&&mx<wei[i])
                    cur=i,mx=wei[i];
            if(mx==-1) break;
            vst[cur]=1,s=t,cur;
            for(int i=0;i<n;++i)
                if(!vst[i]&&!del[i]) wei[i]+=edge[cur][i];
        }
    }
    int solve(){
        int res=INF;
        for(int i=0,x,y;i<n-1;++i){
            search(x,y),res=min(res,wei[y]),del[y]=1;
            for(int j=0;j<n;++j)
                edge[x][j]=(edge[j][x]+edge[y][j]);
        }
        return res;
    }
};

```

4.7 BoundedFlow

```

struct BoundedFlow{//0-base
    struct Edge{
        int to,cap,flow,rev;
    };
    vector<Edge> G[MXN];
    int n,s,t,dis[MXN],cur[MXN],cnt[MXN];
    void init(int _n){n=_n;
        for(int i=0;i<n+2;++i) G[i].clear(),cnt[i]=0;
    }
    void add_edge(int u,int v,int lcap,int rcap){
        cnt[u]-=lcap,cnt[v]+=lcap;
        G[u].pb(Edge{v,rcap,lcap,(int)G[v].size()});
        G[v].pb(Edge{u,0,0,(int)G[u].size()-1});
    }
    void add_edge(int u,int v,int cap){
        G[u].pb(Edge{v,cap,0,(int)G[v].size()});
        G[v].pb(Edge{u,0,0,(int)G[u].size()-1});
    }
    int dfs(int u,int cap){
        if(u==t || !cap) return cap;
        for(int &i=cur[u];i<(int)G[u].size();++i){
            Edge &e=G[u][i];
            if(dis[e.to]==dis[u]+1 && e.flow!=e.cap){
                int df=dfs(e.to,min(e.cap-e.flow,cap));

```

```

        if(df){
            e.flow+=df;
            G[e.to][e.rev].flow-=df;
            return df;
        }
    }
    dis[u]=-1;
    return 0;
}
bool bfs(){
    FILL(dis,-1);
    queue<int> q;
    q.push(s),dis[s]=0;
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        for(auto &u:G[tmp])
            if(!~dis[u.to] && u.flow!=u.cap){
                q.push(u.to);
                dis[u.to]=dis[tmp]+1;
            }
    }
    return dis[t]!=-1;
}
int maxflow(int _s,int _t){
    s=_s,t=_t;
    int flow=0,df;
    while(bfs()){
        FILL(cur,0);
        while(df=dfs(s,INF)) flow+=df;
    }
    return flow;
}
bool solve(){
    int sum=0;
    for(int i=0;i<n;++i)
        if(cnt[i]>0) add_edge(n+1,i,cnt[i]),sum+=cnt[i];
        else if(cnt[i]<0) add_edge(i,n+2,-cnt[i]);
    if(sum!=maxflow(n+1,n+2)) sum=-1;
    for(int i=0;i<n;++i)
        if(cnt[i]>0) G[n+1].pop_back(),G[i].pop_back();
        else if(cnt[i]<0) G[i].pop_back(),G[n+2].pop_back();
    return sum!=-1;
}
int solve(int _s,int _t){
    add_edge(_t,_s,INF);
    if(!solve()) return -1; //invalid flow
    int x=G[_t].back().flow;
    return G[_t].pop_back(),G[_s].pop_back(),x;
}
};

```

4.8 Gomory Hu tree

```

struct Gomory_Hu_tree{//0-base
    MaxFlow Dinic;
    int n;
    vector<pii> G[MAXN];
    void init(int _n){n=_n;
        for(int i=0;i<n;++i) G[i].clear();
    }
    void solve(vector<int> &v){
        if(v.size()<=1) return;
        int s=rand()%SZ(v);
        swap(v.back(),v[s]),s=v.back();
        int t=v[rand()%SZ(v)-1];
        vector<int> L,R;
        int x=(Dinic.reset(),Dinic.maxflow(s,t));
        G[s].pb(pii(t,x)),G[t].pb(pii(s,x));
        for(int i:v)
            if(~Dinic.dis[i]) L.pb(i);
            else R.pb(i);
        solve(L),solve(R);
    }
    void build(){
        vector<int> v(n);
        for(int i=0;i<n;++i) v[i]=i;
        solve(v);
    }
};

```

```

    }
}glt; //test by BZOJ 4519
MaxFlow &Dinic=glt.Dinic;

```

4.9 NumberofMaximalClique

```

// bool g[][] : adjacent array indexed from 1 to n
void dfs(int sz){
    int i, j, k, t, cnt, best = 0;
    if(ne[sz]==ce[sz]){ if (ce[sz]==0) ++ans; return; }
    for(t=0, i=1; i<=ne[sz]; ++i){
        for (cnt=0, j=ne[sz]+1; j<=ce[sz]; ++j)
            if (!g[lst[sz][i]][lst[sz][j]]) ++cnt;
        if (t==0 || cnt<best) t=i, best=cnt;
    } if (t && best<=0) return;
    for (k=ne[sz]+1; k<=ce[sz]; ++k) {
        if (t>0){ for (i=k; i<=ce[sz]; ++i)
            if (!g[lst[sz][t]][lst[sz][i]]) break;
        swap(lst[sz][k], lst[sz][i]);
        i=lst[sz][k]; ne[sz+1]=ce[sz+1]=0;
        for (j=1; j<k; ++j)if (g[i][lst[sz][j]])
            lst[sz+1][++ne[sz+1]]=lst[sz][j];
        for (ce[sz+1]=ne[sz+1], j=k+1; j<=ce[sz]; ++j)
            if (g[i][lst[sz][j]]) lst[sz+1][++ce[sz+1]]=lst[sz][j];
        dfs(sz+1); ++ne[sz]; --best;
        for (j=k+1, cnt=0; j<=ce[sz]; ++j) if (!g[i][lst[sz][j]]) ++cnt;
        if (t==0 || cnt<best) t=k, best=cnt;
        if (t && best<=0) break;
    }
}
void work(){
    ne[0]=0; ce[0]=0;
    for(int i=1; i<=n; ++i) lst[0][++ce[0]]=i;
    ans=0; dfs(0);
}

```

4.10 isap

```

struct Maxflow {
    static const int MAXV = 20010;
    static const int INF = 1000000;
    struct Edge {
        int v, c, r;
        Edge(int _v, int _c, int _r):
            v(_v), c(_c), r(_r) {}
    };
    int s, t;
    vector<Edge> G[MAXV*2];
    int iter[MAXV*2], d[MAXV*2], gap[MAXV*2], tot;
    void init(int x) {
        tot = x+2;
        s = x+1, t = x+2;
        for(int i = 0; i <= tot; i++) {
            G[i].clear();
            iter[i] = d[i] = gap[i] = 0;
        }
    }
    void addEdge(int u, int v, int c) {
        G[u].push_back(Edge(v, c, SZ(G[v])));
        G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
    }
    int dfs(int p, int flow) {
        if(p == t) return flow;
        for(int &i = iter[p]; i < SZ(G[p]); i++) {
            Edge &e = G[p][i];
            if(e.c > 0 && d[p] == d[e.v]+1) {
                int f = dfs(e.v, min(flow, e.c));
                if(f) {
                    e.c -= f;
                    G[e.v][e.r].c += f;
                    return f;
                }
            }
        }
    }
    if( (--gap[d[p]]) == 0) d[s] = tot;
    else {
        d[p]++;
    }
}

```

```

    iter[p] = 0;
    ++gap[d[p]];
}
return 0;
}
int solve() {
    int res = 0;
    gap[0] = tot;
    for(res = 0; d[s] < tot; res += dfs(s, INF));
    return res;
}
} flow;

```

5 String

5.1 KMP

```

int F[MAXN];
vector<int> match(string A, string B){
    vector<int> ans;
    F[0]=-1, F[1]=0;
    for(int i=1, j=0; i<B.size(); F[++i]=++j){
        if(B[i]==B[j]) F[i]=F[j]; //optimize
        while(j!=-1&&B[i]!=B[j]) j=F[j];
    }
    for(int i=0, j=0; i-j+B.size()<=A.size(); ++i, ++j){
        while(j!=-1&&A[i]!=B[j]) j=F[j];
        if(j==B.size()-1) ans.pb(i-j);
    }
    return ans;
}

```

5.2 Z-value

```

const int MAXn = 1e5 + 5;
int z[MAXn];
void make_z(string s){
    int l = 0, r = 0;
    for(int i = 1; i < s.size(); i++){
        for(z[i] = max(0, min(r - i + 1, z[i - 1]));
            i + z[i] < s.size() && s[i + z[i]] == s[z[i]]; z[i]++);
        if(i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}

```

5.3 Manacher

```

int z[MAXN];
int Manacher(string tmp){
    string s = "&";
    int l=0, r=0, x, ans;
    for(char c:tmp) s.pb(c), s.pb('%');
    ans=0, x=0;
    for(int i=1; i<SZ(s); ++i){
        z[i]=r > i ? min(z[2*l-i], r-i) : 1;
        while(s[i+z[i]]==s[i-z[i]]) ++z[i];
        if(z[i]+i>r) r=z[i]+i, l=i;
    }
    for(int i=1; i<SZ(s); ++i)
        if(s[i]!='%')
            x=max(x, z[i]);
    ans=x/2*2, x=0;
    for(int i=1; i<SZ(s); ++i)
        if(s[i]!='%')
            x=max(x, z[i]);
    return max(ans, (x-1)/2*2+1);
}

```

5.4 Suffix Array

```

struct suffix_array{
    int box[MAXN], tp[MAXN], k, m;
    bool not_equ(int *ra, int a, int b, int k, int n){
        return ra[a]!=ra[b] || a+k>n || b+k>n || ra[a+k]!=ra[b+k];
    }
    void radix(int *box, int *key, int *it, int *ot, int m, int n){
        fill_n(box, m, 0);
        for(int i=0; i<n; ++i) ++box[key[i]];
        partial_sum(box, box+m, box);
        for(int i=n-1; i>=0; --i) ot[--box[key[it[i]]]]=it[i];
    }
    void make_sa(int *sa, int *ra, string s, int n){
        for(int i=0; i<n; ++i) ra[i]=s[i];
        do{
            iota(tp, tp+k, n-k, iota(sa+k, sa+n, 0));
            radix(box, ra+k, sa+k, tp+k, m, n-k);
            radix(box, ra, tp, sa, m, n);
            tp[sa[0]]=0, m=1;
            for(int i=1; i<n; ++i){
                m+=not_equ(ra, sa[i], sa[i-1], k, n);
                tp[sa[i]]=m-1;
            }
            for(int i=0; i<n; ++i) ra[i]=tp[i];
            k*=2;
        }while(k<n&&m!=n);
    }
    void make_he(int *he, int *sa, int *ra, string s, int n){
        for(int j=0, k=0; j<n; ++j){
            if(ra[j])
                for(; s[j+k]==s[sa[ra[j]-1]+k]; ++k);
            he[ra[j]]=k, k=max(0, k-1);
        }
    }
    int sa[MAXN], ra[MAXN], he[MAXN];
    void build(string s){
        FILL(sa, 0), FILL(ra, 0), FILL(he, 0);
        FILL(box, 0), FILL(tp, 0), k=1, m=256;
        make_sa(sa, ra, s, s.size());
        make_he(he, sa, ra, s, s.size());
    }
};

```

5.5 SAIS

```

const int N = 300010;
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
    bool _t[N*2];
    int _s[N*2], _sa[N*2], _c[N*2], x[N], _p[N], _q[N*2],
        hei[N], r[N];
    int operator [] (int i){ return _sa[i]; }
    void build(int *s, int n, int m){
        memcpy(_s, s, sizeof(int) * n);
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
    }
    void mkhei(int n){
        REP(i,n) r[_sa[i]] = i;
        hei[0] = 0;
        REP(i,n) if(r[i]) {
            int ans = i>0 ? max(hei[r[i-1]] - 1, 0) : 0;
            while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans++;
            hei[r[i]] = ans;
        }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t,
        int *c, int n, int z){
        bool uniq = t[n-1] = true, neq;
        int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s + n,
            lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
        memcpy(x, c, sizeof(int) * z); \
        XD; \
        memcpy(x + 1, c, sizeof(int) * (z - 1)); \

```

```

REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i]
]-1]]++ = sa[i]-1; \
memcpy(x, c, sizeof(int) * z); \
for(int i = n - 1; i >= 0; i--) if(sa[i] && t[sa[i]
]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
MS0(c, z);
REP(i,n) uniq &= ++c[s[i]] < 2;
REP(i,z-1) c[i+1] += c[i];
if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return; }
for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s[i
+1] ? t[i+1] : s[i]<s[i+1]);
MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[s[i
]]]=p[q[i]=nn++]=i);
REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1]) {
    neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
[i])*sizeof(int));
    ns[q[lst=sa[i]]]=nmxz+neq;
}
sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz
+ 1);
MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s[p[
nsa[i]]]]] = p[nsa[i]]);
}
}sa;
int H[ N ], SA[ N ];
void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // ip is int array, len is array length
    // ip[0..n-1] != 0, and ip[len] = 0
    ip[len++] = 0;
    sa.build(ip, len, 128);
    for (int i=0; i<len; i++) {
        H[i] = sa.hei[i + 1];
        SA[i] = sa._sa[i + 1];
    }
    // resulting height, sa array \in [0,len)
}

```

5.6 Aho-Corasick Automatan

```

const int len=400000,sigma=26;
struct AC_Automatan{
    int nx[len][sigma],fl[len],cnt[len],pri[len],top;
    int newnode(){
        fill(nx[top],nx[top]+sigma,-1);
        return top++;
    }
    void init(){top=1,newnode();}
    int input(string &s){//return the end_node of string
        int X=1;
        for(char c:s){
            if(!~nx[X][c-'a'])nx[X][c-'a']=newnode();
            X=nx[X][c-'a'];
        }
        return X;
    }
    void make_fl(){
        queue<int> q;
        q.push(1),fl[1]=0;
        for(int t=0;!q.empty();){
            int R=q.front();
            q.pop(),pri[t++]=R;
            for(int i=0;i<sigma;++i)
                if(~nx[R][i]){
                    int X=nx[R][i],Z=f1[R];
                    for(;Z&&!~nx[Z][i];)Z=f1[Z];
                    fl[X]=Z?nx[Z][i]:1,q.push(X);
                }
        }
    }
    void get_v(string &s){
        int X=1;
        fill(cnt,cnt+top,0);
        for(char c:s){
            while(X&&!~nx[X][c-'a'])X=f1[X];
            X=X?nx[X][c-'a']:1,++cnt[X];
        }
        for(int i=top-2;i>0;--i) cnt[f1[pri[i]]]+=cnt[pri[i]
]];
    }
}

```

```
};
```

5.7 Smallest Rotation

```

string mcp(string s){
    int n=SZ(s),i=0,j=1;
    s+=s;
    while(i<n&&j<n){
        int k=0;
        while(k<n&&s[i+k]==s[j+k]) ++k;
        if(s[i+k]<=s[j+k]) j+=k+1;
        else i+=k+1;
        if(i==j) ++j;
    }
    int ans=i<n?i:j;
    return s.substr(ans,n);
}

int z[MAXN];
int Manacher(string tmp){
    string s="&";
    int l=0,r=0,x,ans;
    for(char c:tmp) s.pb(c),s.pb('&');
    ans=0,x=0;
    for(int i=1;i<SZ(s);++i){
        z[i]=r > i ? min(z[2*i-i],r-i) : 1;
        while(s[i+z[i]]==s[i-z[i]])++z[i];
        if(z[i]+i>r)r=z[i]+i,l=i;
    }
    for(int i=1;i<SZ(s);++i)
        if(s[i]=='&')
            x=max(x,z[i]);
    ans=x/2*2,x=0;
    for(int i=1;i<SZ(s);++i)
        if(s[i]!='&')
            x=max(x,z[i]);
    return max(ans,(x-1)/2*2+1);
}

```

5.8 De Bruijn sequence

```

constexpr int MAXC = 10, MAXN = 100000 + 10;
struct DBseq {
    int C, N, K, L, buf[MAXC * MAXN]; //K <= C^N
    void dfs(int *out, int t, int p, int &ptr) {
        if (ptr >= L) return;
        if (t > N) {
            if (N % p == 0) {
                for (int i = 1; i <= p && ptr < L; ++i)
                    out[ptr++] = buf[i];
            }
        } else {
            buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
        }
        for (int j = buf[t - p] + 1; j < C; ++j)
            buf[t] = j, dfs(out, t + 1, t, ptr);
    }
    void solve(int _c, int _n, int _k, int *out) {
        int p = 0;
        C = _c, N = _n, K = _k, L = N + K - 1;
        dfs(out, 1, 1, p);
        if (p < L) fill(out + p, out + L, 0);
    }
} db;

```

5.9 SAM

```

const int MAXM = 1000010;
struct SAM{
    int tot, root, lst, mom[MAXM], mx[MAXM];
    int acc[MAXM], nxt[MAXM][33];
    int newNode(){
        int res = ++tot;
        fill(nxt[res], nxt[res]+33, 0);
    }
}

```

```

    mom[res] = mx[res] = acc[res] = 0;
    return res;
}
void init(){
    tot = 0;
    root = newNode();
    mom[root] = 0, mx[root] = 0;
    lst = root;
}
void push(int c){
    int p = lst;
    int np = newNode();
    mx[np] = mx[p]+1;
    for(; p && nxt[p][c] == 0; p = mom[p])
        nxt[p][c] = np;
    if(p == 0) mom[np] = root;
    else{
        int q = nxt[p][c];
        if(mx[p]+1 == mx[q]) mom[np] = q;
        else{
            int nq = newNode();
            mx[nq] = mx[p]+1;
            for(int i = 0; i < 33; i++)
                nxt[nq][i] = nxt[q][i];
            mom[nq] = mom[q];
            mom[q] = nq;
            mom[np] = nq;
            for(; p && nxt[p][c] == q; p = mom[p])
                nxt[p][c] = nq;
        }
    }
    lst = np;
}
void push(char *str){
    for(int i = 0; str[i]; i++)
        push(str[i]-'a'+1);
}
} sam;

```

5.10 PalTree

```

/*
 * sfail: compressed fail links with same diff
 * O(lgn): length of sfail link path
 */
const int MAXN = 1e6+10;
struct PalT{
    int tot,lst;
    int nxt[MAXN][26], len[MAXN];
    int fail[MAXN], diff[MAXN], sfail[MAXN];
    char* s;
    int newNode(int l, int _fail) {
        int res = ++tot;
        fill(nxt[res], nxt[res]+26, 0);
        len[res] = l, fail[res] = _fail;
        diff[res] = l - len[_fail];
        if (diff[res] == diff[_fail])
            sfail[res] = sfail[_fail];
        else
            sfail[res] = _fail;
        return res;
    }
    void push(int p) {
        int np = lst;
        int c = s[p]-'a';
        while (p-len[np]-1 < 0 || s[p] != s[p-len[np]-1])
            np = fail[np];
        if ((lst=nxt[np][c])) return;
        int nq_f = 0;
        if (len[np]+2 == 1) nq_f = 2;
        else {
            int tf = fail[np];
            while (p-len[tf]-1 < 0 || s[p] != s[p-len[tf]-1])
                tf = fail[tf];
            nq_f = nxt[tf][c];
        }
        int nq = newNode(len[np]+2, nq_f);
        nxt[np][c] = nq;
        lst=nq;
    }
}

```

```

void init(char* _s){
    s = _s;
    tot = 0;
    newNode(-1, 1);
    newNode(0, 1);
    diff[2] = 0;
    lst = 2;
}
} palt;

```

5.11 cyclicLCS

```

#define L 0
#define LU 1
#define U 2
const int mov[3][2]={0,-1, -1,-1, -1,0};
int al,bl;
char a[MAXL*2],b[MAXL*2]; // 0-indexed
int dp[MAXL*2][MAXL];
char pred[MAXL*2][MAXL];
inline int lcs_length(int r) {
    int i=r+al,j=bl,l=0;
    while(i>r) {
        char dir=pred[i][j];
        if(dir==LU) l++;
        i+=mov[dir][0];
        j+=mov[dir][1];
    }
    return l;
}
inline void reroot(int r) { // r = new base row
    int i=r,j=1;
    while(j<=bl&&pred[i][j]!=LU) j++;
    if(j>bl) return;
    pred[i][j]=L;
    while(i<2*al&&j<=bl) {
        if(pred[i+1][j]==U) {
            i++;
            pred[i][j]=L;
        } else if(j<bl&&pred[i+1][j+1]==LU) {
            i++;
            j++;
            pred[i][j]=L;
        } else {
            j++;
        }
    }
}
int cyclic_lcs() {
    // a, b, al, bl should be properly filled
    // note: a WILL be altered in process
    // -- concatenated after itself
    char tmp[MAXL];
    if(al>bl) {
        swap(al,bl);
        strcpy(tmp,a);
        strcpy(a,b);
        strcpy(b,tmp);
    }
    strcpy(tmp,a);
    strcat(a,tmp);
    // basic lcs
    for(int i=0;i<2*al;i++) {
        dp[i][0]=0;
        pred[i][0]=U;
    }
    for(int j=0;j<=bl;j++) {
        dp[0][j]=0;
        pred[0][j]=L;
    }
    for(int i=1;i<2*al;i++) {
        for(int j=1;j<=bl;j++) {
            if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
            if(dp[i][j-1]==dp[i][j]) pred[i][j]=L;
            else if(a[i-1]==b[j-1]) pred[i][j]=LU;
            else pred[i][j]=U;
        }
    }
    // do cyclic lcs
}

```



```

int clcs=0;
for(int i=0;i<a1;i++) {
    clcs=max(clcs,lcs_length(i));
    reroot(i+1);
}
// recover a
a[a1]='\0';
return clcs;
}

```

6 Math

6.1 ax+by=gcd

```

pii gcd(int a,int b){
    if(b==0) return pii(1,0);
    else{
        int p=a/b;
        pii q=gcd(b,a%b);
        return MP(q.Y,q.X-q.Y*p);
    }
}

```

6.2 floor and ceil

```

int floor(int a,int b){
    return a/b-(a%b&&a<0^b<0);
}
int ceil(int a,int b){
    return a/b+(a%b&&a<0^b>0);
}

```

6.3 Miller Rabin

```

// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pirmes <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool Miller_Rabin(ll a,ll n){
    if((a=a%n)==0) return 1;
    if(n&1^1) return n==2;
    ll tmp=(n-1)/((n-1)&(-n+1)),t=__lg((n-1)&(-n+1)),x=1;
    for(;tmp;tmp/=2,a=mul(a,a,n))
        if(tmp&1) x=mul(x,a,n);
    if(x==1 || x==n-1) return 1;
    while(--t)
        if((x=mul(x,x,n))==n-1) return 1;
    return 0;
}

```

6.4 Big number

```

template<typename T>
inline string to_string(const T& x){
    stringstream ss;
    return ss<<x,ss.str();
}
struct bigN:vector<ll>{
    const static int base=1000000000,width=log10(base);
    bool negative;
    bigN(const_iterator a,const_iterator b):vector<ll>(a,
        b){}
    bigN(string s){
        if(s.empty())return;
        if(s[0]=='-')negative=1,s=s.substr(1);
        else negative=0;
        for(int i=int(s.size())-1;i>=0;i-=width){
            ll t=0;
            for(int j=max(0,i-width+1);j<=i;j++)
                t=t*10+s[j]-'0';
            push_back(t);
        }
    }
}

```

```

}
trim();
}
template<typename T>
bigN(const T &x):bigN(to_string(x)){}
bigN():negative(0){}
void trim(){
    while(size()&&!back())pop_back();
    if(empty())negative=0;
}
void carry(int _base=base){
    for(size_t i=0;i<size();++i){
        if(at(i)>=0&&at(i)<_base)continue;
        if(i+1u==size())push_back(0);
        int r=at(i)%_base;
        if(r<0)r+=_base;
        at(i+1)+=(at(i)-r)/_base,at(i)=r;
    }
}
int abscmp(const bigN &b)const{
    if(size()>b.size())return 1;
    if(size()<b.size())return -1;
    for(int i=int(size())-1;i>=0;--i){
        if(at(i)>b[i])return 1;
        if(at(i)<b[i])return -1;
    }
    return 0;
}
int cmp(const bigN &b)const{
    if(negative!=b.negative)return negative?-1:1;
    return negative?-abscmp(b):abscmp(b);
}
bool operator<(const bigN&b)const{return cmp(b)<0;}
bool operator>(const bigN&b)const{return cmp(b)>0;}
bool operator<=(const bigN&b)const{return cmp(b)<=0;}
bool operator>=(const bigN&b)const{return cmp(b)>=0;}
bool operator==(const bigN&b)const{return !cmp(b);}
bool operator!=(const bigN&b)const{return cmp(b)!=0;}
bigN abs()const{
    bigN res=*this;
    return res.negative=0, res;
}
bigN operator-()const{
    bigN res=*this;
    return res.negative=!negative,res.trim(),res;
}
bigN operator+(const bigN &b)const{
    if(negative)return -(*this)+(-b);
    if(b.negative)return *this+(-b);
    bigN res=*this;
    if(b.size()>size())res.resize(b.size());
    for(size_t i=0;i<b.size();++i)res[i]+=b[i];
    return res.carry(),res.trim(),res;
}
bigN operator-(const bigN &b)const{
    if(negative)return -(*this)-(-b);
    if(b.negative)return *this+(-b);
    if(abscmp(b)<0)return -(b-(*this));
    bigN res=*this;
    if(b.size()>size())res.resize(b.size());
    for(size_t i=0;i<b.size();++i)res[i]-=b[i];
    return res.carry(),res.trim(),res;
}
bigN operator*(const bigN &b)const{
    bigN res;
    res.negative=negative!=b.negative;
    res.resize(size()+b.size());
    for(size_t i=0;i<size();++i)
        for(size_t j=0;j<b.size();++j)
            if((res[i+j]+=at(i)*b[j])>=base){
                res[i+j+1]+=res[i+j]/base;
                res[i+j]%=base;
            }
    return res.trim(),res;
}
bigN operator/(const bigN &b)const{
    int norm=base/(b.back()+1);
    bigN x=abs()*norm;
    bigN y=b.abs()*norm;
    bigN q,r;
    q.resize(x.size());
    for(int i=int(x.size())-1;i>=0;--i){

```

```

    r=r*base+x[i];
    int s1=r.size()<=y.size()?0:r[y.size()];
    int s2=r.size()<y.size()?0:r[y.size()-1];
    int d=(ll(base)*s1+s2)/y.back();
    r=r-y*d;
    while(r.negative)r=r+y,--d;
    q[i]=d;
}
q.negative=negative!=b.negative;
return q.trim(),q;
}
bigN operator%(const bigN &b)const{
    return *this-(*this/b)*b;
}
friend istream& operator>>(istream &ss, bigN &b){
    string s;
    return ss>>s, b=s, ss;
}
friend ostream& operator<<(ostream &ss, const bigN &b)
{
    if(b.negative)ss<<'-';
    ss<<(b.empty()?0:b.back());
    for(int i=int(b.size()-2);i>=0;--i)
        ss<<setw(width)<<setfill('0')<<b[i];
    return ss;
}
template<typename T>
operator T(){
    stringstream ss;
    ss<<*this;
    T res;
    return ss>>res,res;
}
};

```

6.5 Fraction

```

struct fraction{
    ll n,d;
    fraction(const ll &n=0, const ll &d=1):n(n),d(d){
        ll t=__gcd(n,d);
        n/=t,d/=t;
        if(d<0) n=-n,d=-d;
    }
    fraction operator-(const fraction &b)const{
        return fraction(-n,d);
    }
    fraction operator+(const fraction &b)const{
        return fraction(n*b.d+b*n*d,d*b.d);
    }
    fraction operator-(const fraction &b)const{
        return fraction(n*b.d-b*n*d,d*b.d);
    }
    fraction operator*(const fraction &b)const{
        return fraction(n*b.n,d*b.d);
    }
    fraction operator/(const fraction &b)const{
        return fraction(n*b.d,d*b.n);
    }
    void print(){
        cout << n;
        if(d!=1) cout << "/" << d;
    }
};

```

6.6 Simultaneous Equations

```

struct matrix { //n * (m + 1)
    static constexpr int MAXN = 110;
    int n, m;
    fraction M[MAXN][MAXN + 1], sol[MAXN];
    matrix(int n = 0, int m = 0): n(n), m(m), M(), sol() {
    }
    int solve() { //-1: inconsistent, >= 0: rank
        for (int i = 0; i < n; ++i) {
            int piv = 0;
            while (piv < m && !M[i][piv].n) ++piv;

```

```

            if (piv == m) continue;
            for (int j = 0; j < n; ++j) {
                if (i == j) continue;
                fraction tmp = -M[j][piv] / M[i][piv];
                for (int k = 0; k <= m; ++k) M[j][k] =
                    tmp * M[i][k] + M[j][k];
            }
            int rank = 0;
            for (int i = 0; i < n; ++i) {
                int piv = 0;
                while (piv < m && !M[i][piv].n) ++piv;
                if (piv == m && M[i][m].n) return -1;
                else if (piv < m) ++rank, sol[piv] = M[i][m]
                    / M[i][piv];
            }
            return rank;
        }
    };

```

6.7 Pollard Rho

```

// does not work when n is prime
ll f(ll x, ll mod){ return add(mul(x,x,mod),1,mod); }
ll pollard_rho(ll n){
    if(!(n&1)) return 2;
    while(1){
        ll y=2,x=rand()%(n-1)+1,res=1;
        for(int sz=2;res==1;y=x,sz*=2)
            for(int i=0;i<sz&&res<=1;++i)
                x=f(x,n),res=__gcd(abs(x-y),n);
        if(res!=0&&res!=n) return res;
    }
}

```

6.8 Simplex Algorithm

```

const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM], d[MAXN][MAXM];
double x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[MAXN][MAXM], double b[MAXN],
    double c[MAXM], int n, int m){
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for (int i = 0; i < n + m; ++i) ix[i] = i;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if (d[r][m] > d[i][m]) r = i;
    }
    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for (double dd;; ) {
        if (r < n) {
            int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
            d[r][s] = 1.0 / d[r][s];
            for (int j = 0; j <= m; ++j)
                if (j != s) d[r][j] *= -d[r][s];
            for (int i = 0; i <= n + 1; ++i) if (i != r) {
                for (int j = 0; j <= m; ++j) if (j != s)
                    d[i][j] += d[r][j] * d[i][s];
                d[i][s] *= d[r][s];
            }
        }
        r = -1; s = -1;
        for (int j = 0; j < m; ++j)
            if (s < 0 || ix[s] > ix[j]) {
                if (d[n + 1][j] > eps ||

```

```

        (d[n + 1][j] > -eps && d[n][j] > eps))
    s = j;
}
if (s < 0) break;
for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
    if (r < 0 ||
        (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s])
        < -eps ||
        (dd < eps && ix[r + m] > ix[i + m]))
        r = i;
}
if (r < 0) return -1; // not bounded
}
if (d[n + 1][m] < -eps) return -1; // not executable
double ans = 0;
for (int i = 0; i < m; ++i) x[i] = 0;
for (int i = m; i < n + m; ++i) { // the missing
    enumerated x[i] = 0
    if (ix[i] < m - 1) {
        ans += d[i - m][m] * c[ix[i]];
        x[ix[i]] = d[i - m][m];
    }
}
return ans;
}
}

```

6.9 chineseRemainder

```

LL solve(LL x1, LL m1, LL x2, LL m2) {
    LL g = __gcd(m1, m2);
    if ((x2 - x1) % g) return -1; // no sol
    m1 /= g; m2 /= g;
    pair<LL, LL> p = gcd(m1, m2);
    LL lcm = m1 * m2 * g;
    LL res = p.first * (x2 - x1) * m1 + x1;
    return (res % lcm + lcm) % lcm;
}

```

6.10 QuadraticResidue

```

int Jacobi(int a, int m) {
    int s = 1;
    for (; m > 1; ) {
        a %= m;
        if (a == 0) return 0;
        const int r = __builtin_ctz(a);
        if ((r & 1) && ((m + 2) & 4)) s = -s;
        a >>= r;
        if (a & m & 2) s = -s;
        swap(a, m);
    }
    return s;
}

int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    const int jc = Jacobi(a, p);
    if (jc == 0) return 0;
    if (jc == -1) return -1;
    int b, d;
    for (; ; ) {
        b = rand();
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (p + 1) >> 1; e >>= 1; ) {
        if (e & 1) {
            tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1
                * f1 % p)) % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1
            % p)) % p;
        f1 = (2LL * f0 * f1) % p;
        f0 = tmp;
    }
}

```

```

return g0;
}

```

6.11 PiCount

```

int prc[maxn];
long long phic[msz][nsz];

void sieve() {
    bitset<maxn> v;
    pr.push_back(0);
    for (int i = 2; i < maxn; ++i) {
        if (!v[i]) pr.push_back(i);
        for (int j = 1; i * pr[j] < maxn; ++j) {
            v[i * pr[j]] = true;
            if (i % pr[j] == 0) break;
        }
    }
    for (int i = 1; i < pr.size(); ++i) prc[pr[i]] = 1;
    for (int i = 1; i < maxn; ++i) prc[i] += prc[i - 1];
}

long long p2(long long, long long);
long long phi(long long m, long long n) {
    if (m < msz && n < nsz && phic[m][n] != -1) return
        phic[m][n];
    if (n == 0) return m;
    if (pr[n] >= m) return 1;
    long long ret = phi(m, n - 1) - phi(m / pr[n], n - 1);
    if (m < msz && n < nsz) phic[m][n] = ret;
    return ret;
}

long long pi(long long m) {
    if (m < maxn) return prc[m];
    long long n = pi(cbrt(m));
    return phi(m, n) + n - 1 - p2(m, n);
}

long long p2(long long m, long long n) {
    long long ret = 0;
    long long lim = sqrt(m);
    for (int i = n + 1; pr[i] <= lim; ++i) ret += pi(m
        / pr[i]) - pi(pr[i]) + 1;
    return ret;
}

```

6.12 Algorithms about Primes

```

/*
* 12721
* 13331
* 14341
* 75577
* 123457
* 222557
* 556679
* 999983
* 1097774749
* 1076767633
* 100102021
* 999997771
* 1001010013
* 1000512343
* 987654361
* 999991231
* 999888733
* 98789101
* 987777733
* 999991921
* 1010101333
* 1010102101
* 100000000039
* 10000000000037
* 2305843009213693951
* 4611686018427387847
* 9223372036854775783
* 18446744073709551557
*/

```

7 Polynomial

7.1 Fast Fourier Transform

```
template<int MAXN>
struct FFT {
    using val_t = complex<double>;
    const double PI = acos(-1);
    val_t w[MAXN];
    FFT() {
        for (int i = 0; i < MAXN; ++i) {
            double arg = 2 * PI * i / MAXN;
            w[i] = val_t(cos(arg), sin(arg));
        }
    }
    void bitrev(val_t *a, int n) {
        int i = 0;
        for (int j = 1; j < n - 1; ++j) {
            for (int k = n >> 1; (i ^ k) < k; k >>= 1)
                if (j < i) swap(a[i], a[j]);
        }
    }
    void trans(val_t *a, int n, bool inv = false) {
        bitrev(a, n);
        for (int L = 2; L <= n; L <= 1) {
            int dx = MAXN / L;
            for (int i = 0; i < n; i += L) {
                for (int j = i, x = 0; j < i + (L >> 1); ++j, x += dx) {
                    val_t tmp = a[j + (L >> 1)] * w[x];
                    a[j + (L >> 1)] = a[j] - tmp;
                    a[j] += tmp;
                }
            }
        }
        if (inv) {
            reverse(a + 1, a + n);
            for (int i = 0; i < n; ++i) a[i] /= n;
        }
    }
};
```

7.2 Number Theory Transform

```
//(2^16)+1, 65537, 3
//7*17*(2^23)+1, 998244353, 3
//1255*(2^20)+1, 1315962881, 3
//51*(2^25)+1, 1711276033, 29
template<int MAXN, LL P, LL RT> //MAXN must be 2^k
struct NTT {
    LL w[MAXN];
    LL mpow(LL a, LL n) {
        LL r = 1;
        while (n) {
            if (n & 1) r = r * a % P;
            n >>= 1, a = a * a % P;
        }
        return r;
    }
    LL inv(LL a) {
        return mpow(a, P - 2);
    }
    NTT() {
        LL dw = mpow(RT, (P - 1) / MAXN);
        w[0] = 1;
        for (int i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P;
    }
    void bitrev(LL *a, int n) {
        int i = 0;
        for (int j = 1; j < n - 1; ++j) {
            for (int k = n >> 1; (i ^ k) < k; k >>= 1)
                if (j < i) swap(a[i], a[j]);
        }
    }
};
```

```
void operator()(LL *a, int n, bool f = false) { //0
    <= a[i] < P
    bitrev(a, n);
    for (int L = 2; L <= n; L <= 1) {
        int dx = MAXN / L;
        for (int i = 0; i < n; i += L) {
            for (int j = i, x = 0; j < i + (L >> 1); ++j, x += dx) {
                LL tmp = a[j + (L >> 1)] * w[x] % P;
                if ((a[j + (L >> 1)] = a[j] - tmp) < 0) a[j + (L >> 1)] += P;
                if ((a[j] += tmp) >= P) a[j] -= P;
            }
        }
    }
    if (f) {
        reverse(a + 1, a + n);
        LL invn = inv(n);
        for (int i = 0; i < n; ++i) a[i] = a[i] * invn % P;
    }
};
```

7.3 Fast Walsh Transform

```
/*
x: a[j], y: a[j + (L >> 1)]
or: (x, y + x), (x, y - x)
and: (x + y, y), (x - y, y)
xor: (x + y, x - y), (x + y, x - y)/2
*/
void fwt(val_t *a, int n) { //or
    for (int L = 2; L <= n; L <= 1) {
        for (int i = 0; i < n; i += L) {
            for (int j = i; j < i + (L >> 1); ++j) {
                a[j + (L >> 1)] += a[j]; //inv: a[j + (L >> 1)] -= a[j];
            }
        }
    }
}
```

7.4 Polynomial Operation

```
template<int MAXN, LL P, LL RT>
struct PolyOp {
    NTT<MAXN, P, RT> ntt;
    const LL INV2 = ntt.inv(2);
    int get_sz(int n) {
        int sz = 1;
        while (sz < n) sz <= 1;
        return sz;
    }
    void mul(LL *a, int n, LL *b, int m, LL *c) {
        static LL buf1[MAXN], buf2[MAXN];
        int sz = get_sz(n + m - 1);
        copy(a, a + n, buf1), fill(buf1 + n, buf1 + sz, 0);
        copy(b, b + m, buf2), fill(buf2 + m, buf2 + sz, 0);
        ntt(buf1, sz), ntt(buf2, sz);
        for (int i = 0; i < sz; ++i) c[i] = buf1[i] * buf2[i] % P;
        ntt(c, sz, true);
    }
    void inv(LL *a, int n, LL *b) {
        static LL buf[MAXN];
        if (n == 1) return b[0] = ntt.inv(a[0]), void();
        inv(a, (n + 1) / 2, b);
        int sz = get_sz(n * 2);
        copy(a, a + n, buf), fill(buf + n, buf + sz, 0);
        fill(b + n, b + sz, 0);
        ntt(buf, sz), ntt(b, sz);
        for (int i = 0; i < sz; ++i) {
```

```

        b[i] *= (2 - b[i] * buf[i]) % P;
        if ((b[i] % P) < 0) b[i] += P;
    }
    ntt(b, sz, true), fill(b + n, b + sz, 0);
}
LL _msqrt(LL x) {
    for (LL i = 0; i <= P / 2; ++i) if (i * i % P
        == x) return i;
    return -1;
}
void sqrt(LL *a, int n, LL *b) {
    static LL invb[MAXN], buf[MAXN];
    if (n == 1) return b[0] = _msqrt(a[0]), void();
    sqrt(a, (n + 1) / 2, b);
    int sz = get_sz(n * 2);
    inv(b, n, invb);
    copy(a, a + n, buf), fill(buf + n, buf + sz, 0);
    ;
    ntt(b, sz), ntt(inv, sz), ntt(buf, sz);
    for (int i = 0; i < sz; ++i) {
        if ((b[i] += buf[i] * invb[i] % P) >= P) b[i]
            -= P;
        b[i] = b[i] * INV2 % P;
    }
    ntt(b, sz, true), fill(b + n, b + sz, 0);
}
void div(LL *a, int n, LL *b, int m, LL *q, LL *r)
{
    static LL invb[MAXN], buf[MAXN];
    if (n < m) {
        fill(q, q + m, 0), copy(a, a + n, r), fill(
            r + n, r + m, 0);
        return;
    }
    int mod_sz = n - m + 1;
    copy(b, b + m, buf), reverse(buf, buf + m);
    if (m < mod_sz) fill(buf + m, buf + mod_sz, 0);
    inv(buf, mod_sz, invb);
    copy(a, a + n, buf), reverse(buf, buf + n);
    mul(buf, mod_sz, invb, mod_sz, q);
    fill(q + mod_sz, q + n, 0), reverse(q, q +
        mod_sz);
    mul(b, m, q, mod_sz, buf);
    for (int i = 0; i < n; ++i) {
        if ((r[i] = a[i] - buf[i]) < 0) r[i] += P;
    }
}
void dx(LL *a, int n, LL *b) {
    b[n - 1] = 0;
    for (int i = 1; i < n; ++i) b[i - 1] = i * a[i]
        % P;
}
void Sx(LL *a, int n, LL *b) {
    b[0] = 0;
    for (int i = 1; i < n; ++i) b[i] = ntt.inv(i) *
        a[i - 1] % P;
}
};

```

8 Geometry

8.1 Default Code

```

typedef pair<double, double> pdd;
typedef pair<pdd, pdd> Line;
const double eps = 1e-12;
pdd operator+(const pdd &a, const pdd &b)
{ return pdd(a.X+b.X, a.Y+b.Y); }
pdd operator-(const pdd &a, const pdd &b)
{ return pdd(a.X-b.X, a.Y-b.Y); }
pdd operator*(const pdd &a, const double &b)
{ return pdd(a.X*b, a.Y*b); }
pdd operator/(const pdd &a, const double &b)
{ return pdd(a.X/b, a.Y/b); }
double dot(const pdd &a, const pdd &b)
{ return a.X*b.X + a.Y*b.Y; }
double cross(const pdd &a, const pdd &b)
{ return a.X*b.Y - a.Y*b.X; }

```

```

double abs2(const pdd &a)
{ return dot(a, a); }
double abs(const pdd &a)
{ return sqrt(dot(a, a)); }
int ori(const pdd &a, const pdd &b, const pdd &c) {
    double res = cross(b - a, c - a);
    if (fabs(res) < eps) return 0;
    return res > 0 ? 1 : -1;
}
bool collinearity(const pdd &p1, const pdd &p2, const
    pdd &p3) {
    return fabs(cross(p1 - p3, p2 - p3)) < eps;
}
bool btw(const pdd &p1, const pdd &p2, const pdd &p3) {
    if (!collinearity(p1, p2, p3)) return 0;
    return dot(p1 - p3, p2 - p3) < eps;
}
bool seg_intersect(const pdd &p1, const pdd &p2, const
    pdd &p3, const pdd &p4) {
    int a123 = ori(p1, p2, p3);
    int a124 = ori(p1, p2, p4);
    int a341 = ori(p3, p4, p1);
    int a342 = ori(p3, p4, p2);
    if (a123 == 0 && a124 == 0)
        return btw(p1, p2, p3) || btw(p1, p2, p4) || btw(p3, p4, p1)
            || btw(p3, p4, p2);
    return a123 * a124 <= 0 && a341 * a342 <= 0;
}
pdd intersect(const pdd &p1, const pdd &p2, const pdd &p3
    , const pdd &p4) {
    double a123 = cross(p2 - p1, p3 - p1);
    double a124 = cross(p2 - p1, p4 - p1);
    return (p4 * a123 - p3 * a124) / (a123 - a124);
}
pdd foot(const pdd &p1, const pdd &p2, const pdd &p3) {
    pdd tmp = p2 - p1;
    swap(tmp.X, tmp.Y), tmp.Y *= -1;
    return intersect(p1, p2, p3, p3 + tmp);
}

```

8.2 Convex hull

```

struct convex_hull {
    vector<pdd> dots;
    void add_dot(double a, double b) {
        dots.pb(pdd(a, b));
    }
    vector<pdd> hull() {
        vector<pdd> ans;
        sort(dots.begin(), dots.end());
        ans.pb(dots[0]), ans.pb(dots[1]);
        for (int i = 2; i < SZ(dots); ++i) {
            while (SZ(ans) >= 2)
                if (ori(ans[SZ(ans) - 2], ans.back(), dots[i]) <= 0)
                    ans.pop_back();
                else break;
            ans.pb(dots[i]);
        }
        for (int i = SZ(dots) - 2, t = SZ(ans); i >= 0; --i) {
            while (SZ(ans) > t)
                if (ori(ans[SZ(ans) - 2], ans.back(), dots[i]) <= 0)
                    ans.pop_back();
                else break;
            ans.pb(dots[i]);
        }
        ans.pop_back();
        return ans;
    }
};

```

8.3 External bisector

```

pdd external_bisector(pdd p1, pdd p2, pdd p3) { //213
    pdd L1 = p2 - p1, L2 = p3 - p1;
    L2 = L2 * abs(L1) / abs(L2);
    return L1 + L2;
}

```

8.4 Heart

```
pdd excenter(pdd p0,pdd p1,pdd p2,double &radius){
    p1=p1-p0,p2=p2-p0;
    double x1=p1.X,y1=p1.Y,x2=p2.X,y2=p2.Y;
    double m=2.*(x1*y2-y1*x2);
    center.X=(x1*x1*y2-x2*x2*y1+y1*y2*(y1-y2))/m;
    center.Y=(x1*x2*(x2-x1)-y1*y1*x2+x1*y2*y2)/m;
    return radius=abs(center),center+p0;
}

pdd incenter(pdd p1,pdd p2,pdd p3,double &radius){
    double a=abs(p2-p1),b=abs(p3-p1),c=abs(p3-p2);
    double s=(a+b+c)/2,area=sqrt(s*(s-a)*(s-b)*(s-c));
    pdd L1=external_bisector(p1,p2,p3),L2=
        external_bisector(p2,p1,p3);
    return radius=area/s,intersect(p1,p1+L1,p2,p2+L2),
}

pdd escenter(pdd p1,pdd p2,pdd p3){//213
    pdd L1=external_bisector(p1,p2,p3),L2=
        external_bisector(p2,p2+p2-p1,p3);
    return intersect(p1,p1+L1,p2,p2+L2);
}

pdd barycenter(pdd p1,pdd p2,pdd p3){
    return (p1+p2+p3)/3;
}

pdd orthocenter(pdd p1,pdd p2,pdd p3){
    pdd L1=p3-p2,L2=p3-p1;
    swap(L1.X,L1.Y),L1.X*=-1;
    swap(L2.X,L2.Y),L2.X*=-1;
    return intersect(p1,p1+L1,p2,p2+L2);
}
```

8.5 Polar Angle Sort

```
pdd c; //sort base
int Quadrant(pdd a){
    if(a.X>0&&a.Y>=0) return 1;
    if(a.X<=0&&a.Y>0) return 2;
    if(a.X<0&&a.Y<=0) return 3;
    if(a.X>=0&&a.Y<0) return 4;
}
bool cmp(pdd a,pdd b){
    a=a-c,b=b-c;
    if(Quadrant(a)!=Quadrant(b))
        return Quadrant(a)<Quadrant(b);
    if(cross(b,a)==0) return abs(a)<abs(b);
    return cross(b,a)>0;
}
bool cmp(pdd a,pdd b){
    a=a-c,b=b-c;
    if(fabs(atan2(a.Y,a.X)-atan2(b.Y,b.X))>eps)
        return atan2(a.Y,a.X)<atan2(b.Y,b.X);
    return abs(a)<abs(b);
}
```

8.6 Intersection of two circles

```
vector<pdd> interCircle(pdd o1 ,double r1 ,pdd o2 ,
    double r2){
    if(abs(o1-o2)<max(r1,r2)-min(r1,r2)) return {};
    double d2=abs2(o1-o2);
    double d=sqrt(d2);
    if(d>r1+r2) return {};
    pdd u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
    double A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2
        +d));
    pdd v = pdd(o1.Y-o2.Y,-o1.X+o2.X)*A/(2*d2);
    return {u+v, u-v};
}
```

8.7 Intersection of polygon and circle

```
// Divides into multiple triangle, and sum up
// test by HDU2892
const double PI=acos(-1);
double _area(pdd pa, pdd pb, double r){
    if(abs(pa)<abs(pb)) swap(pa, pb);
    if(abs(pb)<eps) return 0;
    double S, h, theta;
    double a=abs(pb),b=abs(pa),c=abs(pb-pa);
    double cosB = dot(pb,pb-pa) / a / c, B = acos(cosB);
    double cosC = dot(pa,pb) / a / b, C = acos(cosC);
    if(a > r){
        S = (C/2)*r*r;
        h = a*b*sin(C)/c;
        if (h < r && B < PI/2) S -= (acos(h/r)*r*r - h*sqrt
            (r*r-h*h));
    }
    else if(b > r){
        theta = PI - B - asin(sin(B)/r*a);
        S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
    }
    else S = .5*sin(C)*a*b;
    return S;
}
double area_poly_circle(const vector<pdd> poly,const
    pdd &o,const double r){
    double S=0;
    for(int i=0;i<SZ(poly);++i)
        S+=_area(poly[i]-o,poly[(i+1)%SZ(poly)]-o,r)*ori(0,
            poly[i],poly[(i+1)%SZ(poly)]);
    return fabs(S);
}
```

8.8 Intersection of line and circle

```
vector<pdd> line_interCircle(const pdd &p1,const pdd &
    p2,const pdd &c,const double r){
    pdd ft=foot(p1,p2,c),vec=p2-p1;
    double dis=abs(c-ft);
    if(fabs(dis-r)<eps) return vector<pdd>{ft};
    if(dis>r) return {};
    vec=vec*sqrt(r*r-dis*dis)/abs(vec);
    return vector<pdd>{ft+vec,ft-vec};
}
```

8.9 Half plane intersection

```
bool isin( Line l0, Line l1, Line l2 ){
    // Check inter(l1, l2) in l0
    pdd p = intersect(l1.X,l1.Y,l2.X,l2.Y);
    return cross(l0.Y - l0.X,p - l0.X) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (L.Y - L.X) ^ (p - L.X) > 0
 */
/* --^-- Line.X --^-- Line.Y --^-- */
vector<Line> halfPlaneInter(vector<Line> lines){
    int sz = lines.size();
    vector<double> ata(sz),ord(sz);
    for(int i=0; i<sz; ++i) {
        ord[i] = i;
        pdd d = lines[i].Y - lines[i].X;
        ata[i] = atan2(d.Y, d.X);
    }
    sort(ord.begin(), ord.end(), [&](int i,int j){
        if( fabs(ata[i] - ata[j]) < eps )
            return (cross(lines[i].Y-lines[i].X,
                lines[j].Y-lines[j].X)<0;
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for (int i=0; i<sz; ++i)
        if (!i || fabs(ata[ord[i]] - ata[ord[i-1]]) > eps)
            fin.pb(lines[ord[i]]);
    deque<Line> dq;
```



```

for (int i=0; i<SZ(fin); i++){
    while(SZ(dq)>=2&&!isin(fin[i],dq[SZ(dq)-2],dq.back()))
        dq.pop_back();
    while(SZ(dq)>=2&&!isin(fin[i],dq[0],dq[1]))
        dq.pop_front();
    dq.push_back(fin[i]);
}
while(SZ(dq)>=3&&!isin(dq[0],dq[SZ(dq)-2],dq.back()))
    dq.pop_back();
while(SZ(dq)>=3&&!isin(dq.back(), dq[0], dq[1]))
    dq.pop_front();
vector<Line> res(ALL(dq));
return res;
}

```

8.10 Convexhull3D

```

#define SIZE(X) (int(X.size()))
#define PI 3.14159265358979323846264338327950288
struct Pt{
    Pt cross(const Pt &p) const
    { return Pt(y * p.z - z * p.y, z * p.x - x * p.z, x *
        p.y - y * p.x); }
} info[N];
int mark[N][N], n, cnt;;
double mix(const Pt &a, const Pt &b, const Pt &c)
{ return a * (b ^ c); }
double area(int a, int b, int c)
{ return norm((info[b] - info[a]) ^ (info[c] - info[a]))
    ); }
double volume(int a, int b, int c, int d)
{ return mix(info[b] - info[a], info[c] - info[a], info
    [d] - info[a]); }
struct Face{
    int a, b, c; Face(){}
    Face(int a, int b, int c): a(a), b(b), c(c) {}
    int &operator [](int k)
    { if (k == 0) return a; if (k == 1) return b; return
        c; }
};
vector<Face> face;
void insert(int a, int b, int c)
{ face.push_back(Face(a, b, c)); }
void add(int v) {
    vector<Face> tmp; int a, b, c; cnt++;
    for (int i = 0; i < SIZE(face); i++) {
        a = face[i][0]; b = face[i][1]; c = face[i][2];
        if(Sign(volume(v, a, b, c)) < 0)
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] =
                mark[c][a] = mark[a][c] = cnt;
        else tmp.push_back(face[i]);
    } face = tmp;
    for (int i = 0; i < SIZE(tmp); i++) {
        a = face[i][0]; b = face[i][1]; c = face[i][2];
        if (mark[a][b] == cnt) insert(b, a, v);
        if (mark[b][c] == cnt) insert(c, b, v);
        if (mark[c][a] == cnt) insert(a, c, v);
    }
}
int Find(){
    for (int i = 2; i < n; i++) {
        Pt ndir = (info[0] - info[i]) ^ (info[1] - info[i]);
        if (ndir == Pt()) continue; swap(info[i], info[2]);
        for (int j = i + 1; j < n; j++) if (Sign(volume(0,
            1, 2, j)) != 0) {
            swap(info[j], info[3]); insert(0, 1, 2); insert
                (0, 2, 1); return 1;
        }
    } return 0; }
int main() {
    for (; scanf("%d", &n) == 1; ) {
        for (int i = 0; i < n; i++) info[i].Input();
        sort(info, info + n); n = unique(info, info + n) -
            info;
        face.clear(); random_shuffle(info, info + n);
        if (Find()) { memset(mark, 0, sizeof(mark)); cnt =
            0;
            for (int i = 3; i < n; i++) add(i); vector<Pt>
                Ndir;
            for (int i = 0; i < SIZE(face); ++i) {

```

```

                Pt p = (info[face[i][0]] - info[face[i][1]]) ^
                    (info[face[i][2]] - info[face[i][1]]);
                p = p / norm( p ); Ndir.push_back(p);
            } sort(Ndir.begin(), Ndir.end());
            int ans = unique(Ndir.begin(), Ndir.end()) - Ndir
                .begin();
            printf("%d\n", ans);
        } else printf("1\n");
    } }
double calcDist(const Pt &p, int a, int b, int c)
{ return fabs(mix(info[a] - p, info[b] - p, info[c] - p)
    ) / area(a, b, c); }
//compute the minimal distance of center of any faces
double findDist() { //compute center of mass
    double totalWeight = 0; Pt center(.0, .0, .0);
    Pt first = info[face[0][0]];
    for (int i = 0; i < SIZE(face); ++i) {
        Pt p = (info[face[i][0]]+info[face[i][1]]+info[face
            [i][2]]+first)*.25;
        double weight = mix(info[face[i][0]] - first, info[
            face[i][1]]
            - first, info[face[i][2]] - first);
        totalWeight += weight; center = center + p * weight
            ;
    } center = center / totalWeight;
    double res = 1e100; //compute distance
    for (int i = 0; i < SIZE(face); ++i)
        res = min(res, calcDist(center, face[i][0], face[i
            ][1], face[i][2]));
    return res; }

```

8.11 CircleCover

```

#define N 1021
struct CircleCover{
    int C; Circ c[ N ];
    bool g[ N ][ N ], overlap[ N ][ N ];
    // Area[i] : area covered by at least i circles
    D Area[ N ];
    void init( int _C ){ C = _C; }
    bool CCinter( Circ& a, Circ& b, Pt& p1, Pt& p2 ){
        Pt o1 = a.O, o2 = b.O;
        D r1 = a.R, r2 = b.R;
        if( norm( o1 - o2 ) > r1 + r2 ) return false;
        if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
            return true;
        D d2 = ( o1 - o2 ) * ( o1 - o2 );
        D d = sqrt(d2);
        if( d > r1 + r2 ) return false;
        Pt u=(o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
        D A=sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
        Pt v=Pt( o1.Y-o2.Y, -o1.X + o2.X ) * A / (2*d2);
        p1 = u + v; p2 = u - v;
        return true;
    }
    struct Teve {
        Pt p; D ang; int add;
        Teve() {}
        Teve(Pt _a, D _b, int _c):p(_a), ang(_b), add(_c){}
        bool operator<(const Teve &a)const
        {return ang < a.ang;}
    }eve[ N * 2 ];
    // strict: x = 0, otherwise x = -1
    bool disjunct( Circ& a, Circ& b, int x )
    {return sign( norm( a.O - b.O ) - a.R - b.R ) > x;}
    bool contain( Circ& a, Circ& b, int x )
    {return sign( a.R - b.R - norm( a.O - b.O ) ) > x;}
    bool contain(int i, int j){
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 ||
            (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
            contain(c[i], c[j], -1);
    }
    void solve(){
        for( int i = 0 ; i <= C + 1 ; i ++ )
            Area[ i ] = 0;
        for( int i = 0 ; i < C ; i ++ )
            for( int j = 0 ; j < C ; j ++ )
                overlap[i][j] = contain(i, j);
        for( int i = 0 ; i < C ; i ++ )

```

```

    for( int j = 0 ; j < C ; j ++ )
        g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                    disjunct(c[i], c[j], -1));
    for( int i = 0 ; i < C ; i ++ ){
        int E = 0, cnt = 1;
        for( int j = 0 ; j < C ; j ++ )
            if( j != i && overlap[j][i] )
                cnt ++;
        for( int j = 0 ; j < C ; j ++ )
            if( i != j && g[i][j] ){
                Pt aa, bb;
                CCinter(c[i], c[j], aa, bb);
                D A=atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                D B=atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
                eve[E ++] = Teve(bb, B, 1);
                eve[E ++] = Teve(aa, A, -1);
                if(B > A) cnt ++;
            }
        if( E == 0 ) Area[ cnt ] += pi * c[i].R * c[i].R;
        else{
            sort( eve , eve + E );
            eve[E] = eve[0];
            for( int j = 0 ; j < E ; j ++ ){
                cnt += eve[j].add;
                Area[cnt] += (eve[j].p ^ eve[j + 1].p) * .5;
                D theta = eve[j + 1].ang - eve[j].ang;
                if (theta < 0) theta += 2. * pi;
                Area[cnt] +=
                    (theta - sin(theta)) * c[i].R*c[i].R * .5;
            }
        }
    }
}
};

```

8.12 DelaunayTriangulation

/ Delaunay Triangulation:*

Given a sets of points on 2D plane, find a triangulation such that no points will strictly inside circumcircle of any triangle.

*find : return a triangle contain given point
add_point : add a point into triangulation*

*A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]*

*calculation involves $O(|V|^6)$ */*

```

const int N = 100000 + 5;
const type inf = 2e3;
type eps = 1e-6; // 0 when integer
type sqr(type x) { return x*x; }
// return p4 is in circumcircle of tri(p1,p2,p3)
bool in_cc(const Pt& p1, const Pt& p2, const Pt& p3,
           const Pt& p4){
    type u11 = p1.X - p4.X; type u12 = p1.Y - p4.Y;
    type u21 = p2.X - p4.X; type u22 = p2.Y - p4.Y;
    type u31 = p3.X - p4.X; type u32 = p3.Y - p4.Y;
    type u13 = sqr(p1.X)-sqr(p4.X)+sqr(p1.Y)-sqr(p4.Y);
    type u23 = sqr(p2.X)-sqr(p4.X)+sqr(p2.Y)-sqr(p4.Y);
    type u33 = sqr(p3.X)-sqr(p4.X)+sqr(p3.Y)-sqr(p4.Y);
    type det = -u13*u22*u31 + u12*u23*u31 + u13*u21*u32
                -u11*u23*u32 - u12*u21*u33 + u11*u22*u33;
    return det > eps;
}
type side(const Pt& a, const Pt& b, const Pt& p)
{ return (b - a) ^ (p - a); }
typedef int SdRef;
struct Tri;
typedef Tri* TriRef;
struct Edge {
    TriRef tri; SdRef side;
    Edge():tri(0), side(0){}
    Edge(TriRef _tri, SdRef _side):tri(_tri), side(_side)
    {}
};
struct Tri {
    Pt p[3];

```

```

    Edge edge[3];
    TriRef chd[3];
    Tri() {}
    Tri(const Pt& p0, const Pt& p1, const Pt& p2) {
        p[0] = p0; p[1] = p1; p[2] = p2;
        chd[0] = chd[1] = chd[2] = 0;
    }
    bool has_chd() const { return chd[0] != 0; }
    int num_chd() const {
        return chd[0] == 0 ? 0
            : chd[1] == 0 ? 1
            : chd[2] == 0 ? 2 : 3;
    }
    bool contains(Pt const& q) const {
        for( int i = 0 ; i < 3 ; i ++ )
            if( side(p[i], p[(i + 1) % 3], q) < -eps )
                return false;
        return true;
    }
} pool[ N * 10 ], *tris;
void edge( Edge a, Edge b ){
    if(a.tri) a.tri->edge[a.side] = b;
    if(b.tri) b.tri->edge[b.side] = a;
}
struct Trig { // Triangulation
    Trig(){
        the_root = // Tri should at least contain all
                    points
                    new(tris++)Tri(Pt(-inf,-inf),Pt(+inf+inf,-inf),Pt
                    (-inf,+inf+inf));
    }
    TriRef find(Pt p)const{ return find(the_root,p); }
    void add_point(const Pt& p){ add_point(find(the_root,
        p),p); }
    TriRef the_root;
    static TriRef find(TriRef root, const Pt& p) {
        while( true ){
            if( !root->has_chd() )
                return root;
            for( int i = 0; i < 3 && root->chd[i] ; ++i )
                if (root->chd[i]->contains(p)) {
                    root = root->chd[i];
                    break;
                }
        }
        assert( false ); // "point not found"
    }
}
void add_point(TriRef root, Pt const& p) {
    TriRef tab,tbc,tca;
    /* split it into three triangles */
    tab=new(tris++) Tri(root->p[0],root->p[1],p);
    tbc=new(tris++) Tri(root->p[1],root->p[2],p);
    tca=new(tris++) Tri(root->p[2],root->p[0],p);
    edge(Edge(tab,0), Edge(tbc,1));
    edge(Edge(tbc,0), Edge(tca,1));
    edge(Edge(tca,0), Edge(tab,1));
    edge(Edge(tab,2), root->edge[2]);
    edge(Edge(tbc,2), root->edge[0]);
    edge(Edge(tca,2), root->edge[1]);
    root->chd[0] = tab;
    root->chd[1] = tbc;
    root->chd[2] = tca;
    flip(tab,2);
    flip(tbc,2);
    flip(tca,2);
}
void flip(TriRef tri, SdRef pi) {
    TriRef trj = tri->edge[pi].tri;
    int pj = tri->edge[pi].side;
    if (!trj) return;
    if (!in_cc(tri->p[0],tri->p[1],tri->p[2],trj->p[pj]
        ])) return;
    /* flip edge between tri,trj */
    TriRef trk = new(tris++) Tri(tri->p[(pi+1)%3], trj
        ->p[pj], tri->p[pi]);
    TriRef trl = new(tris++) Tri(trj->p[(pj+1)%3], tri
        ->p[pi], trj->p[pj]);
    edge(Edge(trk,0), Edge(trl,0));
    edge(Edge(trk,1), tri->edge[(pi+2)%3]);
    edge(Edge(trk,2), trj->edge[(pj+1)%3]);
    edge(Edge(trl,1), trj->edge[(pj+2)%3]);
    edge(Edge(trl,2), tri->edge[(pi+1)%3]);

```

```

    tri->chd[0]=trk; tri->chd[1]=trl; tri->chd[2]=0;
    trj->chd[0]=trk; trj->chd[1]=trl; trj->chd[2]=0;
    flip(trk,1); flip(trk,2);
    flip(trl,1); flip(trl,2);
}
};
vector<TriRef> triang;
set<TriRef> vst;
void go( TriRef now ){
    if( vst.find( now ) != vst.end() )
        return;
    vst.insert( now );
    if( !now->has_chd() ){
        triang.push_back( now );
        return;
    }
    for( int i = 0 ; i < now->num_chd() ; i ++ )
        go( now->chd[ i ] );
}
void build( int n , Pt* ps ){
    tris = pool;
    random_shuffle(ps, ps + n);
    Trig tri;
    for(int i = 0; i < n; ++ i)
        tri.add_point(ps[i]);
    go( tri.the_root );
}

```

8.13 Tangent line of two circles

```

vector<Line> go( const Cir& c1 , const Cir& c2 , int
    sign1 ){
    // sign1 = 1 for outer tang, -1 for inter tang
    vector<Line> ret;
    double d_sq = norm2( c1.0 - c2.0 );
    if( d_sq < eps ) return ret;
    double d = sqrt( d_sq );
    Pt v = ( c2.0 - c1.0 ) / d;
    double c = ( c1.R - sign1 * c2.R ) / d;
    if( c * c > 1 ) return ret;
    double h = sqrt( max( 0.0 , 1.0 - c * c ) );
    for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
        Pt n = { v.X * c - sign2 * h * v.Y ,
                v.Y * c + sign2 * h * v.X };
        Pt p1 = c1.0 + n * c1.R;
        Pt p2 = c2.0 + n * ( c2.R * sign1 );
        if( fabs( p1.X - p2.X ) < eps and
            fabs( p1.Y - p2.Y ) < eps )
            p2 = p1 + perp( c2.0 - c1.0 );
        ret.push_back( { p1 , p2 } );
    }
    return ret;
}

```

8.14 minMaxEnclosingRectangle

```

pdd solve(vector<p11> &dots){
    vector<p11> hull;
    const double INF=1e18,qi=acos(-1)/2*3;
    cv.dots=dots;
    hull=cv.hull();
    double Max=0,Min=INF,deg;
    ll n=hull.size();
    hull.pb(hull[0]);
    for(int i=0,u=1,r=1,i<n;++i){
        p11 nw=hull[i+1]-hull[i];
        while(cross(nw,hull[u+1]-hull[i])>cross(nw,hull[u]-
            hull[i]))
            u=(u+1)%n;
        while(dot(nw,hull[r+1]-hull[i])>dot(nw,hull[r]-hull
            [i]))
            r=(r+1)%n;
        if(!i) l=(r+1)%n;
        while(dot(nw,hull[l+1]-hull[i])<dot(nw,hull[l]-hull
            [i]))
            l=(l+1)%n;
    }
}

```

```

    Min=min(Min,(double)(dot(nw,hull[r]-hull[i])-dot(nw
        ,hull[l]-hull[i]))*cross(nw,hull[u]-hull[i])/
        abs2(nw));
    deg=acos((double)dot(hull[r]-hull[l],hull[u]-hull[i
        ])/abs(hull[r]-hull[l])/abs(hull[u]-hull[i]));
    deg=(qi-deg)/2;
    Max=max(Max,(double)abs(hull[r]-hull[l])*abs(hull[u
        ]-hull[i])*sin(deg)*sin(deg));
}
return pdd(Min,Max);
}

```

8.15 Minkowski Sum

```

/* convex hull Minkowski Sum*/
#define INF 10000000000000LL
int pos( const Pt& tp ){
    if( tp.Y == 0 ) return tp.X > 0 ? 0 : 1;
    return tp.Y > 0 ? 0 : 1;
}
#define N 300030
Pt pt[ N ], qt[ N ], rt[ N ];
LL Lx,Rx;
int dn,un;
inline bool cmp( Pt a, Pt b ){
    int pa=pos( a ),pb=pos( b );
    if(pa==pb) return (a^b)>0;
    return pa<pb;
}
int minkowskiSum(int n,int m){
    int i,j,r,p,q,fi,fj;
    for(i=1,p=0;i<n;i++){
        if( pt[i].Y<pt[p].Y ||
            (pt[i].Y==pt[p].Y && pt[i].X<pt[p].X) ) p=i;
    }
    for(i=1,q=0;i<m;i++){
        if( qt[i].Y<qt[q].Y ||
            (qt[i].Y==qt[q].Y && qt[i].X<qt[q].X) ) q=i;
    }
    rt[0]=pt[p]+qt[q];
    r=1; i=p; j=q; fi=fj=0;
    while(1){
        if((fj&&j==q) ||
            ( (!fi||i==p) &&
                cmp(pt[(p+1)%n]-pt[p],qt[(q+1)%m]-qt[q]) ) ){
            rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
            p=(p+1)%n;
            fi=1;
        }else{
            rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
            q=(q+1)%m;
            fj=1;
        }
        if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
            r++;
        else rt[r-1]=rt[r];
        if(i==p && j==q) break;
    }
    return r-1;
}
void initInConvex(int n){
    int i,p,q;
    LL Ly,Ry;
    Lx=INF; Rx=-INF;
    for(i=0;i<n;i++){
        if(pt[i].X<Lx) Lx=pt[i].X;
        if(pt[i].X>Rx) Rx=pt[i].X;
    }
    Ly=Ry=INF;
    for(i=0;i<n;i++){
        if(i==0&&pt[i].Y<Ly){ Ly=pt[i].Y; p=i; }
        if(pt[i].X==Rx && pt[i].Y<Ry){ Ry=pt[i].Y; q=i; }
    }
    for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
    qt[dn]=pt[q]; Ly=Ry=-INF;
    for(i=0;i<n;i++){
        if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i; }
        if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
    }
    for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
    rt[un]=pt[q];
}

```

```

inline int inConvex(Pt p){
    int L,R,M;
    if(p.X<Lx || p.X>Rx) return 0;
    L=0;R=dn;
    while(L<R-1){ M=(L+R)/2;
        if(p.X<qt[M].X) R=M; else L=M; }
    if(tri(qt[L],qt[R],p)<0) return 0;
    L=0;R=un;
    while(L<R-1){ M=(L+R)/2;
        if(p.X<rt[M].X) R=M; else L=M; }
    if(tri(rt[L],rt[R],p)>0) return 0;
    return 1;
}
int main(){
    int n,m,i;
    Pt p;
    scanf("%d",&n);
    for(i=0;i<n;i++) scanf("%Lld%Lld",&pt[i].X,&pt[i].Y);
    scanf("%d",&m);
    for(i=0;i<m;i++) scanf("%Lld%Lld",&qt[i].X,&qt[i].Y);
    n=minkowskiSum(n,m);
    for(i=0;i<n;i++) pt[i]=rt[i];
    scanf("%d",&m);
    for(i=0;i<m;i++) scanf("%Lld%Lld",&qt[i].X,&qt[i].Y);
    n=minkowskiSum(n,m);
    for(i=0;i<n;i++) pt[i]=rt[i];
    initInConvex(n);
    scanf("%d",&m);
    for(i=0;i<m;i++){
        scanf("%Lld %Lld",&p.X,&p.Y);
        p.X*=3; p.Y*=3;
        puts(inConvex(p)? "YES": "NO");
    }
}

```

9 Else

9.1 Mo's Alogrithm(With modification)

```

struct QUERY{//BLOCK=N^{2/3}
    int L,R,id,LBId,RBId,T;
    QUERY(int l,int r,int id,int lb,int rb,int t):
        L(l),R(r),id(id),LBId(lb),RBId(rb),T(t){}
    bool operator<(const QUERY &b)const{
        if(LBId!=b.LBId) return LBId<b.LBId;
        if(RBId!=b.RBId) return RBId<b.RBId;
        return T<b.T;
    }
};
vector<QUERY> query;
int cur_ans,arr[MAXN],ans[MAXN];
void addTime(int L,int R,int T){}
void subTime(int L,int R,int T){}
void add(int x){}
void sub(int x){}
void solve(){
    sort(ALL(query));
    int L=0,R=0,T=-1;
    for(auto q:query){
        while(T<q.T) addTime(L,R,++T);
        while(T>q.T) subTime(L,R,T--);
        while(R<q.R) add(arr[++R]);
        while(L>q.L) add(arr[--L]);
        while(R>q.R) sub(arr[R--]);
        while(L<q.L) sub(arr[L++]);
        ans[q.id]=cur_ans;
    }
}

```

9.2 Mo's Alogrithm On Tree

```

const int MAXN=40005;
vector<int> G[MAXN]; //1-base
int n,B,arr[MAXN],ans[100005],cur_ans;
int in[MAXN],out[MAXN],dfn[MAXN*2],dft;
int deep[MAXN],sp[(__lg(MAXN*2)+1)[MAXN*2]],bln[MAXN],spt
;

```

```

bitset<MAXN> inset;
struct QUERY{
    int L,R,Lid,id,lca;
    QUERY(int l,int r,int _id):L(l),R(r),lca(0),id(_id){}
    bool operator<(const QUERY &b){
        if(Lid!=b.Lid) return Lid<b.Lid;
        return R<b.R;
    }
};
vector<QUERY> query;
void dfs(int u,int f,int d){
    deep[u]=d,sp[0][spt]=u,bln[u]=spt++;
    dfn[dft]=u,in[u]=dft++;
    for(int v:G[u])
        if(v!=f)
            dfs(v,u,d+1),sp[0][spt]=u,bln[u]=spt++;
    dfn[dft]=u,out[u]=dft++;
}
int lca(int u,int v){
    if(bln[u]>bln[v]) swap(u,v);
    int t=__lg(bln[v]-bln[u]+1);
    int a=sp[t][bln[u]],b=sp[t][bln[v]-(1<<t)+1];
    if(deep[a]<deep[b]) return a;
    return b;
}
void sub(int x){}
void add(int x){}
void flip(int x){
    if(inset[x]) sub(arr[x]);
    else add(arr[x]);
    inset[x]=~inset[x];
}
void solve(){
    B=sqrt(2*n),dft=spt=cur_ans=0,dfs(1,1,0);
    for(int i=1,x=2;x<2*n;++i,x<=1)
        for(int j=0;j+x<=2*n;++j)
            if(deep[sp[i-1][j]]<deep[sp[i-1][j+x/2]])
                sp[i][j]=sp[i-1][j];
            else sp[i][j]=sp[i-1][j+x/2];
    for(auto &q:query){
        int c=lca(q.L,q.R);
        if(c==q.L || c==q.R)
            q.L=out[c==q.L?q.R:q.L],q.R=out[c];
        else if(out[q.L]<in[q.R])
            q.lca=c,q.L=out[q.L],q.R=in[q.R];
        else q.lca=c,c=in[q.L],q.L=out[q.R],q.R=c;
        q.Lid=q.L/B;
    }
    sort(ALL(query));
    int L=0,R=-1;
    for(auto q:query){
        while(R<q.R) flip(dfn[++R]);
        while(L>q.L) flip(dfn[--L]);
        while(R>q.R) flip(dfn[R--]);
        while(L<q.L) flip(dfn[L++]);
        if(q.lca) add(arr[q.lca]);
        ans[q.id]=cur_ans;
        if(q.lca) sub(arr[q.lca]);
    }
}

```

9.3 DynamicConvexTrick

```

// only works for integer coordinates!!
bool Flag; // 0: insert Line, 1: Lower_bound x
template<class val = ll, class compare = less<val>> //
    sort Lines with comp
struct DynamicConvexTrick{
    static const ll minx = 0, maxx = 1l(1e9) + 5;
    static compare comp;
    struct Line{
        val a, b, l, r; // Line ax + b in [l, r]
        Line(val _a, val _b, val _l = minx, val _r =
            maxx):a(_a), b(_b), l(_l), r(_r){}
        val operator () (val x) const {
            return a * x + b;
        }
    };
    struct cmp{

```

```

    bool operator () (const Line a, const Line b){
        if(Flag == 0)return comp(a.a, b.a);
        return a.r < b.l;
    }
};
inline val idiv(val a, val b){
    return a / b - (a % b && a < 0 ^ b < 0);
}
set<Line, cmp> st;
void ins(val a, val b){
    Flag = 0;
    Line L(a, b);
    auto it = st.lower_bound(L);
    if(it != st.begin() && it != st.end())
        if(!comp((*prev(it))(it->l - 1), L(it->l - 1)) && !comp((*it)(it->l), L(it->l)))
            return;
    while(it != st.end()){
        if(it->a == L.a && !comp(it->b, L.b))return
        ;
        if(comp((*it)(it->r), L(it->r)))it = st.
            erase(it);
        else{
            Line M = *it;
            st.erase(it);
            L.r = max(idiv(L.b - M.b, M.a - L.a),
                minx);
            M.l = L.r + 1;
            it = st.insert(M).X;
            break;
        }
    }
    while(it != st.begin()){
        auto pit = prev(it);
        if(comp((*pit)(pit->l), L(pit->l)))st.erase
            (pit);
        else{
            Line M = *pit;
            st.erase(pit);
            M.r = min(idiv(L.b - M.b, M.a - L.a),
                maxx - 1);
            L.l = M.r + 1;
            st.insert(M);
            break;
        }
    }
    st.insert(L);
}
val operator () (val x){
    Flag = 1;
    auto it = st.lower_bound({0, 0, x, x});
    return (*it)(x);
}
};
DynamicConvexTrick<> DCT;

```

```

    if(b.charAt(0) == '*')
        System.out.printf("%s\n", ia.multiply(ic));
    if(b.charAt(0) == '/')
        System.out.printf("%s\n", ia.divide(ic));
    }
}

```

10 JAVA

10.1 Big number

```

import java.util.Scanner;
import java.math.BigInteger;

public class JAVA{
    public static void main(String[] args){
        Scanner cin = new Scanner(System.in);
        String a, b, c;
        while(cin.hasNext()){
            a = cin.next();
            b = cin.next();
            c = cin.next();
            BigInteger ia = new BigInteger(a);
            BigInteger ic = new BigInteger(c);
            if(b.charAt(0) == '+')
                System.out.printf("%s\n", ia.add(ic));
            if(b.charAt(0) == '-')
                System.out.printf("%s\n", ia.subtract(ic));
        }
    }
}

```