Notebook UNTreeCiclo

Contents

1 C++

1.1 C++ plantilla

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(v) v.begin(), v.end()
#define sz(arr) ((int) arr.size())
#define rep(i, a, b) for(int i = a; i < (b); ++i)
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef long long ll;
const char ln = '\n';

#define watch(x) cout<<#x<<"="<<x<<'\n'
#define print(arr) for(auto& x:arr)cout<<x<<" ";cout<<"\n"

typedef long double ld;
typedef vector<ii> vii;
typedef vector<long long> vl;
typedef pair<ll, ll> pll;
typedef vector<pll> vll;
const int INF = 1e9;
const ll INFL = 1e18;
const int MOD = 1e9+7;
const double EPS  = 1e-9;
const ld PI = acosl(-1);
int dirx[4] = {0,-1,1,0};
int diry[4] = {-1,0,0,1};
int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};
int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};
const string ABC = "abcdefghijklmnopqrstuvwxyz";

void main2(){

}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(0);
        cout << setprecision(20) << fixed;
    // freopen("file.in", "r", stdin);
    // freopen("file.out", "w", stdout);
        clock_t start = clock();
        main2();
        cerr<<double(clock()-start)/CLOCKS_PER_SEC<<" s\n";
        return 0;
}
```

1.2 Librerias

```cpp
// En caso de que no sirva #include <bits/stdc++.h>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <sstream>
#include <fstream>
#include <cassert>
#include <climits>
#include <cstdlib>
#include <cstring>
#include <string>
```

```cpp
#include <cstdio>
#include <vector>
#include <cmath>
#include <queue>
#include <deque>
#include <stack>
#include <list>
#include <map>
#include <set>
#include <bitset>
#include <iomanip>
#include <unordered_map>
////
#include <tuple>
#include <random>
#include <chrono>
```

1.3 Create

```python
import os

def folder(problem):
        os.makedirs(problem, exist_ok=True)
        with open(os.path.join(problem, "main.cpp"), "w")
            as f:
                f.write("")
        with open(os.path.join(problem, "in.txt"), "w")
           as f:
                f.write("")

with open("plantilla.cpp", "w") as f:
        f.write("")
with open("out.txt", "w") as f:
        f.write("")

for i in range(ord('A'), ord('P') + 1):
        folder(chr(i))
```

1.4 Bitmask

```
* Operaciones a nivel de bits. Si n es ll usar 1ll<< en
   los corrimientos.

x & 1           -> Verifica si x es impar
x & (1<<i)      -> Verifica si el i-esimo bit esta
   encendido
x = x | (1<<i)  -> Enciende el i-esimo bit
x = x & ~(1<<i) -> Apaga el i-esimo bit
x = x ^ (1<<i)  -> Invierte el i-esimo bit
x = ~x          -> Invierte todos los bits
x & -x          -> Devuelve el bit encendido mas a la
   derecha (potencia de 2, no el indice)
~x & (x+1)      -> Devuelve el bit apagado mas a la
   derecha (potencia de 2, no el indice)
x = x | (x+1)   -> Enciende el bit apagado mas a la
   derecha
x = x & (x-1)   -> Apaga el bit encendido mas a la
   derecha
x = x & ~y      -> Apaga en x los bits encendidos de y

* Funciones del compilador gcc. Si n es ll agregar el
   sufijo ll, por ej: __builtin_clzll(n).

__builtin_clz(x)      -> Cantidad de bits apagados por la
    izquierda
__builtin_ctz(x)      -> Cantidad de bits apagados por la
    derecha. Indice del bit encendido mas a la derecha
__builtin_popcount(x) -> Cantida de bits encendidos
__builtin_ffs(x)          -> Posicion del primer bit
   prendido (lsb+1)

* Logaritmo en base 2 (entero). Indice del bit encendido
   mas a la izquierda. Si x es ll usar 63 y clzll(x).
// O(1)
int lg2(const int &x) { return 31-__builtin_clz(x); }

* Itera, con indices, los bits encendidos de una mascara.
// O(#bits_encendidos)
for (int x = mask; x; x &= x-1) {
        int i = __builtin_ctz(x);
}

* Itera todas las submascaras de una mascara. (Iterar
   todas las submascaras de todas las mascaras es O(3^n))
   .
// O(2^(#bits_encendidos))
for (int sub = mask; ; sub = (sub-1)&mask) {
        // ...
        if (sub == 0) break;
}

// Ascendente
for(int sub = 0; ; sub = (sub-mask)&mask) {
        // ...
        if (sub == mask) break;
}

* retorna la siguiente mask con la misma cantidad
   encendida
ll nextMask(ll x){
        ll c = x & -x;
        ll r = x + c;
        return (((r ^ x) >> 2) / c) | r;
}

// optimiza el .count de los bitsets y el popcount
#pragma GCC target("popcnt")

// Formulas
a | b = a ^ b + a & b
```

```
a ^ (a & b) = (a | b) ^ b
b ^ (a & b) = (a | b) ^ a
(a & b) ^ (a | b) = a ^ b
a + b = a | b + a & b
a + b = a ^ b + 2 * (a & b)
a - b = (a ^ (a & b)) - ((a | b) ^ a)
a - b = ((a | b) ^ b) - ((a | b) ^ a)
a - b = (a ^ (a & b)) - (b ^ (a & b))
a - b = ((a | b) ^ b) - (b ^ (a & b))
a ^ b = ~(a & b) & (a | b)
si (x < y < z) entonces min(x^y, y^z) < (x^z)
```

1.5 Custom Hashing

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

        size_t operator()(long long x) const {
                static const long long FIXED_RANDOM =
                    chrono::steady_clock::now().
                    time_since_epoch().count();
                return splitmix64(x + FIXED_RANDOM);
        }

        size_t operator()(const pair<int,int>& x) const {
                return (size_t) x.first * 37U + (size_t)
                    x.second;
        }

        size_t operator()(const vector<int>& v) const {
                size_t s = 0;
                for(auto &e : v)
                        s^=hash<int>()(e)+0x9e3779b9+(s
                            <<6)+(s>>2);
                return s;
        }
};

unordered_map<long long, int, custom_hash> safe_map; //
    unordered_map or gp_hash_table

safe_map.max_load_factor(0.25);
safe_map.reserve(1024); // potencia de 2 mas cercana

multitest - no usar reserve (por el clear, es pesado)
```

1.6 Random

```
typedef unsigned long long u64;
mt19937_64 rng (chrono::steady_clock::now().
    time_since_epoch().count());
u64 xor_hash=rng();

// return random numbers in the range [l,r]
mt19937 rng (chrono::steady_clock::now().time_since_epoch
    ().count());
double rand(double l, double r){return
    uniform_real_distribution<double>(l, r)(rng);}
int rand(int l, int r){return uniform_int_distribution<
    int>(l, r)(rng);}

shuffle(all(vector), rng);
```

1.7 Cosas de strings

```
// si el caracter que separa el texto es distinto al
    espacio
// entonces descomentar el segundo parametro y cambiar el
    while por el otro
vector<string> split(const string &s/*, char c */){
        vector<string> v;
        stringstream ss(s);
        string sub;
        while(ss>>sub)v.push_back(sub);
        // while(getline(ss,sub,c))if(sz(sub))v.push_back
            (sub);
        return v;
}

string s;
for(char& c:s)c=toupper(c);
for(char& c:s)c=tolower(c);
int n=stoi(s); // string -> int
int n=stoi(s, nullptr, 2); // bin string -> int
double d=stod(s); // string -> double
string s=to_string(n); // int -> string
cout<<"\U0001F600"; // emojis

Quitar repetidos (lo pongo aca porque no se donde mas
    ponerlo)
sort(all(bs));
bs.resize(unique(all(bs)) - bs.begin());
```

2 Arboles

2.1 Centroid Decomposition

```
// O(n*log(n))
// 1) init(adj,n);
struct CentroidDecomposition{
```

```
	vector<vi> adj;
	vi dad,sz,proc;

	int operator[](int i){return dad[i];}
	void init(vector<vi>& adj2, int n){
		proc.assign(n,false);
		dad.resize(n);
		sz.resize(n);
		adj=adj2;
		build();
	}

	void build(int v=0, int p=-1){
		int n=dfsSz(v, p);
		int centroid=dfsCentroid(v, p, n);
		dad[centroid]=p;
		// anadir dfs para el conteo de caminos
		proc[centroid]=true;
		for(int u:adj[centroid]){
			if(u==p || proc[u])continue;
			build(u,centroid);
		}
	}

	int dfsSz(int v,int p){
		sz[v]=1;
		for(int u:adj[v]){
			if(u==p || proc[u])continue;
			sz[v]+=dfsSz(u, v);
		}
		return sz[v];
	}

	int dfsCentroid(int v, int p, int n){
		for(int u:adj[v]){
			if(u==p || proc[u])continue;
			if(sz[u]>n/2)return dfsCentroid(u
			    ,v,n);
		}
		return v;
	}
};

// para el arbol de centroides
// for(int b=a;b!=-1;b=cd[b])
```

2.2 Hash Tree

```
const int MOD=1e9+97;
const int P[2]={998244353,1000000007};
const int Q[2]={1000000033,1000000021};
const int R[2]={123456789,987654321};

int add(int a, int b){return a+b>=MOD?a+b-MOD:a+b;}
int mul(int a, int b){return ll(a)*b%MOD;}
int binpow(int a, int b){
	int res=1;a%=MOD;
	while(b>0){
		if(b&1)res=mul(res,a);
		a=mul(a,a);
		b>>=1;
	}
	return res%MOD;
}

// O(n), 1-indexed
struct Tree{
	vector<vi> g;
	int n;

	Tree(int _n):n(_n){g.resize(n+1);}
	void add_edge(int u, int v){
		g[u].push_back(v);
		g[v].push_back(u);
	}

	ii hash(int u, int pre=0){
		vector<vi> nw(2,vi());
		for(int v:g[u])
			if(v!=pre){
				ii tmp=hash(v,u);
				nw[0].push_back(tmp.first
				    );
				nw[1].push_back(tmp.
				    second);
			}
		ii ans={0,0};
		for(int i=0;i<2;++i){
			int& tmp=(i?ans.second:ans.first)
			    ;
			for(int x:nw[i])tmp=add(tmp,
			    binpow(P[i], x));
			tmp=add(mul(tmp,Q[i]),R[i]);
		}
		return ans;
	}

	// Isomorphism
	bool iso(Tree& t){
		vi a=get_centers();
		vi b=t.get_centers();
		for(int x:a)for(int y:b)if(hash(x)==t.
		    hash(y))return 1;
		return 0;
	}

	vi get_centers(){
		auto du=bfs(1);
		int v=max_element(all(du))-du.begin();
		auto dv=bfs(v);
		int u=max_element(all(dv))-dv.begin();
		du=bfs(u);
```

```
            vi ans;
            for(int i=1;i<=n;++i){
                if(du[i]+dv[i]==du[v] && du[i]>=
                    du[v]/2 && dv[i]>=du[v]/2){
                        ans.push_back(i);
                }
            }
            return ans;
        }

        vi bfs(int s){
            queue<int> q;
            vi d(n+1, n*2);
            d[0]=-1;
            q.push(s);
            d[s]=0;
            while(!q.empty()){
                int u=q.front();
                q.pop();
                for(int v:g[u])
                    if(d[u]+1<d[v]){
                        d[v]=d[u]+1;
                        q.push(v);
                    }
            }
            return d;
        }
};
```

2.3 Heavy Light Decomposition

```
typedef long long T;
T oper(T a, T b){return max(a,b);}
T null=-1e18;
struct SegTree{}; // Add Segment tree

// O(nlog(n)) build
// O(log(n)^2) (query - update) path
// O(log(n)) (query - update) subtree, node

// 1) call build(adj,n,root)
struct HLD{
        SegTree st;
        vector<vi> adj;
        vi dad,root,dep,sz,pos;
        int time;
        bool edges=false; // if the values are on edges
            instead of nodes

        void build(vector<vi>& adj2, int n, int v=0){ //
            v is the root
                adj=adj2;
                dad.resize(n);
                root.resize(n);
                dep.resize(n);
                sz.resize(n);
                pos.resize(n);

                root[v]=dad[v]=v;
                dep[v]=time=0;

                dfsSz(v);
                dfsHld(v);
                // vector<T> palst(n);
                // for(int i=0;i<n;++i)palst[pos[i]]=vals
                    [i];
                // st.build(palst);
                st.build(n);
        }

        void dfsSz(int x){
                sz[x]=0;
                for(int& y:adj[x]){
                        if(y==dad[x])continue;
                        dad[y]=x;dep[y]=dep[x]+1;
                        dfsSz(y);
                        sz[x]+=sz[y]+1;
                        if(sz[y]>sz[adj[x][0]])swap(y,adj
                            [x][0]);
                }
        }

        void dfsHld(int x){
                pos[x]=time++;
                for(int y:adj[x]){
                        if(y==dad[x])continue;
                        root[y]=(y==adj[x][0]?root[x]:y);
                        dfsHld(y);
                }
        }

        // O(log(n)^2)
        template <class Oper>
        void processPath(int x, int y, Oper op){
                for(;root[x]!=root[y];y=dad[root[y]]){
                        if(dep[root[x]]>dep[root[y]])swap
                            (x,y);
                        op(pos[root[y]],pos[y]);
                }
                if(dep[x]>dep[y])swap(x,y);
                op(pos[x]+edges,pos[y]);
        }

        void modifyPath(int x, int y, int v){
                processPath(x,y,[this,&v](int l, int r){
                        st.upd(l,r,v);
                });
        }

        T queryPath(int x, int y){
                T res=null;
                processPath(x,y,[this,&res](int l, int r)
                    {
```

```
                res=oper(res, st.get(l,r));
            });
            return res;
        }
        // O(log(n))
        void modifySubtree(int x, int v){
            st.upd(pos[x]+edges,pos[x]+sz[x],v);
        }
        T querySubtree(int x){
            return st.get(pos[x]+edges,pos[x]+sz[x]);
        }
        void modify(int x, int v){st.set(pos[x],v);}
        void modifyEdge(int x, int y, int v){
            if(dep[x]<dep[y])swap(x,y);
            modify(x,v);
        }
};
```

2.4 Kruskal Reconstruction Tree

```
// Kruskal Reconstruction Tree (KRT)
// the main idea is to build a tree to efficiently answer
    queries
// about the minimum or maximum edge weight between two
    nodes.
// each edge will be represented as a node in the tree.
// query(a,b) = lca(a,b)

// Add LCA

const int maxn = 1e5+5;
const int maxm = 2e5+5;
vector<vi> adj;

// sometimes it is useful
int ver[2*(maxn+maxm)]; // node at position i in euler
    tour
int st[maxn+maxm]; // start time of v
int ft[maxn+maxm]; // finish time of v

struct DSU{
        vi p,size;
        vector<bool> roots; // if the graph is a forest
        DSU(int n){
                p.assign(n,0);
                size.assign(n,1);
                roots.assign(n,true);
                for(int i=0;i<n;++i)p[i]=i;
        }
        int get(int a){return (a==p[a]?a:p[a]=get(p[a]))
            ;}
        // unite node a and node b with the edge m =>
            node m
        void unite(int a, int b, int m){
                a=get(a);b=get(b);
                if(a==b)return;
                size[m]=size[a]+size[b];
                p[a]=p[b]=m;
                roots[a]=false;
                roots[b]=false;
                adj[m].push_back(a);
                adj[m].push_back(b);
        }
};
```

2.5 LCA Binary Lifting

```
// O(n*log(n)) build
// O(log(n)) kth, lca, dist
struct LCA{
        vector<vi> up;
        vi dep;
        int n,maxlog;

        void build(vector<vi>& adj, int root){
                n=sz(adj);
                maxlog=ceil(log2(n))+3;
                up.assign(n, vi(maxlog, -1));
                dep.assign(n,0);
                dfs(adj,root);
                calc(n);
        }

        void dfs(vector<vi>& adj, int v=0, int p=-1){
                up[v][0]=p;
                for(int u:adj[v]){
                        if(u==p)continue;
                        dep[u]=dep[v]+1;
                        dfs(adj, u, v);
                }
        }

        void calc(int n){
                for(int l=1;l<maxlog;++l){
                        for(int i=0;i<n;++i){
                                if(up[i][l-1]!=-1){
                                        up[i][l]=up[up[i
                                            ][l-1]][l-1];
                                }
                        }
                }
        }

        // kth ancestor, return -1 if it doesnt exits
        int kth(int u, int k){
                for(int l=maxlog-1;l>=0;--l){
                        if(u!=-1 && k&(1<<l)){
                                u=up[u][l];
```

```
                }
            }
            return u;
        }

        int lca(int a, int b){
            // if(kth(a, dep[a])!=kth(b, dep[b]))
              return -1; // forest
            a=kth(a, dep[a]-min(dep[a], dep[b]));
            b=kth(b, dep[b]-min(dep[a], dep[b]));
            if(a==b)return a;
            for(int l=maxlog-1;l>=0;--l){
                if(up[a][l]!=up[b][l]){
                    a=up[a][l];
                    b=up[b][l];
                }
            }
            return up[a][0];
        }

    int dist(int a, int b){
        return dep[a]+dep[b]-2*dep[lca(a,b)];
    }
};
```

2.6 LCA RMQ

```
// Add RMQ - Min
typedef int T;
struct Table{
        void build(vector<T>& a);
        int get(int l, int r);
};

// O(n*log(n)) build
// O(1) lca
struct LCA{
        Table rmq;
        vi time, path, tmp;
        int n,ti;

        void build(vector<vi>& adj, int root){
                path.clear();tmp.clear();
                n=sz(adj);ti=0;
                time.resize(n);
                dfs(adj, root);
                rmq.build(tmp);
        }

        void dfs(vector<vi>& adj, int u, int p=-1){
                time[u]=ti++;
                for(int v:adj[u]){
                        if(v==p)continue;
                        path.push_back(u);
                        tmp.push_back(time[u]);
                        dfs(adj, v, u);
                }
        }

        int lca(int a, int b){ // check forest
                if(a==b)return a;
                a=time[a],b=time[b];
                if(a>b)swap(a,b);
                return path[rmq.get(a,b-1)];
        }
};
```

2.7 Sack

```
const int maxn = 1e5+5;
vi adj[maxn];
int ver[2*maxn]; // nodo en la posicion i del euler tour
int len[maxn];  // tamano del subarbol de v
int st[maxn]; // tiempo inicial de v
int ft[maxn]; // tiempo final de v
int pos=0;

// O(n*log(n))
// 1) dfs0(root);
// 2) dfs1(root);

void dfs0(int v=0, int p=-1){
        len[v]=1;
        ver[pos]=v;
        st[v]=pos++;
        for(int u:adj[v]){
                if(u==p)continue;
                dfs0(u,v);
                len[v]+=len[u];
        }
        ver[pos]=v;
        ft[v]=pos++;
}

bool vis[maxn];
void ask(int v, bool add){
        if(vis[v] && !add){
                vis[v]=false;
                // eliminar nodo v
                // ...
        }else if(!vis[v] && add){
                vis[v]=true;
                // anadir nodo v
                // ...
        }
}

void dfs1(int v=0, int p=-1, bool keep=true){
        int mx=0,id=-1;
        for(int u:adj[v]){
```

```
            if(u==p)continue;
            if(len[u]>mx){
                mx=len[u];
                id=u;
            }
        }
        for(int u:adj[v]){
            if(u!=p && u!=id)
                dfs1(u,v,0);
        }
        if(id!=-1)dfs1(id,v,1);
        for(int u:adj[v]){
            if(u==p || u==id)continue;
            for(int p=st[u];p<ft[u];++p)
                ask(ver[p], 1);
        }
        ask(v, 1);
        // responder las consultas relacionadas con el
           subarbol de v
        // ...
        if(keep)return;
        for(int p=st[v];p<ft[v];++p)
            ask(ver[p], 0);
}
```

2.8 Virtual Tree

```
// O(k*log(k))
// 1) build(n,root,adj);
// 2) query(nodes);

LCA g; // Add LCA
int lca(int a, int b){return g.lca(a,b);};
struct VirtualTree{
        vector<vi> adj,adjVT;
        vector<int> st,ft;
        vector<bool> important;
        int pos=0;

        void build(vector<vi>& adj2, int n, int root){
                important.assign(n,false);
                adjVT.assign(n,vi());
                st.resize(n);
                ft.resize(n);
                adj=adj2;pos=0;
                dfs(root);
        }

        void dfs(int v, int p=-1){
                st[v]=pos++;
                for(int u:adj[v]){
                        if(u==p)continue;
                        dfs(u, v);
                }
                ft[v]=pos++;
        }

        bool upper(int v, int u){return st[v]<=st[u] &&
            ft[v]>=ft[u];}

        int getRootVirtualTree(vi nodes){
                sort(all(nodes), [&](int v, int u){
                    return st[v] < st[u]; });
                int m=sz(nodes);
                for(int i=0;i<m-1;++i){
                        int v=lca(nodes[i], nodes[i+1]);
                        nodes.push_back(v);
                }

                sort(all(nodes), [&](int v, int u){
                    return st[v] < st[u]; });
                nodes.erase(unique(all(nodes)), nodes.end
                    ());
                for(int u:nodes)adjVT[u].clear();

                vi s;
                s.push_back(nodes[0]);
                m=sz(nodes);
                for(int i=1;i<m;++i){
                        int v=nodes[i];
                        while(sz(s)>=2 && !upper(s.back()
                            , v)){
                                adjVT[s[sz(s)-2]].
                                    push_back(s.back());
                                s.pop_back();
                        }
                        s.push_back(v);
                }
                while(sz(s)>=2){
                        adjVT[s[sz(s)-2]].push_back(s.
                            back());
                        s.pop_back();
                }
                return s[0];
        }

        void dfs2(int u, int p=-1){
                if(important[u]){
                        // pass
                }else{
                        // pass
                }
                for(int v:adjVT[u]){
                        if(v==p)continue;
                        dfs2(v,u);
                }
        }

        void query(vi& nodes){
                for(int u:nodes)important[u]=true;
                int root=getRootVirtualTree(nodes);
```

```
            dfs2(root);
            // cout ans
            for(int u:nodes)important[u]=false;
        }
};
```

3 Estructuras de Datos

3.1 Bit

```
// O(n) build
// O(log(n)) get, upd
typedef long long T;
struct BIT{
    vector<T> t;
    int n;

    BIT(int _n){
        n=_n;
        t.assign(n+1,0);
    }
    void upd(int i, T v){ // add v to ith element
        for(int j=i+1;j<=n;j+=j&-j)t[j]+=v;
    }
    T get(int i){ // get sum of range [0,i0)
        T ans=0;
        for(int j=i;j;j-=j&-j)ans+=t[j];
        return ans;
    }
    T get(int l, int r){ // get sum of range [l,r]
        return get(r+1)-get(l);
    }
};
```

3.2 Bit 2D

```
// O(n*m) build
// O(log(n)*log(m)) get, upd
typedef long long T;
struct BIT2D{
    vector<vector<T>> bit;
    int n,m;

    BIT2D(int _n, int _m){
        n=_n;m=_m;
        bit.assign(n+1, vector<T>(m+1,0));
    }
    T get(int x, int y){
        if(x<0 || y<0)return 0;
        T v=0;
        for(int i=x+1;i;i-=i&-i)
            for(int j=y+1;j;j-=j&-j)v+=bit[i
                ][j];
        return v;
    }
    T get(int x, int y, int x2, int y2){
        return get(x2,y2)-get(x-1,y2)-get(x2,y-1)
            +get(x-1,y-1);
    }
    void upd(int x, int y, T dt){
        if(x<0 || y<0)return;
        for(int i=x+1;i<=n;i+=i&-i)
            for(int j=y+1;j<=m;j+=j&-j)bit[i
                ][j]+=dt;
    }
};
```

3.3 Cartesian Tree

```
// O(n) build
typedef long long T;
struct CartesianTree{ // 1-indexed
    vector<int> l,r;
    int root,n;

    CartesianTree(vector<T>& a){
        reverse(all(a));
        a.push_back(0);
        reverse(all(a));
        int tot=0;n=sz(a)-1;
        l.assign(n+1,0);
        r.assign(n+1,0);
        vector<int> s(n+1,0);
        vector<bool> vis(n+1,false);
        for(int i=1;i<=n;++i){
            int k=tot;
            while(k>0 && a[s[k-1]]>a[i])k--;
                // < max heap
            if(k)r[s[k-1]]=i;
            if(k<tot)l[i]=s[k];
            s[k++]=i;
            tot=k;
        }
        for(int i=1;i<=n;++i)vis[l[i]]=vis[r[i
            ]]=1;
        root=0;
        for(int i=1;i<=n;++i){
            if(!vis[i])root=i;
        }
    }
};
```

3.4 Disjoint Set Union

```cpp
struct dsu{
	vi p,size;
	int sets,maxSize;

	dsu(int n){
		p.assign(n,0);
		size.assign(n,1);
		sets = n;
		for (int i = 0; i<n; i++) p[i] = i;
	}

	int find_set(int i) {return (p[i] == i) ? i : (p[i] = find_set(p[i]));}

	bool is_same_set(int i, int j) {return find_set(i) == find_set(j);}

	void unionSet(int i, int j){
		if (!is_same_set(i, j)){
			int a = find_set(i), b = find_set(j);
			if (size[a] < size[b]) swap(a, b);
			p[b] = a;
			size[a] += size[b];
			maxSize = max(size[a], maxSize);
			sets--;
		}
	}
};
```

3.5 Disjoint Sparse Table

```cpp
// lo mismo que sparse table, pero para st opers
// O(n*log(n)) build
// O(1) get
typedef int T;
T null = 0;
T op(T a, T b){return a^b;}
struct DST {
	vector<vector<T>> pre, suf;
	int k, n;
	DST(vector<T>& a) {
		n = sz(a);
		k = log2(n) + 2;
		pre.assign(k + 1, vector<T>(n));
		suf.assign(k + 1, vector<T>(n));
		for(int j = 0; (1 << j) <= n; ++j) {
			int mask = (1 << j) - 1;
			T nw = null;
			for(int i = 0; i < n; ++i) {
				nw = op(nw, a[i]);
				pre[j][i] = nw;
				if((i & mask) == mask) nw = null;
			}
			nw = null;
			for(int i = n - 1; i >= 0; --i) {
				nw = op(a[i], nw);
				suf[j][i] = nw;
				if((i & mask) == 0) nw = null;
			}
		}
	}
	T get(int l, int r) {
		if(l == r) return pre[0][l];
		int i = 31 - __builtin_clz(l ^ r);
		return op(suf[i][l], pre[i][r]);
	}
};
```

3.6 Dynamic Connectivity Offline

```cpp
typedef pair<int, int> ii;
struct DSU {
	vector<int> p,size,h;
	int sets;
	void build(int n){
		sets=n;
		p.assign(n,0);
		size.assign(n,1);
		for(int i=0;i<n;++i)p[i]=i;
	}
	int get(int a){return (a==p[a]?a:get(p[a]));}
	void unite(int a, int b){
		a=get(a);b=get(b);
		if(a==b)return;
		if(size[a]>size[b])swap(a,b);
		h.push_back(a);
		size[b]+=size[a];
		p[a]=b;sets--;
	}
	void rollback(int s){
		while(sz(h)>s){
			int a=h.back();
			h.pop_back();
			size[p[a]]-=size[a];
			p[a]=a;sets++;
		}
	}
};

// O(q*log(q)*log(n))
enum { ADD, DEL, QUERY };
struct Query { int type,u,v; };
struct DynCon {
	map<ii, int> edges;DSU uf;
	vector<Query> q;
```

```
        vector<int> t;
        void add(int u, int v){
                if(u>v)swap(u,v);
                edges[{u,v}]=sz(q);
                q.push_back({ADD, u, v});
                t.push_back(-1);
        }
        void del(int u, int v){
                if(u>v)swap(u,v);
                int i=edges[{u,v}];
                t[i]=sz(q);
                q.push_back({DEL, u, v});
                t.push_back(i);
        }
        void query(){
                q.push_back({QUERY, -1, -1});
                t.push_back(-1);
        }
        void dnc(int l, int r){
                if(r-l==1){
                        if(q[l].type==QUERY)
                                cout<<uf.sets<<"\n";
                        return;
                }
                int m=l+(r-l)/2,k=sz(uf.h);
                for(int i=m;i<r;++i)
                        if(q[i].type==DEL && t[i]<l)
                                uf.unite(q[i].u, q[i].v);
                dnc(l, m);
                uf.rollback(k);
                for(int i=l;i<m;++i)
                        if(q[i].type==ADD && t[i]>=r)
                                uf.unite(q[i].u, q[i].v);
                dnc(m, r);
                uf.rollback(k);
        }
        void init(int n){
                uf.build(n);
                if(!sz(q))return;
                for(int& ti:t)if(ti==-1)ti=sz(q);
                dnc(0, sz(q));
        }
};
```

3.7 DSU Bipartite

```
// Bipartite graph
// get return the leader and the parity of the distance
    to the leader
typedef pair<int, int> ii;
struct DSU{
        vector<int> p,size,len;
        DSU(int n){
                p.assign(n,0);
                len.assign(n,0);
                size.assign(n,1);
                for(int i=0;i<n;++i)p[i]=i;
        }
        ii get(int a){
                if(a==p[a])return {a, 0};
                ii va=get(p[a]);
                p[a]=va.first;
                len[a]=(len[a]+va.second)%2;
                return {p[a], len[a]};
        }
        void unite(int a, int b){
                ii va=get(a);
                ii vb=get(b);
                if(va.first==vb.first)return;
                if(size[va.first]>size[vb.first])swap(va,
                    vb);
                p[va.first]=vb.first;
                len[va.first]=(va.second+vb.second+1)%2;
                size[vb.first]+=size[va.first];
        }
};
```

3.8 Dynamic Segment Tree

```
// O(q*log(n)), q => queries
typedef long long T;
T null=0, noVal=0;
T oper(T a, T b){return a+b;}
struct Node{
        T val,lz;
        int l,r;
        Node *pl,*pr;
        Node(int ll, int rr){
                val=null;lz=noVal;
                pl=pr=nullptr;
                l=ll;r=rr;
        }
        void update(){
                if(r-l==1)return;
                val=oper(pl->val, pr->val);
        }
        void update(T v){
                val+=((T)(r-l))*v;
                lz+=v;
        }
        void extends(){
                if(r-l!=1 && !pl){
                        int m=(r+l)/2;
                        pl=new Node(l, m);
                        pr=new Node(m, r);
                }
```

```
        }
        void propagate(){
                if(r-l==1)return;
                if(lz==noVal)return;
                pl->update(lz);
                pr->update(lz);
                lz=noVal;
        }
};
typedef Node* PNode;
struct SegTree{
        PNode root;
        SegTree(int l, int r){root=new Node(l, r+1);}

        void upd(PNode x, int l, int r, T v){
                int lx=x->l,rx=x->r;
                if(lx>=r || l>=rx)return;
                if(lx>=l && rx<=r){
                        x->update(v);
                        return;
                }
                x->extends();
                x->propagate();
                upd(x->pl,l,r,v);
                upd(x->pr,l,r,v);
                x->update();
        }

        T get(PNode x, int l, int r){
                int lx=x->l,rx=x->r;
                if(lx>=r || l>=rx)return null;
                if(lx>=l && rx<=r)return x->val;
                x->extends();
                x->propagate();
                T v1=get(x->pl,l,r);
                T v2=get(x->pr,l,r);
                return oper(v1,v2);
        }

        T get(int l, int r){return get(root,l,r+1);}
        void upd(int l, int r, T v){upd(root,l,r+1,v);}
};
```

3.9 Implicit Treap

```
// Treap => Binary Search Tree + Binary Heap
// 1. create a empty root (PTreap root=nullptr;)
// 2. Append the nodes in order (left -> right)
// PTreap tmp=new Treap(x);
// root=merge(root, tmp);

typedef long long T;
typedef unsigned long long u64;
mt19937_64 rng (chrono::steady_clock::now().
        time_since_epoch().count());

T null = 0;
struct Treap{
        Treap *l,*r; // left child, right child
        u64 prior; // random
        T val,sum,lz; // value, sum subtree, lazy
        int sz; // size subtree
        Treap(T v){
                l=r=nullptr;
                prior=rng();
                val=sum=v;
                lz=0;sz=1;
        }
        ~Treap(){
                delete l;
                delete r;
        }
};

typedef Treap* PTreap;
int cnt(PTreap x){return (!x?0:x->sz);}
T sum(PTreap x){return (!x?0:x->sum);}

void update_helper(PTreap x, T v){
        // lz += v
        // val += v
        // sum += v
}

// propagate the lazy
void push(PTreap x){
        if(x && x->lz){ // check x->lz
                if(x->l)update_helper(x->l, 1);
                if(x->r)update_helper(x->r, 1);
                x->lz=0;
        }
}

// updates node with its children information
void pull(PTreap x){
        push(x->l);
        push(x->r);
        x->sz=cnt(x->l)+cnt(x->r)+1;
        x->sum=sum(x->l)+sum(x->r)+x->val;
}

// Updates node value += v
void upd(PTreap x, T v){
        if(!x)return;
        pull(x);
        update_helper(x,v);
}

// O(log(n)) divide the treap in two parts
// [count nodes == left], [the rest of nodes]
pair<PTreap, PTreap> split(PTreap x, int left){
        if(!x)return {nullptr, nullptr};
```

```
		push(x);
		if(cnt(x->l)>=left){
			auto got=split(x->l, left);
			x->l=got.second;
			pull(x);
			return {got.first, x};
		}else{
			auto got=split(x->r, left-cnt(x->l)-1);
			x->r=got.first;
			pull(x);
			return {x, got.second};
		}
}

// O(log(n)) merge two treap
// [nodes treap x ... nodes treap y]
PTreap merge(PTreap x, PTreap y){
		if(!x)return y;
		if(!y)return x;
		push(x);push(y);
		if(x->prior<=y->prior){
			x->r=merge(x->r, y);
			pull(x);
			return x;
		}else{
			y->l=merge(x, y->l);
			pull(y);
			return y;
		}
}

// O(n) print the treap
void dfs(PTreap x){
		if(!x)return;
		push(x);
		dfs(x->l);
		cout<<x->val<<" ";
		dfs(x->r);
}
```

3.10 Implicit Treap Father

```
// Treap => Binary Search Tree + Binary Heap
// 1. create a empty root (PTreap root=nullptr;)
// 2. Append the nodes in order (left -> right)
// PTreap tmp=new Treap(x);
// root=merge(root, tmp);

// si se edita un treap, se tiene que hacer un pullAll
   hasta la raiz
// si no se hace esto, el treap queda con informacion
   pasada

// si se va a modificar un treap, hacer un pushAll para
   bajar los lazy

typedef long long T;
typedef unsigned long long u64;
mt19937_64 rng (chrono::steady_clock::now().
   time_since_epoch().count());

T null = 0;
struct Treap{
		Treap *l,*r,*dad; // left child, right child
		u64 prior; // random
		T val,sum; // value, sum subtree
		int sz; // size subtree
		Treap(T v){
			l=r=dad=nullptr;
			prior=rng();
			val=sum=v;
			sz=1;
		}
		~Treap(){
			delete l;
			delete r;
		}
};

typedef Treap* PTreap;
int cnt(PTreap x){return (!x?0:x->sz);}
T sum(PTreap x){return (!x?0:x->sum);}

// updates node with its children information
void pull(PTreap x){
		x->sz=cnt(x->l)+cnt(x->r)+1;
		x->sum=sum(x->l)+sum(x->r)+x->val;
		if(x->l)x->l->dad=x; //
		if(x->r)x->r->dad=x; //
}

// O(log(n)) divide the treap in two parts
// [count nodes == left], [the rest of nodes]
pair<PTreap, PTreap> split(PTreap x, int left){
		if(!x)return {nullptr, nullptr};
		if(cnt(x->l)>=left){
			auto got=split(x->l, left);
			if(got.first)got.first->dad=nullptr; //
			x->l=got.second;
			x->dad=nullptr; //
			pull(x);
			return {got.first, x};
		}else{
			auto got=split(x->r, left-cnt(x->l)-1);
			if(got.second)got.second->dad=nullptr; //
			x->r=got.first;
			x->dad=nullptr; //
			pull(x);
			return {x, got.second};
		}
}
```

```cpp
// O(log(n)) merge two treap
// [nodes treap x ... nodes treap y]
PTreap merge(PTreap x, PTreap y){
        if(!x)return y;
        if(!y)return x;
        if(x->prior<=y->prior){
                x->r=merge(x->r, y);
                pull(x);
                return x;
        }else{
                y->l=merge(x, y->l);
                pull(y);
                return y;
        }
}

// O(log(n)) propagate the lazy [root->x]
void pushAll(PTreap x){
        if(!x)return;
        pushAll(x->dad);
        push(x);
}

// O(log(n)) update the treap [root->x]
void pullAll(PTreap x){
        if(!x)return;
        pull(x);
        pullAll(x->dad);
}

// O(log(n)) return the root and the position of x (1-
    indexed)
pair<PTreap, int> findRoot(PTreap x){
        pushAll(x);
        int pos=cnt(x->l);
        while(x->dad){
                PTreap f=x->dad;
                if(x==f->r)pos+=cnt(f->l)+1;
                x=f;
        }
        return {x,pos+1};
}
```

3.11 Li Chao

```cpp
// inf max abs value that the function may take
typedef long long ty;

struct Line {
        ty m, b;

        Line(){}
        Line(ty m, ty b): m(m), b(b){}

        ty eval(ty x){return m * x + b;}
};

struct nLiChao{
        // see coments for min

        nLiChao *left = nullptr, *right = nullptr;
        ty l, r;
        Line line;

        nLiChao(ty l, ty r): l(l), r(r){
                line = {0, -inf}; // change to {0, inf};
        }

        // T(Log(Rango)) M(Log(rango))
        void addLine(Line nline){
                ty m = (l + r) >> 1;
                bool lef = nline.eval(l) > line.eval(l);
                    // change > to <
                bool mid = nline.eval(m) > line.eval(m);
                    // change > to <

                if (mid) swap(nline, line);

                if (r == l) return;

                if (lef != mid){
                        if (!left){
                                left = new nLiChao(l, m);
                                left -> line = nline;
                        }

                        else left -> addLine(nline);
                }
                else{
                        if (!right){
                                right = new nLiChao(m +
                                    1, r);
                                right -> line = nline;
                        }

                        else right -> addLine(nline);
                }
        }

        // T(Log(Rango))
        ty get(ty x) {
                ty m = (l + r) >> 1;
                ty op1 = -inf, op2 = -inf; // change to
                    inf

                if(l == r) return line.eval(x);
                else if(x < m){
                        if (left) op1 = left -> get(x);
                        return max(line.eval(x), op1); //
                            change max to min
                }
                else{
                        if (right) op2 = right -> get(x);
                        return max(line.eval(x), op2); //
```

```
                            change max to min
                }
        }
};

int main() {
        // (rango superior) * (pendiente maxima) puede
           desbordarse
        // usar double o long double en el eval para
           estos casos
        // (puede dar problemas de precision)
        nLiChao liChao(0, 1e18);
}
```

3.12 Link Cut Tree

```
// 1-indexed
// All operations are O(log(n))
typedef long long T;
struct SplayTree{
        struct Node{
                int ch[2]={0, 0},p=0;
                T val=0,path=0; // values for path
                T sub=0,vir=0; // values for subtree
                bool flip=0; // values for lazy
        };
        vector<Node> ns;
        SplayTree(int n):ns(n+1){}
        T path(int u){return (u?ns[u].path:0);}
        T subsum(int u){return (u?ns[u].sub:0);}
        void push(int x){
                if(!x)return;
                int l=ns[x].ch[0],r=ns[x].ch[1];
                if(ns[x].flip){
                        ns[l].flip^=1,ns[r].flip^=1;
                        swap(ns[x].ch[0], ns[x].ch[1]);
                        // if the operation is like a
                           segment tree
                        // check swap the values
                        ns[x].flip=0;
                }
        }
        void pull(int x){
                int l=ns[x].ch[0],r=ns[x].ch[1];
                push(l);push(r);
                ns[x].path=max({path(l), path(r), ns[x].
                    val});
                ns[x].sub=ns[x].vir+subsum(l)+subsum(r)+
                    ns[x].val;
        }
        void set(int x, int d, int y){ns[x].ch[d]=y;ns[y
            ].p=x;pull(x);}
        void splay(int x){
                auto dir=[&](int x){
                        int p=ns[x].p;if(!p)return -1;
                        return ns[p].ch[0]==x?0:ns[p].ch
                            [1]==x?1:-1;
                };
                auto rotate=[&](int x){
                        int y=ns[x].p,z=ns[y].p,dx=dir(x)
                            ,dy=dir(y);
                        set(y,dx,ns[x].ch[!dx]);
                        set(x,!dx,y);
                        if(~dy)set(z,dy,x);
                        ns[x].p=z;
                };
                for(push(x);~dir(x);){
                        int y=ns[x].p, z=ns[y].p;
                        push(z);push(y);push(x);
                        int dx=dir(x),dy=dir(y);
                        if(~dy)rotate(dx!=dy?x:y);
                        rotate(x);
                }
        }
};

struct LinkCut:SplayTree{
        LinkCut(int n):SplayTree(n){}
        // return the root of us tree
        int root(int u){
                access(u);splay(u);push(u);
                while(ns[u].ch[0]){u=ns[u].ch[0];push(u)
                    ;}
                return splay(u),u;
        }
        // return the parent of u
        int parent(int u){
                access(u);splay(u);push(u);
                u=ns[u].ch[0];push(u);
                while(ns[u].ch[1]){u=ns[u].ch[1];push(u)
                    ;}
                return splay(u),u;
        }
        int access(int x){
                int u=x,v=0;
                for(;u;v=u,u=ns[u].p){
                        splay(u);
                        int& ov=ns[u].ch[1];
                        ns[u].vir+=ns[ov].sub;
                        ns[u].vir-=ns[v].sub;
                        ov=v;pull(u);
                }
                return splay(x),v;
        }
        // reroot the tree with x as root
        void reroot(int x){
                access(x);ns[x].flip^=1;push(x);
        }
```

```
        // create a edge u->v, u is the child of v
        void link(int u, int v){
                reroot(u);access(v);
                ns[v].vir+=ns[u].sub;
                ns[u].p=v;pull(v);
        }
        // delete the edge u->v, u is the child of v
        void cut(int u, int v){
                int r=root(u);
                reroot(u);access(v);
                ns[v].ch[0]=ns[u].p=0;pull(v);
                reroot(r);
        }
        // delete the edge u->parent(u)
        void cut(int u){
                access(u);
                ns[ns[u].ch[0]].p=0;
                ns[u].ch[0]=0;pull(u);
        }
        int lca(int u, int v){
                if(root(u)!=root(v))return -1;
                access(u);return access(v);
        }
        // return sum of the subtree of u with v as father
        T subtree(int u, int v){
                int r=root(u);
                reroot(v);access(u);
                T ans=ns[u].vir+ns[u].val;
                return reroot(r),ans;
        }
        T path(int u, int v){
                int r=root(u);
                reroot(u);access(v);pull(v);
                T ans=ns[v].path;
                return reroot(r),ans;
        }
        void set(int u, T val){
                access(u);
                ns[u].val=val;
                pull(u);
        }
};
```

3.13 Link Cut Tree Lazy

```
// 1-indexed
// All operations are O(log(n))
typedef long long T;
struct SplayTree{
        struct Node{
                int ch[2]={0, 0},p=0;
                T val=0,path=0,sz=1; // values for path
                T sub=0,vir=0,ssz=0,vsz=0; // values for subtree
                bool flip=0;T lz=0; // values for lazy
        };
        vector<Node> ns;

        SplayTree(int n):ns(n+1){}

        T path(int u){return (u?ns[u].path:0);}
        T size(int u){return (u?ns[u].sz:0);}
        T subsize(int u){return (u?ns[u].ssz:0);}
        T subsum(int u){return (u?ns[u].sub:0);}
        void push(int x){
                if(!x)return;
                int l=ns[x].ch[0],r=ns[x].ch[1];
                if(ns[x].flip){
                        ns[l].flip^=1,ns[r].flip^=1;
                        swap(ns[x].ch[0], ns[x].ch[1]);
                        // if the operation is like a segment tree
                        // check swap the values
                        ns[x].flip=0;
                }
                if(ns[x].lz){ // check the lazy
                        // propagate the lazy
                        ns[x].sub+=ns[x].lz*ns[x].ssz;
                        ns[x].vir+=ns[x].lz*ns[x].vsz;
                        // ...
                }
        }

        void pull(int x){
                int l=ns[x].ch[0],r=ns[x].ch[1];
                push(l);push(r);
                ns[x].sz=size(l)+size(r)+1;
                ns[x].path=max({path(l), path(r), ns[x].val});
                ns[x].sub=ns[x].vir+subsum(l)+subsum(r)+ns[x].val;
                ns[x].ssz=ns[x].vsz+subsize(l)+subsize(r)+1;
        }

        void set(int x, int d, int y){ns[x].ch[d]=y;ns[y].p=x;pull(x);}
        void splay(int x){
                auto dir=[&](int x){
                        int p=ns[x].p;if(!p)return -1;
                        return ns[p].ch[0]==x?0:ns[p].ch[1]==x?1:-1;
                };
                auto rotate=[&](int x){
                        int y=ns[x].p,z=ns[y].p,dx=dir(x),dy=dir(y);
                        set(y,dx,ns[x].ch[!dx]);
                        set(x,!dx,y);
```

```
                                if(~dy)set(z,dy,x);
                                ns[x].p=z;
                        };
                        for(push(x);~dir(x);){
                                int y=ns[x].p, z=ns[y].p;
                                push(z);push(y);push(x);
                                int dx=dir(x),dy=dir(y);
                                if(~dy)rotate(dx!=dy?x:y);
                                rotate(x);
                        }
                }
};

struct LinkCut:SplayTree{
        LinkCut(int n):SplayTree(n){}

        // return the root of us tree
        int root(int u){
                access(u);splay(u);push(u);
                while(ns[u].ch[0]){u=ns[u].ch[0];push(u)
                    ;}
                return splay(u),u;
        }

        // return the parent of u
        int parent(int u){
                access(u);splay(u);push(u);
                u=ns[u].ch[0];push(u);
                while(ns[u].ch[1]){u=ns[u].ch[1];push(u)
                    ;}
                return splay(u),u;
        }

        int access(int x){
                int u=x,v=0;
                for(;u;v=u,u=ns[u].p){
                        splay(u);
                        int& ov=ns[u].ch[1];
                        ns[u].vir+=ns[ov].sub;
                        ns[u].vsz+=ns[ov].ssz;
                        ns[u].vir-=ns[v].sub;
                        ns[u].vsz-=ns[v].ssz;
                        ov=v;pull(u);
                }
                return splay(x),v;
        }

        // reroot the tree with x as root
        void reroot(int x){
                access(x);ns[x].flip^=1;push(x);
        }

        // create a edge u->v, u is the child of v
        void link(int u, int v){
                reroot(u);
                access(v);
                ns[v].vir+=ns[u].sub;
                ns[v].vsz+=ns[u].ssz;
                ns[u].p=v;pull(v);
        }

        // delete the edge u->v, u is the child of v
        void cut(int u, int v){
                int r=root(u);
                reroot(u);
                access(v);
                ns[v].ch[0]=ns[u].p=0;pull(v);
                reroot(r);
        }

        // delete the edge u->parent(u)
        void cut(int u){
                access(u);
                ns[ns[u].ch[0]].p=0;
                ns[u].ch[0]=0;pull(u);
        }

        int lca(int u, int v){
                if(root(u)!=root(v))return -1;
                access(u);return access(v);
        }

        int depth(int u){
                int r=root(u);
                reroot(r);
                access(u);splay(u);push(u);
                return ns[u].sz-1;
        }

        T path(int u, int v){
                int r=root(u);
                reroot(u);access(v);pull(v);
                T ans=ns[v].path;
                return reroot(r),ans;
        }

        void set(int u, T val){
                access(u);
                ns[u].val=val;
                pull(u);
        }

        // update the value of the nodes in the path u->v
             with += val
        void upd(int u, int v, T val){
                int r=root(u);
                reroot(u);access(v);splay(v);
                // change only the lazy
                // ns[v].val+=val;
                reroot(r);
        }

        T comp_size(int u){return ns[root(u)].ssz;}
        T subtree_size(int u){
                int p=parent(u);
```

```
                if(!p)return comp_size(u);
                cut(u);int ans=comp_size(u);
                link(u,p);return ans;
        }
        T subtree_size(int u, int v){ // subtree of u
           with v as father
                int r=root(u);
                reroot(v);access(u);
                T ans=ns[u].vsz+1;
                return reroot(r),ans;
        }

        T comp_sum(int u){return ns[root(u)].sub;}
        T subtree_sum(int u){
                int p=parent(u);
                if(!p)return comp_sum(u);
                cut(u);T ans=comp_sum(u);
                link(u,p);return ans;
        }
        T subtree_sum(int u, int v){ // subtree of u with
            v as father
                int r=root(u);
                reroot(v);access(u);
                T ans=ns[u].vir+ns[u].val; // por el
                    reroot
                return reroot(r),ans;
        }
};
```

3.14 Merge Sort Tree

```
// O(n*log(n)) build
// O(log(n)^2) get
typedef long long T;
struct SegTree{
        int size;
        vector<vector<T>> vals;

        void oper(int x){
                merge(all(vals[2*x+1]), all(vals[2*x+2]),
                     back_inserter(vals[x]));
        }
        SegTree(vector<T>& a){
                size=1;
                while(size<sz(a))size*=2;
                vals.resize(2*size);
                build(a, 0, 0, size);
        }

        void build(vector<T>& a, int x, int lx, int rx){
                if(rx-lx==1){
                        if(lx<sz(a))vals[x]={a[lx]};
                        return;
                }
                int m=(lx+rx)/2;
                build(a, 2*x+1, lx, m);
                build(a, 2*x+2, m, rx);
                oper(x);
        }

        int get(int l, int r,int val, int x, int lx, int
           rx){
                if(lx>=r || l>=rx)return 0;
                if(lx>=l && rx<=r){
                        return upper_bound(all(vals[x]),
                            val)-vals[x].begin();
                }
                int m=(lx+rx)/2;
                int v1=get(l,r,val,2*x+1,lx,m);
                int v2=get(l,r,val,2*x+2,m,rx);
                return v1+v2;
        }

        int get(int l, int r,int val){return get(l,r+1,
           val,0,0,size);}
};
```

3.15 MOs Algorithm

```
// O((n+q)*sq), sq=n^(1/2)
// 1. fill queries[]
// 2. solve(n);
// 3. print ans[]
int sq;
struct query {int l,r,idx;};
bool cmp(query& a, query& b){
        int x=a.l/sq;
        if(a.l/sq!=b.l/sq)return a.l/sq<b.l/sq;
        return (x&1?a.r<b.r:a.r>b.r);
}

vector<query> queries;
vector<ll> ans;

ll act();
void add(int i); // add a[i]
void remove(int i); // remove a[i]

void solve(int n){
        sq=ceil(sqrt(n));
        sort(all(queries), cmp);
        ans.assign(sz(queries),0);
        int l=0,r=-1;
        for(auto [li,ri,i]:queries){
                while(r<ri)add(++r);
                while(l>li)add(--l);
                while(r>ri)remove(r--);
                while(l<li)remove(l++);
                ans[i]=act();
        }
```

```
}
```

3.16 MOs Tree

```
// add LCA
struct LCA{};

vector<vector<int>> adj;
const int maxn=1e5+5;
int ver[2*maxn]; // node at position i in euler tour
int st[maxn]; // start time of v
int ft[maxn]; // finish time of v
int pos=0;
LCA tree;

// O((n+q)*sq), sq=n^(1/2)
// 1. build euler tour and lca
// 2. add queries[]
// if(st[a]>st[b])swap(a,b);
// queries.push_back({st[a]+1,st[b],i});
// 3. solve(n);
// 4. print ans[]
int sq;

void dfs(int u=0, int p=-1){
        ver[pos]=u;
        st[u]=pos++;
        for(int v:adj[u]){
                if(v==p)continue;
                dfs(v,u);
        }
        ver[pos]=u;
        ft[u]=pos++;
}

struct query {int l,r,idx;};
bool cmp(query& a, query& b){
        int x=a.l/sq;
        if(a.l/sq!=b.l/sq)return a.l/sq<b.l/sq;
        return (x&1?a.r<b.r:a.r>b.r);
}

vector<query> queries;
vector<ll> ans;
bool vis[maxn];

ll act();
void add(int u); // add node u
void remove(int u); // remove node u
void ask(int u){
        if(!vis[u])add(u);
        else remove(u);
        vis[u]=!vis[u];
}

void solve(int n){
        sq=ceil(sqrt(n));
        sort(all(queries), cmp);
    ans.resize(sz(queries));
        int l=0,r=-1;
    for(auto [li,ri,i]:queries){
                while(r<ri)ask(ver[++r]);
                while(l>li)ask(ver[--l]);
                while(r>ri)ask(ver[r--]);
                while(l<li)ask(ver[l++]);
                int a=ver[l-1],b=ver[r];
                int c=tree.lca(a,b);
                ask(c);
                ans[i]=act();
                ask(c);
        }
}
```

3.17 MOs Updates

```
// O(q*(s+(n/s)^2) => O(q*(n^(2/3))), s=(2*(n^2))^(1/3) -
    s=n^(2/3)
// 1. fill queries[] and upds[]
// dont confuse index in queries with updates, they are
    different
// the struct upd saves the old value and the new value
// 2. solve(n);
// 3. print ans[]
int sq;
struct upd{int i,old,cur;};
struct query {int l,r,t,idx;};
bool cmp(query& a, query& b){
        int x=a.l/sq;
        if(a.l/sq!=b.l/sq)return a.l/sq<b.l/sq;
        if(a.r/sq!=b.r/sq)return (x&1?a.r<b.r:a.r>b.r);
        return a.t<b.t;
}

vector<query> queries;
vector<upd> upds;
vector<ll> ans;

ll act();
void add(int i); // add a[i]
void remove(int i); // remove a[i]
void update(int i, int v, int l, int r){
        // check if the update is with an active element
        if(l<=i && i<=r){
                remove(i);
                // a[i]=v;
                // ...
                add(i);
        }
        // a[i]=v;
        // ...
```

```
}
void solve(int n){
        sq=ceil(pow(n,2.0/3.0));
        sort(all(queries), cmp);
    ans.resize(sz(queries));
        int l=0,r=-1,t=0;
        for(auto [li,ri,ti,i]:queries){
                while(t<ti)update(upds[t].i,upds[t].cur,l
                    ,r),++t;
                while(t>ti)--t,update(upds[t].i,upds[t].
                    old,l,r);
                while(r<ri)add(++r);
                while(l>li)add(--l);
                while(r>ri)remove(r--);
                while(l<li)remove(l++);
                ans[i]=act();
        }
}
```

3.18 Ordered set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T> using ordered_set = tree<T,
    null_type,less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
template<typename T> using ordered_multiset = tree<T,
    null_type,less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// -------- CONSTRUCTOR -------- //
// 1. Para ordenar por MAX cambiar less<int> por greater<
    int>
// 2. Para multiset cambiar less<int> por less_equal<int>
//      Para borrar siendo multiset:
//      int idx = st.order_of_key(value);
//      st.erase(st.find_by_order(idx));
//      st.swap(st2);
// -------- METHODS --------- //
st.find_by_order(k) // returns pointer to the k-th
    smallest element
st.order_of_key(x)  // returns how many elements are
    smaller than x
st.find_by_order(k) == st.end() // true, if element does
    not exist
```

3.19 Persistent Segment Tree

```
// O(n*log(n)) build
// O(log(n)) get, set
// O((n+q)*log(n)) memory
typedef long long T;
struct Node{
        T val;
        int l,r; // saves the range of the node [l,r]
};
struct SegTree{
        vector<Node> ns;
        vector<int> roots; // roots of the differents
            versions
        T null=0;
        int act=0,size; // act: number of nodes

        T oper(T a, T b){return a+b;}
        SegTree(vector<T>& a, int n){
                size=n;
                roots.push_back(build(a, 0, size));
        }

        void update(int x){
                ns[x].val=oper(ns[ns[x].l].val, ns[ns[x].
                    r].val);
        }

        int newNode(T x){
                Node tmp={x,-1,-1};
                ns.push_back(tmp);
                return act++;
        }

        int newNode(int l, int r){
                Node tmp={null,l,r};
                ns.push_back(tmp);
                update(act);
                return act++;
        }

        int build(vector<T>& a, int l, int r){
                if(r-l==1){return newNode(a[l]);}
                int m=(l+r)/2;
                return newNode(build(a, l, m),build(a, m,
                    r));
        }

        int set(int x, int i, T v, int l, int r){
                if(r-l==1)return newNode(v);
                int m=(l+r)/2;
                if(i<m)return newNode(set(ns[x].l, i, v,
                    l, m), ns[x].r);
                else return newNode(ns[x].l, set(ns[x].r,
                    i, v, m, r));
        }

        T get(int x, int lx, int rx, int l, int r){
                if(lx>=r || l>=rx)return null;
                if(lx>=l && rx<=r)return ns[x].val;
                int m=(lx+rx)/2;
                T v1=get(ns[x].l, lx, m, l, r);
```

```
			T v2=get(ns[x].r, m, rx, l, r);
			return oper(v1,v2);
		}

		T get(int l, int r, int time){return get(roots[time], 0, size, l, r+1);}
		void set(int i, T v, int time){roots.push_back(set(roots[time], i, v, 0, size));}
};
```

3.20 Persistent Segment Tree Lazy

```
// O(n*log(n)) build
// O(log(n)) get, upd
// O((n+q)*log(n)) memory
typedef long long T;
struct Node {
	Node* left = nullptr;
	Node* right = nullptr;
	T val = 0, prop = 0;
};
typedef Node* PNode;
struct PerSegTree {
	vector<PNode> roots{};
	vector<T> vec{};
	int n = 0;

	T op(T a, T b){
		return a+b;
	}

	PNode newKid(PNode& curr) {
		PNode newNode = new Node();
		newNode->left = curr->left;
		newNode->right = curr->right;
		newNode->prop = curr->prop;
		newNode->val = curr->val;
		return newNode;
	}

	void lazy(int i, int j, PNode& curr) {
		if (!curr->prop) return;
		curr->val += ((T)(j - i + 1)) * curr->prop;
		if (i != j) {
			curr->left = newKid(curr->left);
			curr->right = newKid(curr->right);
			curr->left->prop += curr->prop;
			curr->right->prop += curr->prop;
		}
		curr->prop = 0;
	}

	PNode build(int i, int j) {
		PNode newNode = new Node();
		if (i == j) {
			newNode->val = vec[i];
		} else {
			int mid = i + (j - i) / 2;
			PNode leftt = build(i, mid);
			PNode right = build(mid + 1, j);
			newNode->val = op(leftt->val, right->val);
			newNode->left = leftt;
			newNode->right = right;
		}
		return newNode;
	}

	PNode upd(int i, int j, int l, int r, T value, PNode& curr) {
		lazy(i, j, curr);
		if (i >= l && j <= r) {
			PNode newNode = newKid(curr);
			newNode->prop += value;
			lazy(i, j, newNode);
			return newNode;
		}
		if (i > r || j < l) {
			return curr;
		}
		PNode newNode = new Node();
		int mid = i + (j - i) / 2;
		newNode->left = upd(i, mid, l, r, value, curr->left);
		newNode->right = upd(mid + 1, j, l, r, value, curr->right);
		newNode->val = op(newNode->left->val, newNode->right->val);
		return newNode;
	}

	T get(int i, int j, int l, int r, PNode& curr) {
		lazy(i, j, curr);
		if (j < l || r < i) {
			return 0;
		}
		if (i >= l && j <= r) {
			return curr->val;
		}
		int mid = i + (j - i) / 2;
		return op(get(i, mid, l, r, curr->left), get(mid + 1, j, l, r, curr->right));
	}

	// public methods
	void build(vector<T>& vec) {
		if (vec.empty()) return;
		n = vec.size();
		this->vec = vec;
```

```
                auto root = build(0, n - 1);
                roots.push_back(root);
        }

        void upd(int l, int r, T value, int time) {
                roots.push_back(upd(0, n - 1, l, r, value
                    , roots[time]));
        }

        T get(int l, int r, int time) {
                return get(0, n - 1, l, r, roots[time]);
        }

        int size() { return roots.size(); }
};
```

3.21 Polynomial Updates

```
ll gauss(ll x){return (x*(x+1ll))/2ll;}
struct Node{
        ll sum=0; // the nodes value
        ll acum=0; //  count completed levels
        ll cnt=0; // count of updates +1, +2, +3, ...
        void build(ll v){
                acum=cnt=0;
                sum=v;
        }
        void oper(Node& a, Node& b){
                sum=a.sum+b.sum;
                acum=cnt=0;
        }
        void lazy(ll len, ll _acum, ll _cnt){
                sum+=_acum*len+gauss(len)*_cnt;
                acum+=_acum;
                cnt+=_cnt;
        }
};
struct SegTree{
        vector<Node> vals;
        Node null;
        int size;

        SegTree(vector<ll>& a){
                size=1;
                while(size<sz(a))size*=2;
                vals.resize(2*size);
                build(a, 0, 0, size);
        }

        void build(vector<ll>& a, int x, int lx, int rx){
                if(rx-lx==1){
                        if(lx<sz(a))vals[x].build(a[lx]);
                        return;
                }
                int m=(lx+rx)/2;
                build(a, 2*x+1, lx, m);
                build(a, 2*x+2, m, rx);
                vals[x].oper(vals[2*x+1], vals[2*x+2]);
        }

        void propagate(int x, int lx, int rx){
                if(rx-lx==1)return;
                if(vals[x].cnt==0)return;
                int m=(rx+lx)/2;
                vals[2*x+1].lazy(m-lx, vals[x].acum, vals
                    [x].cnt);
                vals[2*x+2].lazy(rx-m, vals[x].acum+ll(m-
                    lx)*vals[x].cnt, vals[x].cnt);
                vals[x].acum=vals[x].cnt=0;
        }

        void upd(int l, int r, ll v, int x, int lx, int
            rx){
                if(rx<=l || r<=lx)return;
                if(l<=lx && rx<=r){
                        vals[x].lazy(rx-lx,v*(lx-l),v);
                        return;
                }
                propagate(x,lx,rx);
                int m=(lx+rx)/2;
                upd(l,r,v,2*x+1,lx,m);
                upd(l,r,v,2*x+2,m,rx);
                vals[x].oper(vals[2*x+1], vals[2*x+2]);
        }

        ll get(int l, int r, int x, int lx, int rx){
                if(rx<=l || r<=lx)return null.sum;
                if(l<=lx && rx<=r)return vals[x].sum;
                propagate(x,lx,rx);
                int m=(lx+rx)/2;
                ll v1=get(l,r,2*x+1,lx,m);
                ll v2=get(l,r,2*x+2,m,rx);
                return v1+v2;
        }

        ll get(int l, int r){return get(l,r+1,0,0,size);}
        void upd(int l, int r, ll v){upd(l,r+1,v,0,0,size
            );}
        // v es la cantidad de veces que se aplica la
            operacion +1, +2, +3
};
```

3.22 Segment Tree Iterativo

```
struct segtree{
        int n; vl v; ll nulo = 0;
        ll op(ll a, ll b) {return a + b;}
        segtree(int n) : n(n) {v = vl(2*n, nulo);}
        segtree(vl &a) : n(sz(a)), v(2*n){
```

```
                for(int i = 0; i<n; i++) v[n + i] = a[i];
                for (int i = n-1; i>=1; --i) v[i] = op(v[
                    i<<1], v[i<<1|1]);
        }

        void upd(int k, ll nv){
                for (v[k += n] = nv; k > 1; k >>= 1) v[k
                    >>1] = op(v[k], v[k^1]);
        }

        ll get(int l, int r){
                ll vl = nulo, vr = nulo;
                for (l += n, r += n+1; l < r; l >>= 1, r
                    >>= 1){
                        if (l&1) vl = op(vl, v[l++]);
                        if (r&1) vr = op(v[--r], vr);
                }
                return op(vl, vr);
        }
};
```

3.23 Segment Tree Recursivo

```
typedef long long T;
struct SegTree{
        vector<T> vals,lazy;
        T null=0,nolz=0;
        int size;

        T op(T a, T b){return a+b;}
        SegTree(vector<T>& a){
                size=1;
                while(size<sz(a))size*=2;
                vals.resize(2*size);
                lazy.assign(2*size, nolz);
                build(a, 0, 0, size);
        }

        void build(vector<T>& a, int x, int lx, int rx){
                if(rx-lx==1){
                        if(lx<sz(a))vals[x]=a[lx];
                        return;
                }
                int m=(lx+rx)/2;
                build(a, 2*x+1, lx, m);
                build(a, 2*x+2, m, rx);
                vals[x]=op(vals[2*x+1], vals[2*x+2]);
        }

        void propagate(int x, int lx, int rx){
                if(rx-lx==1)return;
                if(lazy[x]==nolz)return;
                int m=(lx+rx)/2;
                lazy[2*x+1]+=lazy[x];
                vals[2*x+1]+=lazy[x]*((T)(m-lx));
                lazy[2*x+2]+=lazy[x];
                vals[2*x+2]+=lazy[x]*((T)(rx-m));
                lazy[x]=nolz;
        }

        void upd(int l, int r, T v,int x, int lx, int rx)
            {
                if(rx<=l || r<=lx)return;
                if(l<=lx && rx<=r){
                        lazy[x]+=v;
                        vals[x]+=v*((T)(rx-lx));
                        return;
                }
                propagate(x,lx,rx);
                int m=(lx+rx)/2;
                upd(l,r,v,2*x+1,lx,m);
                upd(l,r,v,2*x+2,m,rx);
                vals[x]=op(vals[2*x+1], vals[2*x+2]);
        }

        void set(int i, T v, int x, int lx, int rx){
                if(rx-lx==1){
                        vals[x]=v;
                        return;
                }
                propagate(x,lx,rx);
                int m=(lx+rx)/2;
                if(i<m)set(i,v,2*x+1,lx,m);
                else set(i,v,2*x+2,m,rx);
                vals[x]=op(vals[2*x+1], vals[2*x+2]);
        }

        T get(int l, int r, int x, int lx, int rx){
                if(rx<=l || r<=lx)return null;
                if(l<=lx && rx<=r)return vals[x];
                propagate(x,lx,rx);
                int m=(lx+rx)/2;
                T v1=get(l,r,2*x+1,lx,m);
                T v2=get(l,r,2*x+2,m,rx);
                return op(v1,v2);
        }

        T get(int l, int r){return get(l,r+1,0,0,size);}
        void upd(int l, int r, T v){upd(l,r+1,v,0,0,size)
            ;}
        void set(int i, T val){set(i,val,0,0,size);}
};
```

3.24 Segment Tree 2D

```
// O(n^2*log(n^2)) build
// O(log(n)^2) get, set
const int N=1000+1;
typedef int T;
T st[2*N][2*N];
```

```cpp
struct SegTree{
	int n,m,neutro=0;
	T op(T a, T b){return a+b;}
	SegTree(int n, int m): n(n), m(m){
		for(int i=0;i<2*n;++i)for(int j=0;j<2*m
		    ;++j)st[i][j]=neutro;
	}
	SegTree(vector<vector<T>>& a): n(sz(a)), m(n ? sz
	    (a[0]) : 0){ build(a); }

	void build(vector<vector<T>>& a){
		for(int i=0;i<n;++i)for(int j=0;j<m;++j)
		    st[i+n][j+m]=a[i][j];
		for(int i=0;i<n;++i)for(int j=m-1;j>=1;--
		    j)st[i+n][j]=op(st[i+n][j<<1], st[i+n
		    ][j<<1|1]);
		for(int i=n-1;i>=1;--i)for(int j=0;j<2*m
		    ;++j)st[i][j]=op(st[i<<1][j], st[i
		    <<1|1][j]);
	}

	void set(int x, int y, T v){
		st[x+n][y+m]=v;
		for(int j=y+m;j>1;j>>=1)st[x+n][j>>1]=op(
		    st[x+n][j], st[x+n][j^1]);
		for(int i=x+n;i>1;i>>=1)for(int j=y+m;j;j
		    >>=1)st[i>>1][j]=op(st[i][j], st[i^1][
		    j]);
	}

	T get(int x0, int y0, int x1, int y1){
		T r=neutro;
		for(int i0=x0+n,i1=x1+n+1;i0<i1;i0>>=1,i1
		    >>=1){
			int t[4],q=0;
			if(i0&1)t[q++]=i0++;
			if(i1&1)t[q++]=--i1;
			for(int k=0;k<q;++k)for(int j0=y0
			    +m,j1=y1+m+1;j0<j1;j0>>=1,j1
			    >>=1){
				if(j0&1)r=op(r,st[t[k]][
				    j0++]);
				if(j1&1)r=op(r,st[t[k
				    ]][--j1]);
			}
		}
		return r;
	}
};
```

3.25 Segment Tree Beats

```cpp
// O(n*log(n)) build
// O(log(n)) get, upd
// updMax[l,r] -> ai = max(ai, v)
// updMin[l,r] -> ai = min(ai, v)
// updAdd[l,r] -> ai = ai + v
// get[l,r] -> return sum of the range [l,r]
typedef long long T;
T null=0,noVal=0;
T INF=1e18;
struct Node{
	T sum,lazy;
	T max1,max2,maxc;
	T min1,min2,minc;

	void build(T x){
		sum=max1=min1=x;
		maxc=minc=1;
		lazy=noVal;
		max2=-INF;
		min2=INF;
	}

	void oper(Node& a, Node& b){
		sum=a.sum+b.sum;

		if(a.max1>b.max1){
			max1=a.max1;
			maxc=a.maxc;
			max2=max(a.max2, b.max1);
		}else if(a.max1<b.max1){
			max1=b.max1;
			maxc=b.maxc;
			max2=max(b.max2, a.max1);
		}else{
			max1=a.max1;
			maxc=a.maxc+b.maxc;
			max2=max(a.max2, b.max2);
		}

		if(a.min1<b.min1){
			min1=a.min1;
			minc=a.minc;
			min2=min(a.min2, b.min1);
		}else if(a.min1>b.min1){
			min1=b.min1;
			minc=b.minc;
			min2=min(b.min2, a.min1);
		}else{
			min1=a.min1;
			minc=a.minc+b.minc;
			min2=min(a.min2, b.min2);
		}
	}
};
struct SegTree{
	vector<Node> vals;
	int size;

	SegTree(vector<T>& a){
```

```
		size=1;
		while(size<sz(a))size*=2;
		vals.resize(2*size);
		build(a, 0, 0, size);
	}

	void build(vector<T>& a, int x, int lx, int rx){
		if(rx-lx==1){
			if(lx<sz(a))vals[x].build(a[lx]);
			return;
		}
		int m=(lx+rx)/2;
		build(a, 2*x+1, lx, m);
		build(a, 2*x+2, m, rx);
		vals[x].oper(vals[2*x+1], vals[2*x+2]);
	}

	void propagateMax(T v, int x, int lx, int rx){
		if(vals[x].min1>=v)return;
		vals[x].sum-=vals[x].min1*vals[x].minc;
		vals[x].min1=v;
		vals[x].sum+=vals[x].min1*vals[x].minc;
		if(rx-lx==1){
			vals[x].max1=v;
		}else{
			if(v>=vals[x].max1){
				vals[x].max1=v;
			}else if(v>vals[x].max2){
				vals[x].max2=v;
			}
		}
	}

	void propagateMin(T v, int x, int lx, int rx){
		if(vals[x].max1<=v)return;
		vals[x].sum-=vals[x].max1*vals[x].maxc;
		vals[x].max1=v;
		vals[x].sum+=vals[x].max1*vals[x].maxc;
		if(rx-lx==1){
			vals[x].min1=v;
		}else{
			if(v<=vals[x].min1){
				vals[x].min1=v;
			}else if(v<vals[x].min2){
				vals[x].min2=v;
			}
		}
	}

	void propagateAdd(T v, int x, int lx, int rx){
		vals[x].sum+=v*((T)(rx-lx));
		vals[x].lazy+=v;
		vals[x].max1+=v;
		vals[x].min1+=v;
		if(vals[x].max2!=-INF)vals[x].max2+=v;
		if(vals[x].min2!=INF)vals[x].min2+=v;
	}

	void propagate(int x, int lx, int rx){
		if(rx-lx==1)return;
		int m=(lx+rx)/2;
		if(vals[x].lazy!=noVal){
			propagateAdd(vals[x].lazy, 2*x+1,
			     lx, m);
			propagateAdd(vals[x].lazy, 2*x+2,
			     m, rx);
			vals[x].lazy=noVal;
		}

		propagateMin(vals[x].max1, 2*x+1, lx, m);
		propagateMin(vals[x].max1, 2*x+2, m, rx);

		propagateMax(vals[x].min1, 2*x+1, lx, m);
		propagateMax(vals[x].min1, 2*x+2, m, rx);
	}

	void updAdd(int l, int r, T v,int x, int lx, int
	    rx){
		if(lx>=r || l>=rx)return;
		if(lx>=l && rx<=r){
			propagateAdd(v, x, lx, rx);
			return;
		}
		propagate(x,lx,rx);
		int m=(lx+rx)/2;
		updAdd(l,r,v,2*x+1,lx,m);
		updAdd(l,r,v,2*x+2,m,rx);
		vals[x].oper(vals[2*x+1], vals[2*x+2]);
	}

	void updMax(int l, int r, T v,int x, int lx, int
	    rx){
		if(lx>=r || l>=rx || vals[x].min1>v)
		    return;
		if(lx>=l && rx<=r && vals[x].min2>v){
			propagateMax(v, x, lx, rx);
			return;
		}
		propagate(x,lx,rx);
		int m=(lx+rx)/2;
		updMax(l,r,v,2*x+1,lx,m);
		updMax(l,r,v,2*x+2,m,rx);
		vals[x].oper(vals[2*x+1], vals[2*x+2]);
	}

	void updMin(int l, int r, T v,int x, int lx, int
	    rx){
		if(lx>=r || l>=rx || vals[x].max1<v)
		    return;
		if(lx>=l && rx<=r && vals[x].max2<v){
			propagateMin(v, x, lx, rx);
			return;
		}
```

```
			propagate(x,lx,rx);
			int m=(lx+rx)/2;
			updMin(l,r,v,2*x+1,lx,m);
			updMin(l,r,v,2*x+2,m,rx);
			vals[x].oper(vals[2*x+1], vals[2*x+2]);
		}

		T get(int l, int r, int x, int lx, int rx){
			if(lx>=r || l>=rx)return null;
			if(lx>=l && rx<=r)return vals[x].sum;
			propagate(x,lx,rx);
			int m=(lx+rx)/2;
			T v1=get(l,r,2*x+1,lx,m);
			T v2=get(l,r,2*x+2,m,rx);
			return v1+v2;
		}

		T get(int l, int r){return get(l,r+1,0,0,size);}
		void updAdd(int l, int r, T v){updAdd(l,r+1,v
		    ,0,0,size);}
		void updMin(int l, int r, T v){updMin(l,r+1,v
		    ,0,0,size);}
		void updMax(int l, int r, T v){updMax(l,r+1,v
		    ,0,0,size);}
};
```

3.26 Sparse Table

```
// O(n*log(n)) build
// O(1) get
typedef long long T;
T op(T a, T b); // max, min, gcd ...
struct Table{
	vector<vector<T>> st;
	Table(vector<T>& v){
		st.clear();
		int n=v.size();
		st.push_back(v);
		for(int j=1;(1<<j)<=n;++j){
			st.push_back(vector<T>(n));
			for(int i=0;i+(1<<(j-1))<n;++i){
				st[j][i]=op(st[j-1][i],st
				    [j-1][i+(1<<(j-1))]);
			}
		}
	}
	T get(int l, int r){
		int j=31-__builtin_clz(r-l+1);
		return op(st[j][l], st[j][r-(1<<j)+1]);
	}
};
```

3.27 Sparse Table 2D

```
// O(n*m*log(n)*log(m)) build
// O(1) get
typedef int T;
const int maxn = 1000, logn = 10;
T st[logn][maxn][logn][maxn];
int lg2[maxn+1];
T op(T a, T b); // min, max, gcd...

void build(int n, int m, vector<vector<T>>& a){
	for(int i=2;i<=max(n,m);++i)lg2[i]=lg2[i/2]+1;
	for(int i=0;i<n;++i){
		for(int j=0;j<m;++j)
			st[0][i][0][j]=a[i][j];
		for(int k2=1;k2<logn;++k2)
			for(int j=0;j+(1<<(k2-1))<m;++j)
				st[0][i][k2][j]=op(st[0][
				    i][k2-1][j], st[0][i][
				    k2-1][j+(1<<(k2-1))]);
	}
	for(int k1=1;k1<logn;++k1)
		for(int i=0;i<n;++i)
			for(int k2=0;k2<logn;++k2)
				for(int j=0;j<m;++j)
					st[k1][i][k2][j]=
					    op(st[k1-1][i
					    ][k2][j], st[
					    k1-1][i+(1<<(
					    k1-1))][k2][j
					    ]);
}

T get(int x1, int y1, int x2, int y2){
	x2++;y2++;
	int a=lg2[x2-x1];
	int b=lg2[y2-y1];
	return op(
		op(st[a][x1][b][y1],
			st[a][x2-(1<<a)][b][y1]),
		op(st[a][x1][b][y2-(1<<b)],
			st[a][x2-(1<<a)][b][y2-(1<<b)])
	);
}
```

3.28 Sqrt Descomposition

```
// O(n) build
// O(n/b+b) get, set
typedef long long T;
struct SQRT{
	int b; // check b
	vector<T> a,bls;
```

```
	SQRT(vector<T>& arr, int n){
		b=ceil(sqrt(n));a=arr;
		bls.assign(b, 0);
		for(int i=0;i<n;++i){
			bls[i/b]+=a[i];
		}
	}
	void set(int x, int v){
		bls[x/b]-=a[x];
		a[x]=v;
		bls[x/b]+=a[x];
	}
	T get(int r){
		T res=0;
		for(int i=0;i<r/b;++i){res+=bls[i];}
		for(int i=(r/b)*b;i<r;++i){res+=a[i];}
		return res;
	}
	T get(int l, int r){
		return get(r+1)-get(l);
	}
};
```

3.29 Treap

```
// Treap => Binary Search Tree + Binary Heap
// 1. create a empty root (PTreap root=nullptr;)
// 2. Append the nodes in asc order
// PTreap tmp=new Treap(x);
// root=merge(root, tmp);

typedef long long T;
typedef unsigned long long u64;
mt19937_64 rng (chrono::steady_clock::now().
    time_since_epoch().count());

T null = 0;
struct Treap{
	Treap *l,*r,*dad; // left child, right child
	u64 prior; // random
	T val; // value
	int sz; // size subtree
	Treap(T v){
		l=r=nullptr;
		prior=rng();
		val=v;sz=1;
	}
	~Treap(){
		delete l;
		delete r;
	}
};

typedef Treap* PTreap;
int cnt(PTreap x){return (!x?0:x->sz);}

// updates node with its children information
void pull(PTreap x){
	x->sz=cnt(x->l)+cnt(x->r)+1;
	if(x->l)x->l->dad=x;
	if(x->r)x->r->dad=x;
}

// O(log(n)) divide the treap in two parts
// [nodes value <= key], [nodes value > key]
pair<PTreap, PTreap> split(PTreap x, T key){
	if(!x)return {nullptr, nullptr};
	if(x->val>key){
		auto got=split(x->l, key);
		x->l=got.second;
		pull(x);
		return {got.first, x};
	}else{
		auto got=split(x->r, key);
		x->r=got.first;
		pull(x);
		return {x, got.second};
	}
}

// O(log(n)) merge two treap
// if all values in treap x < all values in treap y
PTreap merge(PTreap x, PTreap y){
	if(!x)return y;
	if(!y)return x;
	if(x->prior<=y->prior){
		x->r=merge(x->r, y);
		pull(x);
		return x;
	}else{
		y->l=merge(x, y->l);
		pull(y);
		return y;
	}
}

// O(n*log(n))
// Combine two treap into one
PTreap combine(PTreap x, PTreap y){
	if(!x)return y;
	if(!y)return x;
	if(x->prior<y->prior)swap(x, y);
	auto z=split(y, x->val);
	x->r=combine(x->r, z.second);
	x->l=combine(z.first, x->l);
	return x;
}

// O(log(n))
// return kth element - indexed 0
T kth(PTreap& x, int k){
```

```
        if(!x)return null;
        if(k==cnt(x->l))return x->val;
        if(k<cnt(x->l))return kth(x->l, k);
        return kth(x->r, k-cnt(x->l)-1);
}

// O(log(n))
// return {index, val}
pair<int, T> lower_bound(PTreap x, T key){
        if(!x)return {0, null};
        if(x->val<key){
                auto y=lower_bound(x->r, key);
                y.first+=cnt(x->l)+1;
                return y;
        }
        auto y=lower_bound(x->l, key);
        if(y.first==cnt(x->l))y.second=x->val;
        return y;
}

// O(n) print the treap
void dfs(PTreap x){
        if(!x)return;
        dfs(x->l);
        cout<<x->val<<" ";
        dfs(x->r);
}
```

3.30 Trie Bit

```
struct node{
int childs[2]{-1, -1};
};

struct TrieBit{
        vector<node> nds;
        vi passNums;

        TrieBit(){
                nds.pb(node());
                passNums.pb(0);
        }

        void insert(int num){
                int cur = 0;
                for(int i = 30; i >= 0; i--){
                        bool bit = (num >> i) & 1;

                        if(nds[cur].childs[bit] == -1){
                                nds[cur].childs[bit] =
                                    nds.size();
                                nds.pb(node());
                                passNums.pb(0);
                        }

                        passNums[cur]++;
                        cur = nds[cur].childs[bit];
                }

                passNums[cur]++;
        }

        void remove(int num){
                int cur = 0;
                for(int i = 30; i >= 0; i--){
                        bool bit = (num >> i) & 1;
                        passNums[cur]--;
                        cur = nds[cur].childs[bit];
                }

                passNums[cur]--;
        }

        int maxXor(int num){
                int ans = 0;
                int cur = 0;
                for(int i = 30; i >= 0; i--){
                        bool bit = (num >> i) & 1;

                        int n1 = nds[cur].childs[!bit];
                        if (n1 != -1 && passNums[n1]){
                                ans += (1 << i);
                                bit = !bit;
                        }

                        cur = nds[cur].childs[bit];
                }

                return ans;
        }
};
```

3.31 Two Stacks

```
// O(1) push, pop, get
typedef long long T;
struct Node{T val,acum;};
struct TwoStacks{
        stack<Node> s1,s2;

        void push(T x){
                Node tmp={x,x};
                if(!s2.empty()){
                        // tmp.acum + s2.top().acum
                }
                s2.push(tmp);
        }

        void pop(){
                if(s1.empty()){
                        while(!s2.empty()){
                                Node tmp=s2.top();
                                if(s1.empty()){
```

```
                                        // tmp.acum = tmp
                                            .val
                                }else{
                                        // tmp.acum + s1.
                                            top().acum
                                }
                                s1.push(tmp);
                                s2.pop();
                        }
                }
                s1.pop();
        }
        bool get(){
                if(s1.empty() && s2.empty())return false;
                else if(!s1.empty() && s2.empty()){
                        return true; // eval s1.top();
                }else if(s1.empty() && !s2.empty()){
                        return true; // eval s2.top();
                }else{
                        return true; // eval s1.top() +
                            s2.top()
                }
        }
};
```

3.32 Wavelet Tree

```
const int maxn = 1e5+5;
const int maxv = 1e9;
const int minv = -1e9;

// O(n*log(n)) build
// O(log(n)) kth, lte, cnt, sum
// 1. int a[maxn];
// 2. WaveletTree wt;
// 3. fill a[1;n]
// 4. wt.build(a+1, a+n+1, minv, maxv);
struct WaveletTree { // indexed 1
        int lo, hi;
        WaveletTree *l, *r;
        int *b, bsz, csz;
        ll *c;

        WaveletTree() {
                hi=bsz=csz=0;
                l=r=NULL;
                lo=1;
        }

        void build(int *from, int *to, int x, int y){
                lo=x,hi=y;
                if(from>=to)return;
                int mid=lo+(hi-lo)/2;
                auto f=[mid](int x){return x<=mid;};
                b=(int*)malloc((to-from+2)*sizeof(int));
                bsz=0;
                b[bsz++]=0;
                c=(ll*)malloc((to-from+2)*sizeof(ll));
                csz=0;
                c[csz++]=0;
                for(auto it=from;it!=to;++it){
                        b[bsz]=(b[bsz-1]+f(*it));
                        c[csz]=(c[csz-1]+(*it));
                        bsz++;csz++;
                }
                if(hi==lo)return;
                auto pivot=stable_partition(from, to, f);
                l=new WaveletTree();
                l->build(from, pivot, lo, mid);
                r=new WaveletTree();
                r->build(pivot, to, mid+1, hi);
        }

        //kth smallest element in [l, r]
        int kth(int l, int r, int k){
                if(l>r)return 0;
                if(lo==hi)return lo;
                int inLeft=b[r]-b[l-1], lb=b[l-1], rb=b[r
                    ];
                if(k<=inLeft)return this->l->kth(lb+1, rb
                    , k);
                return this->r->kth(l-lb, r-rb, k-inLeft)
                    ;
        }

        //count of numbers in [l, r] Less than or equal
            to k
        int lte(int l, int r, int k){
                if(l>r || k<lo)return 0;
                if(hi<=k)return r-l+1;
                int lb=b[l-1], rb=b[r];
                return this->l->lte(lb+1, rb, k)+this->r
                    ->lte(l-lb, r-rb, k);
        }

        //count of numbers in [l, r] equal to k
        int count(int l, int r, int k){
                if(l>r || k<lo || k>hi)return 0;
                if(lo==hi)return r-l+1;
                int lb=b[l-1], rb=b[r];
                int mid=(lo+hi)>>1;
                if(k<=mid)return this->l->count(lb+1, rb,
                     k);
                return this->r->count(l-lb, r-rb, k);
        }

        //sum of numbers in [l ,r] less than or equal to
            k
        ll sum(int l, int r, int k){
                if(l>r || k<lo)return 0;
```

```
                if(hi<=k)return c[r]-c[l-1];
                int lb=b[l-1], rb=b[r];
                return this->l->sum(lb+1, rb, k)+this->r
                    ->sum(l-lb, r-rb, k);
        }
        ~WaveletTree(){
                delete l;
                delete r;
        }
};
```

4 Flujos

4.1 Blossom

```
// O(|E||V|^2)
struct network {
  struct struct_edge { int v; struct_edge * n; };
  typedef struct_edge* edge;
  int n;
  struct_edge pool[MAXE]; ///2*n*n;
  edge top;
  vector<edge> adj;
  queue<int> q;
  vector<int> f, base, inq, inb, inp, match;
  vector<vector<int>> ed;
  network(int n) : n(n), match(n, -1), adj(n), top(pool),
      f(n), base(n),
                   inq(n), inb(n), inp(n), ed(n, vector<
                       int>(n)) {}
  void add_edge(int u, int v) {
    if(ed[u][v]) return;
    ed[u][v] = 1;
    top->v = v, top->n = adj[u], adj[u] = top++;
    top->v = u, top->n = adj[v], adj[v] = top++;
  }
  int get_lca(int root, int u, int v) {
    fill(inp.begin(), inp.end(), 0);
    while(1) {
      inp[u = base[u]] = 1;
      if(u == root) break;
      u = f[ match[u] ];
    }
    while(1) {
      if(inp[v = base[v]]) return v;
      else v = f[ match[v] ];
    }
  }
  void mark(int lca, int u) {
    while(base[u] != lca) {
      int v = match[u];
      inb[ base[u ]] = 1;
      inb[ base[v] ] = 1;
      u = f[v];
      if(base[u] != lca) f[u] = v;
    }
  }
  void blossom_contraction(int s, int u, int v) {
    int lca = get_lca(s, u, v);
    fill(inb.begin(), inb.end(), 0);
    mark(lca, u); mark(lca, v);
    if(base[u] != lca) f[u] = v;
    if(base[v] != lca) f[v] = u;
    for(int u = 0; u < n; u++)
      if(inb[base[u]]) {
        base[u] = lca;
        if(!inq[u]) {
          inq[u] = 1;
          q.push(u);
        }
      }
  }
  int bfs(int s) {
    fill(inq.begin(), inq.end(), 0);
    fill(f.begin(), f.end(), -1);
    for(int i = 0; i < n; i++) base[i] = i;
    q = queue<int>();
    q.push(s);
    inq[s] = 1;
    while(q.size()) {
      int u = q.front(); q.pop();
      for(edge e = adj[u]; e; e = e->n) {
        int v = e->v;
        if(base[u] != base[v] && match[u] != v) {
          if((v == s) || (match[v] != -1 && f[match[v]]
              != -1))
            blossom_contraction(s, u, v);
          else if(f[v] == -1) {
            f[v] = u;
            if(match[v] == -1) return v;
            else if(!inq[match[v]]) {
              inq[match[v]] = 1;
              q.push(match[v]);
            }
          }
        }
      }
    }
    return -1;
  }
  int doit(int u) {
    if(u == -1) return 0;
    int v = f[u];
    doit(match[v]);
    match[v] = u; match[u] = v;
    return u != -1;
  }
```

```
  /// (i < net.match[i]) => means match
  int maximum_matching() {
    int ans = 0;
    for(int u = 0; u < n; u++)
      ans += (match[u] == -1) && doit(bfs(u));
    return ans;
  }
};
```

4.2 Dinic

```
// O(|E|*|V|^2)
struct edge { ll v, cap, inv, flow, ori; };
struct network {
	ll n, s, t;
	vector<ll> lvl;
	vector<vector<edge>> g;
	network(ll n) : n(n), lvl(n), g(n) {}
	void add_edge(int u, int v, ll c) {
		g[u].push_back({v, c, sz(g[v]), 0, 1});
		g[v].push_back({u, 0, sz(g[u])-1, c, 0});
	}
	bool bfs() {
		fill(lvl.begin(), lvl.end(), -1);
		queue<ll> q;
		lvl[s] = 0;
		for(q.push(s); q.size(); q.pop()) {
			ll u = q.front();
			for(auto &e : g[u]) {
				if(e.cap > 0 && lvl[e.v] == -1) {
					lvl[e.v] = lvl[u]+1;
					q.push(e.v);
				}
			}
		}
		return lvl[t] != -1;
	}
	ll dfs(ll u, ll nf) {
		if(u == t) return nf;
		ll res = 0;
		for(auto &e : g[u]) {
			if(e.cap > 0 && lvl[e.v] == lvl[u]+1) {
				ll tf = dfs(e.v, min(nf, e.cap));
				res += tf; nf -= tf; e.cap -= tf;
				g[e.v][e.inv].cap += tf;
				g[e.v][e.inv].flow -= tf;
				e.flow += tf;
				if(nf == 0) return res;
			}
		}
		if(!res) lvl[u] = -1;
		return res;
	}
	ll max_flow(ll so, ll si, ll res = 0) {
		s = so; t = si;
		while(bfs()) res += dfs(s, LONG_LONG_MAX);
		return res;
	}
	void min_cut(){
		queue<ll> q;
		vector<bool> vis(n, 0);
		vis[s] = 1;
		for(q.push(s); q.size(); q.pop()) {
			ll u = q.front();
			for(auto &e : g[u]) {
				if(e.cap > 0 && !vis[e.v]) {
					q.push(e.v);
					vis[e.v] = 1;
				}
			}
		}
		vii ans;
		for (int i = 0; i<n; i++){
			for (auto &e : g[i]){
				if (vis[i] && !vis[e.v] && e.ori){
					ans.push_back({i+1, e.v+1});
				}
			}
		}
		for (auto [x, y] : ans) cout << x << ' ' << y << ln;
	}
	bool dfs2(vi &path, vector<bool> &vis, int u){
		vis[u] = 1;
		for (auto &e : g[u]){
			if (e.flow > 0 && e.ori && !vis[e.v]){
				if (e.v == t || dfs2(path, vis, e.v)){
					path.push_back(e.v);
					e.flow = 0;
					return 1;
				}
			}
		}
		return 0;
	}
	void disjoint_paths(){
		vi path;
```

```
                vector<bool> vis(n, 0);
                while (dfs2(path, vis, s)){
                        path.push_back(s);
                        reverse(all(path));
                        cout << sz(path) << ln;
                        for (int v : path) cout << v+1 << ' ';
                        cout << ln;
                        path.clear(); vis.assign(n, 0);
                }
        }
};
```

4.3 L-R Flow

```
const long long inf = 1LL << 61;
struct Dinic {
  struct edge {
    int to, rev;
    long long flow, w;
    int id;
  };
  int n, s, t, mxid;
  vector<int> d, flow_through;
  vector<int> done;
  vector<vector<edge>> g;
  Dinic() {}
  Dinic(int _n) {
    n = _n + 10;
    mxid = 0;
    g.resize(n);
  }
  void add_edge(int u, int v, long long w, int id = -1) {
    edge a = {v, (int)g[v].size(), 0, w, id};
    edge b = {u, (int)g[u].size(), 0, 0, -1};//for bidirectional edges cap(b) = w
    g[u].emplace_back(a);
    g[v].emplace_back(b);
    mxid = max(mxid, id);
  }
  bool bfs() {
    d.assign(n, -1);
    d[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto &e : g[u]) {
        int v = e.to;
        if (d[v] == -1 && e.flow < e.w) d[v] = d[u] + 1, q.push(v);
      }
    }
    return d[t] != -1;
  }
  long long dfs(int u, long long flow) {
    if (u == t) return flow;
    for (int &i = done[u]; i < (int)g[u].size(); i++) {
      edge &e = g[u][i];
      if (e.w <= e.flow) continue;
      int v = e.to;
      if (d[v] == d[u] + 1) {
        long long nw = dfs(v, min(flow, e.w - e.flow));
        if (nw > 0) {
          e.flow += nw;
          g[v][e.rev].flow -= nw;
          return nw;
        }
      }
    }
    return 0;
  }
  long long max_flow(int _s, int _t) {
    s = _s;
    t = _t;
    long long flow = 0;
    while (bfs()) {
      done.assign(n, 0);
      while (long long nw = dfs(s, inf)) flow += nw;
    }
    flow_through.assign(mxid + 10, 0);
    for(int i = 0; i < n; i++) for(auto e : g[i]) if(e.id >= 0) flow_through[e.id] = e.flow;
    return flow;
  }
};
//flow_through[i] = extra flow beyond 'low' sent through edge i
struct LR_Flow {
  Dinic F;
  int n, s, t;
  struct edge {
    int u, v, l, r, id;
  };
  vector<edge> edges;
  LR_Flow() {}
  LR_Flow(int _n) {
    n = _n + 10;
    s = n - 2, t = n - 1;;
    edges.clear();
  }
  void add_edge(int u, int v, int l, int r, int id = -1) {
    assert(0 <= l && l <= r);
    edges.push_back({u, v, l, r, id});
  }
  bool feasible(int _s = -1, int _t = -1, int L = -1, int R = -1) {
```

```
    if (L != -1) edges.push_back({_t, _s, L, R, -1});
    F = Dinic(n);
    long long target = 0;
    for (auto e : edges) {
      int u = e.u, v = e.v, l = e.l, r = e.r, id = e.id;
      if (l != 0) {
        F.add_edge(s, v, l);
        F.add_edge(u, t, l);
        target += l;
      }
      F.add_edge(u, v, r - l, id);
    }
    auto ans = F.max_flow(s, t);
    if (L != -1) edges.pop_back();
    if (ans < target) return 0; //not feasible
    return 1;
  }
  int max_flow(int _s, int _t) { //-1 means flow is not
      feasible
    int mx = 1e5 + 9;
    if (!feasible(_s, _t, 0, mx)) return -1;
    return F.max_flow(_s, _t);
  }
  int min_flow(int _s, int _t) { //-1 means flow is not
      feasible
    int mx = 1e9;
    int ans = -1, l = 0, r = mx;
    while (l <= r) {
      int mid = l + r >> 1;
      if (feasible(_s, _t, 0, mid)) ans = mid, r = mid -
          1;
      else l = mid + 1;
    }
    return ans;
  }
};
```

4.4 Edmonds Karp

```
// O(V * E^2)
ll bfs(vector<vi> &adj, vector<vl> &capacity, int s, int
    t, vi& parent) {
        fill(parent.begin(), parent.end(), -1);
        parent[s] = -2;
        queue<pll> q;
        q.push({s, INFL});

        while (!q.empty()) {
                int cur = q.front().first;
                ll flow = q.front().second;
                q.pop();

                for (int next : adj[cur]) {
                        if (parent[next] == -1LL &&
                            capacity[cur][next]) {
                                parent[next] = cur;
                                ll new_flow = min(flow,
                                    capacity[cur][next]);
                                if (next == t)
                                        return new_flow;
                                q.push({next, new_flow});
                        }
                }
        }

        return 0;
}

ll maxflow(vector<vi> &adj, vector<vl> &capacity, int s,
    int t, int n) {
        ll flow = 0;
        vi parent(n);
        ll new_flow;

        while ((new_flow = bfs(adj, capacity, s, t,
            parent))) {
                flow += new_flow;
                int cur = t;
                while (cur != s) {
                        int prev = parent[cur];
                        capacity[prev][cur] -= new_flow;
                        capacity[cur][prev] += new_flow;
                        cur = prev;
                }
        }

        return flow;
}
```

4.5 Hopcroft Karp

```
// O(|E|*sqrt(|V|))
struct mbm {
  vector<vector<int>> g;
  vector<int> d, match;
  int nil, l, r;
  /// u -> 0 to l, v -> 0 to r
  mbm(int l, int r) : g(l+r), d(1+l+r, INF), match(l+r, l
      +r),
                      nil(l+r), l(l), r(r) {}
  void add_edge(int a, int b) {
    g[a].push_back(l+b);
    g[l+b].push_back(a);
  }
  bool bfs() {
    queue<int> q;
    for(int u = 0; u < l; u++) {
      if(match[u] == nil) {
        d[u] = 0;
```

```
        q.push(u);
      } else d[u] = INF;
    }
    d[nil] = INF;
    while(q.size()) {
      int u = q.front(); q.pop();
      if(u == nil) continue;
      for(auto v : g[u]) {
        if(d[ match[v] ] == INF) {
          d[ match[v] ] = d[u]+1;
          q.push(match[v]);
        }
      }
    }
    return d[nil] != INF;
  }
  bool dfs(int u) {
    if(u == nil) return true;
    for(int v : g[u]) {
      if(d[ match[v] ] == d[u]+1 && dfs(match[v])) {
        match[v] = u; match[u] = v;
        return true;
      }
    }
    d[u] = INF;
    return false;
  }
  int max_matching() {
    int ans = 0;
    while(bfs()) {
      for(int u = 0; u < l; u++) {
        ans += (match[u] == nil && dfs(u));
      }
    }
    return ans;
  }
  void matchs(){
    for (int i = 0; i<l; i++){
      if (match[i] == l+r) continue;
      cout << i+1 << ' ' << match[i]+1-l << ln;
    }
  }
};
```

4.6 Hungarian

```
#define rep(i, a, b) for(int i = a; i < (b); ++i)
typedef double type;
const type INF_TYPE = LLONG_MAX;
pair<type, vi> hungarian(const vector<vector<type>> &a) {
        if (a.empty()) return {0, {}};
        int n = sz(a) + 1, m = sz(a[0]) + 1;
        vector<type> u(n), v(m); vi p(m), ans(n - 1);
        rep(i,1,n) {
                p[0] = i;
                int j0 = 0; // add "dummy" worker 0
                vector<type> dist(m,INF_TYPE);vi pre(m,
                    -1);
                vector<bool> done(m + 1);
                do { // dijkstra
                        done[j0] = true;
                        int i0 = p[j0], j1;type delta =
                            INF_TYPE;
                        rep(j,1,m) if (!done[j]) {
                                auto cur = a[i0 - 1][j -
                                    1] - u[i0] - v[j];
                                if (cur < dist[j]) dist[j
                                    ] = cur, pre[j] = j0;
                                if (dist[j] < delta)
                                    delta = dist[j], j1 =
                                    j;
                        }
                        rep(j,0,m) {
                                if (done[j]) u[p[j]] +=
                                    delta, v[j] -= delta;
                                else dist[j] -= delta;
                        }
                        j0 = j1;
                } while (p[j0]);
                while (j0) { // update alternating path
                        int j1 = pre[j0];
                        p[j0] = p[j1], j0 = j1;
                }
        }
        rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
        return {-v[0], ans}; // min cost
}
```

4.7 Maximum Bipartite Matching

```
// O(|E|*|V|)
struct mbm {
        int l, r;
        vector<vector<int>> g;
        vector<int> match, seen;
        mbm(int l, int r) : l(l), r(r), g(l), match(r),
            seen(r){}
        void add_edge(int l, int r) { g[l].push_back(r);
            }
        bool dfs(int u) {
                for(auto v : g[u]) {
                        if(seen[v]++) continue;
                        if(match[v] == -1 || dfs(match[v
                            ])) {
                                match[v] = u;
                                return true;
                        }
```

```
                }
                return false;
        }
        int max_matching() {
                int ans = 0;
                fill(match.begin(), match.end(), -1);
                for(int u = 0; u < l; ++u) {
                        fill(seen.begin(), seen.end(), 0)
                            ;
                        ans += dfs(u);
                }
                return ans;
        }
        void matchs(){
                for (int i = 0; i<r; i++){
                        if (match[i] == -1) continue;
                        cout << match[i]+1  << ' ' << i+1
                             << ln;
                }
        }
};
```

4.8 Minimum Cost Maximum Flow

```
// O(|V|*|E|^2*log(|E|))
template <class type>
struct mcmf {
        struct edge { int u, v, cap, flow; type cost; };
        int n;
        vector<edge> ed;
        vector<vector<int>> g;
        vector<int> p;
        vector<type> d, phi;
        mcmf(int n) : n(n), g(n), p(n), d(n), phi(n) {}
        void add_edge(int u, int v, int cap, type cost) {
                g[u].push_back(ed.size());
                ed.push_back({u, v, cap, 0, cost});
                g[v].push_back(ed.size());
                ed.push_back({v, u, 0, 0, -cost});
        }
        bool dijkstra(int s, int t) {
                fill(d.begin(), d.end(), INF_TYPE);
                fill(p.begin(), p.end(), -1);
                set<pair<type, int>> q;
                d[s] = 0;
                for(q.insert({d[s], s}); q.size();) {
                        int u = (*q.begin()).second; q.
                            erase(q.begin());
                        for(auto v : g[u]) {
                                auto &e = ed[v];
                                type nd = d[e.u]+e.cost+
                                    phi[e.u]-phi[e.v];
                                if(0 < (e.cap-e.flow) &&
                                    nd < d[e.v]) {
                                        q.erase({d[e.v],
                                            e.v});
                                        d[e.v] = nd; p[e.
                                            v] = v;
                                        q.insert({d[e.v],
                                             e.v});
                                }
                        }
                }
                for(int i = 0; i < n; i++) phi[i] = min(
                    INF_TYPE, phi[i]+d[i]);
                return d[t] != INF_TYPE;
        }
        pair<int, type> max_flow(int s, int t) {
                type mc = 0;
                int mf = 0;
                fill(phi.begin(), phi.end(), 0);
                while(dijkstra(s, t)) {
                        int flow = INF;
                        for(int v = p[t]; v != -1; v = p[
                            ed[v].u ])
                                flow = min(flow, ed[v].
                                    cap-ed[v].flow);
                        for(int v = p[t]; v != -1; v = p[
                            ed[v].u ]) {
                                edge &e1 = ed[v];
                                edge &e2 = ed[v^1];
                                mc += e1.cost*flow;
                                e1.flow += flow;
                                e2.flow -= flow;
                        }
                        mf += flow;
                }
                return {mf, mc};
        }
};
```

4.9 MCMF Vasito

```
// O(|E|*|F|*log(|V|))
typedef int tf;
typedef int tc;
const tf INFFLOW=1e9;
const tc INFCOST=1e9;
struct MCF{
  int n;
  vector<tc> prio, pot; vector<tf> curflow; vector<int>
      prevedge,prevnode;
  priority_queue<pair<tc, int>, vector<pair<tc, int>>,
      greater<pair<tc, int>>> q;
  struct edge{int to, rev; tf f, cap; tc cost;};
  vector<vector<edge>> g;
  MCF(int n):n(n),prio(n),curflow(n),prevedge(n),prevnode
      (n),pot(n),g(n){}
```

```cpp
    void add_edge(int s, int t, tf cap, tc cost) {
      g[s].push_back((edge){t,sz(g[t]),0,cap,cost});
      g[t].push_back((edge){s,sz(g[s])-1,0,0,-cost});
    }
    pair<tf,tc> get_flow(int s, int t) {
      tf flow=0; tc flowcost=0;
      while(1){
        q.push({0, s});
        fill(all(prio),INFCOST);
        prio[s]=0; curflow[s]=INFFLOW;
        while(!q.empty()) {
          auto cur=q.top();
          tc d=cur.first;
          int u=cur.second;
          q.pop();
          if(d!=prio[u]) continue;
          for(int i=0; i<sz(g[u]); ++i) {
            edge &e=g[u][i];
            int v=e.to;
            if(e.cap<=e.f) continue;
            tc nprio=prio[u]+e.cost+pot[u]-pot[v];
            if(prio[v]>nprio) {
              prio[v]=nprio;
              q.push({nprio, v});
              prevnode[v]=u; prevedge[v]=i;
              curflow[v]=min(curflow[u], e.cap-e.f);
            }
          }
        }
        if(prio[t]==INFCOST) break;
        for(int i=0;i<n;i++) pot[i]+=prio[i];
        tf df=min(curflow[t], INFFLOW-flow);
        flow+=df;
        for(int v=t; v!=s; v=prevnode[v]) {
          edge &e=g[prevnode[v]][prevedge[v]];
          e.f+=df; g[v][e.rev].f-=df;
          flowcost+=df*e.cost;
        }
      }
      return {flow,flowcost};
    }
  };
```

4.10 L-R MCMF

```cpp
  //Works for both directed, undirected and with negative
     cost too
  //doesn't work for negative cycles
  //for undirected edges just make the directed flag false
  //Complexity: O(min(E^2 * V log V, E logV * flow))
  using T = long long;
  const T inf = 1LL << 61;
  struct MCMF {
      struct edge {
          int u, v;
          T cap, cost; int id;
          edge(int _u, int _v, T _cap, T _cost, int _id){
              u = _u; v = _v; cap = _cap; cost = _cost; id
                 = _id;
          }
      };
      int n, s, t, mxid; T flow, cost;
      vector<vector<int>> g; vector<edge> e;
      vector<T> d, potential, flow_through;
      vector<int> par; bool neg;
      MCMF() {}
      MCMF(int _n) { // 0-based indexing
          n = _n + 10;
          g.assign(n, vector<int> ());
          neg = false; mxid = 0;
      }
      void add_edge(int u, int v, T cap, T cost, int id =
          -1, bool directed = true) {
          if(cost < 0) neg = true;
          g[u].push_back(e.size());
          e.push_back(edge(u, v, cap, cost, id));
          g[v].push_back(e.size());
          e.push_back(edge(v, u, 0, -cost, -1));
          mxid = max(mxid, id);
          if(!directed) add_edge(v, u, cap, cost, -1, true)
             ;
      }
      bool dijkstra() {
          par.assign(n, -1);
          d.assign(n, inf);
          priority_queue<pair<T, T>, vector<pair<T, T>>,
             greater<pair<T, T>> > q;
          d[s] = 0;
          q.push(pair<T, T>(0, s));
          while (!q.empty()) {
              int u = q.top().second;
              T nw = q.top().first;
              q.pop();
              if(nw != d[u]) continue;
              for (int i = 0; i < (int)g[u].size(); i++) {
                  int id = g[u][i];
                  int v = e[id].v; T cap = e[id].cap;
                  T w = e[id].cost + potential[u] -
                     potential[v];
                  if (d[u] + w < d[v] && cap > 0) {
                      d[v] = d[u] + w;
                      par[v] = id;
                      q.push(pair<T, T>(d[v], v));
                  }
              }
          }
          for (int i = 0; i < n; i++) {
              if (d[i] < inf) d[i] += (potential[i] -
                 potential[s]);
```

```
        }
        for (int i = 0; i < n; i++) {
            if (d[i] < inf) potential[i] = d[i];
        }
        // return d[t] != inf; // for max flow min cost
        return d[t] <= 0; // for min cost flow
    }
    T send_flow(int v, T cur) {
        if(par[v] == -1) return cur;
        int id = par[v];
        int u = e[id].u; T w = e[id].cost;
        T f = send_flow(u, min(cur, e[id].cap));
        cost += f * w;
        e[id].cap -= f;
        e[id^1].cap += f;
        return f;
    }
    //returns {maxflow, mincost}
    pair<T, T> solve(int _s, int _t, T goal = inf) {
        s = _s; t = _t;
        flow = 0, cost = 0;
        potential.assign(n, 0);
        if (neg) {
            // run Bellman-Ford to find starting potential
            d.assign(n, inf);
            d[s] = 0;
            bool relax = true;
            for (int i = 0; i < n && relax; i++) {
                relax = false;
                for (int u = 0; u < n; u++) {
                    for (int k = 0; k < (int)g[u].size(); k++) {
                        int id = g[u][k];
                        int v = e[id].v; T cap = e[id].cap, w = e[id].cost;
                        if (d[v] > d[u] + w && cap > 0) {
                            d[v] = d[u] + w;
                            relax = true;
                        }
                    }
                }
            }
            for(int i = 0; i < n; i++) if(d[i] < inf)
                potential[i] = d[i];
        }
        while (flow < goal && dijkstra()) flow += send_flow(t, goal - flow);
        flow_through.assign(mxid + 10, 0);
        for (int u = 0; u < n; u++) {
            for (auto v: g[u]) {
                if (e[v].id >= 0) flow_through[e[v].id] = e[v ^ 1].cap;
            }
        }
            return make_pair(flow, cost);
        }
};
//flow_through[i] = extra flow beyond 'low' sent through edge i
//it finds the feasible solution with minimum cost
struct LR_Flow{
    MCMF F;
    static const T INF = 1e12;
    // sum of cost should be < INF / 2
    // flow * INF must not overflow in data type
    int n;
    T target;
    LR_Flow() {}
    LR_Flow(int _n) {
        n = _n + 10; target = 0;
        F = MCMF(n);
    }
    void add_edge(int u, int v, T l, T r, T cost = 0, int id = -1, bool directed = true) {
        assert(0 <= l && l <= r);
        target += l;
        F.add_edge(u, v, l, -INF + cost, -1, directed);
            // will try to take this edge first
        F.add_edge(u, v, r - l, cost, id, directed);
    }
    pair<T, T> solve(int s, int t, T goal = inf) {
        auto ans = F.solve(s, t, goal);
        ans.second += INF * target;
        if (abs(ans.second) >= INF / 2) return {-1, -1};
            // not feasible
        return ans;
    }
};
```

4.11 Scaling Algorithm

```
// O(|E|^2*log(C)) C = maximum edge weight of the graph
struct MaxFlow {
    static const ll INF = 1e18;
    struct Edge {int u,v;ll w;};
    int n, s, t;
    vector<vector<int>> g;
    vector<Edge> ed;
    vector<bool> vis;
    ll flow = 0;
    MaxFlow(int n, int s, int t) : n(n), s(s), t(t), g(n) {}
    int add_edge(int u, int v, ll forward, ll backward = 0) {
        const int id = (int)ed.size();
        g[u].emplace_back(id);
        ed.push_back({u, v, forward});
```

```
        g[v].emplace_back(id + 1);
        ed.push_back({v, u, backward});
        return id;
    }
    bool dfs(int node, ll lim) {
        if (node == t) return true;
        if (vis[node]) return false;
        vis[node] = true;
        for (int i : g[node]) {
            auto &e = ed[i];
            auto &back = ed[i ^ 1];
            if (e.w >= lim) {
                if (dfs(e.v, lim)) {
                    e.w -= lim;
                    back.w += lim;
                    return true;
                }
            }
        }
        return false;
    }
    ll max_flow() {
        for (ll bit = 1ll << 62; bit > 0; bit /= 2) {
            bool found = false;
            do {
                vis.assign(n, false);
                found = dfs(s, bit);
                flow += bit * found;
            } while (found);
        }
        return flow;
    }
};
```

4.12 Weighted Matching

```
// O(|V|^3)
typedef int type;
struct matching_weighted {
  int l, r;
  vector<vector<type>> c;
  matching_weighted(int l, int r) : l(l), r(r), c(l,
      vector<type>(r)) {
    assert(l <= r);
  }
  void add_edge(int a, int b, type cost) { c[a][b] = cost
      ; }
  type matching() {
    vector<type> v(r), d(r); // v: potential
    vector<int> ml(l, -1), mr(r, -1); // matching pairs
    vector<int> idx(r), prev(r);
    iota(idx.begin(), idx.end(), 0);
    auto residue = [&](int i, int j) { return c[i][j]-v[j
        ]; };
    for(int f = 0; f < l; ++f) {
      for(int j = 0; j < r; ++j) {
        d[j] = residue(f, j);
        prev[j] = f;
      }
      type w;
      int j, l;
      for (int s = 0, t = 0;;) {
        if(s == t) {
          l = s;
          w = d[ idx[t++] ];
          for(int k = t; k < r; ++k) {
            j = idx[k];
            type h = d[j];
            if (h <= w) {
              if (h < w) t = s, w = h;
              idx[k] = idx[t];
              idx[t++] = j;
            }
          }
          for (int k = s; k < t; ++k) {
            j = idx[k];
            if (mr[j] < 0) goto aug;
          }
        }
        int q = idx[s++], i = mr[q];
        for (int k = t; k < r; ++k) {
          j = idx[k];
          type h = residue(i, j) - residue(i, q) + w;
          if (h < d[j]) {
            d[j] = h;
            prev[j] = i;
            if(h == w) {
              if(mr[j] < 0) goto aug;
              idx[k] = idx[t];
              idx[t++] = j;
            }
          }
        }
      }
      aug: for (int k = 0; k < l; ++k)
        v[ idx[k] ] += d[ idx[k] ] - w;
      int i;
      do {
        mr[j] = i = prev[j];
        swap(j, ml[i]);
      } while (i != f);
    }
    type opt = 0;
    for (int i = 0; i < l; ++i)
      opt += c[i][ml[i]]; // (i, ml[i]) is a solution
    return opt;
  }
};
```

5 Geometria

5.1 2D Tree

```
// given a set of points, answer queries of nearest point
    in O(log(n))
bool onx(pt a, pt b){return a.x < b.x;}
bool ony(pt a, pt b){return a.y < b.y;}
struct Node {
        pt pp;
        lf x0 = inf, x1 = -inf, y0 = inf, y1 = -inf;
        Node *first = 0, *second = 0;

        ll distance(pt p){
                ll x = min(max(x0, p.x), x1);
                ll y = min(max(y0, p.y), y1);
                return norm2(pt(x, y) - p);
        }

        Node(vector<pt>&& vp) : pp(vp[0]){
                for(pt p : vp){
                        x0 = min(x0, p.x);
            x1 = max(x1, p.x);
                        y0 = min(y0, p.y);
            y1 = max(y1, p.y);
                }
                if(vp.size() > 1){
                        sort(all(vp), x1 - x0 >= y1 - y0
                            ? onx : ony);
                        int m = vp.size() / 2;
                        first = new Node({vp.begin(), vp.
                            begin() + m});
                        second = new Node({vp.begin() + m
                            , vp.end()});
                }
        }
};

struct KDTree {
        Node* root;
        KDTree(const vector<pt>& vp): root(new Node({all(
            vp)})){}
        pair<ll, pt> search(pt p, Node *node){
                if(!node->first){
                        // avoid query point as answer
                        // if(p.x == node->pp.x && p.y ==
                            node->pp.y) return {inf, pt()
                            };
                        return {norm2(p-node->pp), node->
                            pp};
                }
                Node *f = node->first, *s = node->second;
                ll bf = f->distance(p), bs = s ->
                    distance(p);
                if(bf > bs) swap(bf, bs), swap(f, s);
                auto best = search(p, f);
                if(bs < best.ff) best = min(best, search(
                    p, s));
                return best;
        }
        pair<ll, pt> nearest(pt p){ return search(p, root
            ); }
};
```

5.2 3D

```
typedef double lf;
struct p3 {
    lf x, y, z;
        p3(){}
        p3(lf x, lf y, lf z): x(x), y(y), z(z){}
    p3 operator + (p3 p){ return {x + p.x, y + p.y, z + p
        .z}; }
    p3 operator - (p3 p){ return {x - p.x, y - p.y, z - p
        .z}; }
    p3 operator * (lf d){ return {x * d, y * d, z * d}; }
    p3 operator / (lf d){ return {x / d, y / d, z / d}; }
        // only for floating point
    // Some comparators
    bool operator == (p3 p){ return tie(x, y, z) == tie(p
        .x, p.y, p.z); }
    bool operator != (p3 p){ return !operator == (p); }
        void print(){ cout << x << " " << y << " " << z
            << "\n"; }
        // scale: (newnorm / norm) * p3
};

lf dot(p3 v, p3 w){ return v.x * w.x + v.y * w.y + v.z *
    w.z; }

p3 cross(p3 v, p3 w){
    return { v.y * w.z - v.z * w.y, v.z * w.x - v.x * w.z
        , v.x * w.y - v.y * w.x };
}

lf norm2(p3 v){ return dot(v, v); }
lf norm(p3 v){ return sqrt(norm2(v)); }
p3 unit(p3 v){ return v / norm(v); }

// ang(RAD)
double angle(p3 v, p3 w){
    double cos_theta = dot(v, w) / norm(v) / norm(w);
    return acos(max(-1.0, min(1.0, cos_theta)));
}

// orient s, pqr form a triangle pos: 'up', zero = on,
    neg = 'dow'
lf orient(p3 p, p3 q, p3 r, p3 s){
        return dot(cross((q - p), (r - p)), (s - p));
}
```

```
// same as 2D but in n-normal direction
lf orient_by_normal(p3 p, p3 q, p3 r, p3 n){
        return dot(cross((q - p), (r - p)), n);
}

struct plane {
    p3 n; lf d; // n: normal d: dist to zero
    // From normal n and offset d
    plane(p3 n, lf d): n(n), d(d) {}
    // From normal n and point P
    plane(p3 n, p3 p): n(n), d(dot(n, p)) {}
    // From three non-collinear points P,Q,R
    plane(p3 p, p3 q, p3 r): plane(cross((q - p), (r - p)
        ), p){}
    // - these work with lf = int
    lf side(p3 p) { return dot(n, p) - d; }
    double dist(p3 p) { return abs(side(p)) / norm(n); }
    plane translate(p3 t) {return {n, d + dot(n, t)}; }
    /// - these require lf = double
    plane shift_up(double dist){ return {n, d + dist *
        norm(n)}; }
    p3 proj(p3 p){ return p - n * side(p) / norm2(n); }
    p3 refl(p3 p){ return p - n * 2 * side(p) / norm2(n);
        }
};

struct line3d {
        p3 d, o; // d: dir o: point on line
        // From two points P, Q
        line3d(p3 p, p3 q): d(q - p), o(p){}
        // From two planes p1, p2 (requires lf = double)
        line3d(plane p1, plane p2){
                d = cross(p1.n, p2.n);
                o = cross((p2.n * p1.d - p1.n * p2.d), d)
                    / norm2(d);
        }
        // - these work with lf = int
        double dist2(p3 p){ return norm2(cross(d, (p - o)
            )) / norm2(d); }
        double dist(p3 p){ return sqrt(dist2(p)); }
        bool cmp_proj(p3 p, p3 q){ return dot(d, p) < dot
            (d, q); }
        // - these require lf = double
        p3 proj(p3 p){ return o + d * dot(d, (p - o)) /
            norm2(d); }
        p3 refl(p3 p){ return proj(p) * 2 - p; }
        p3 inter(plane p){ return o - d * p.side(o) / dot
            (p.n, d); }
        // get other point: pl.o + pl.d * t;
};

double dist(line3d l1, line3d l2) {
        p3 n = cross(l1.d, l2.d);
        if(n == p3(0, 0, 0)) return l1.dist(l2.o); //
            parallel
        return abs(dot((l2.o - l1.o), n)) / norm(n);
}

// closest point on l1 to l2
p3 closest_on_line1(line3d l1, line3d l2) {
        p3 n2 = cross(l2.d, cross(l1.d, l2.d));
        return l1.o + l1.d * (dot((l2.o - l1.o), n2)) /
            dot(l1.d, n2);
}

double small_angle(p3 v, p3 w){ return acos(min(abs(dot(v
    , w)) / norm(v) / norm(w), 1.0)); } // 0 90
double angle(plane p1, plane p2){ return small_angle(p1.n
    , p2.n); }
bool is_parallel(plane p1, plane p2){ return cross(p1.n,
    p2.n) == p3(0, 0, 0); }
bool is_perpendicular(plane p1, plane p2){ return dot(p1.
    n, p2.n) == 0; }
double angle(line3d l1, line3d l2){ return small_angle(l1
    .d, l2.d); }
bool is_parallel(line3d l1, line3d l2){ return cross(l1.d
    , l2.d) == p3(0, 0, 0); }
bool is_perpendicular(line3d l1, line3d l2){ return dot(
    l1.d, l2.d) == 0; }
double angle(plane p, line3d l){ return M_PI / 2 -
    small_angle(p.n, l.d); }
bool is_parallel(plane p, line3d l){ return dot(p.n, l.d)
     == 0; }
bool is_perpendicular(plane p, line3d l){ return cross(p.
    n, l.d) == p3(0, 0, 0); }
line3d perp_through(plane p, p3 o){ return line3d(o, o +
    p.n); }
plane perp_through(line3d l, p3 o){ return plane(l.d, o);
     }

pair<p3, lf> smallest_enclosing_sphere(vector<p3> p) {
    int n = p.size();
    p3 c(0, 0, 0);
    for(int i = 0; i < n; i++) c = c + p[i];
    c = c / n;

    double ratio = 0.1;
    int pos = 0;
    int it = 100000;
    while (it--) {
        pos = 0;
        for (int i = 1; i < n; i++) {
            if(norm2(c - p[i]) > norm2(c - p[pos])) pos =
                 i;
        }
        c = c + (p[pos] - c) * ratio;
        ratio *= 0.998;
    }
    return {c, sqrt(norm2(c - p[pos]))};
}
```

5.3 Circulos

```cpp
// add Lines Points

enum {OUT, IN, ON};

struct circle {
	pt center; lf r;
	// (x - xo)^2 + (y - yo)^2 = r^2

	circle(pt c, lf r): center(c), r(r){};

	// circle that passes through abc
	circle(pt a, pt b, pt c) {
		b = b - a, c = c - a;
		assert(cross(b, c) != 0); // no circumcircle if A, B, C aligned
		pt cen = a + rot90(b * norm2(c) - c * norm2(b)) / cross(b, c) / 2;
		center = cen;
		r = norm(a - cen);
	}

	// diameter = segment pq
	circle(pt p, pt q) {
		center = (p + q) * 0.5L;
		r = dis(p, q) * 0.5L;
	}

	int contains(pt &p) {
		lf det = r * r - dis2(center, p);
		if(fabsl(det) <= EPS) return ON;
		return (det > EPS ? IN : OUT);
	}

	bool in(circle c){ return norm(center - c.center) + r <= c.r + EPS; }  // non strict

};

// centers of the circles that pass through ab and has radius r
vector<pt> centers(pt a, pt b, lf r) {
	if (norm(a - b) > 2 * r + EPS) return {};
	pt m = (a + b) / 2;
	double f = sqrt(r * r / norm2(a - m) - 1);
	pt c = rot90(a - m) * f;
	return {m - c, m + c};
}

vector<pt> inter_cl(circle c, line l){
	vector<pt> s;
	pt p = l.proj(c.center);
	lf d = norm(p - c.center);
	if(d - EPS > c.r) return s;
	if(abs(d - c.r) <= EPS){ s.push_back(p); return s ; }
	d=sqrt(c.r * c.r - d * d);
	s.push_back(p + normalize(l.v) * d);
	s.push_back(p - normalize(l.v) * d);
	return s;
}

vector<pt> inter_cc(circle c1, circle c2) {
	pt dir = c2.center - c1.center;
	lf d2 = dis2(c1.center, c2.center);

	if(d2 <= E0) {
		//assert( fabsl( c1.r - c2.r ) > E0 );
		return {};
	}

	lf td = 0.5L * ( d2 + c1.r * c1.r - c2.r * c2.r ) ;

	lf h2 = c1.r * c1.r - td / d2 * td;

	pt p = c1.center + dir * (td / d2);
	if(fabsl( h2 ) < EPS) return {p};
	if(h2 < 0.0L) return {};

	pt dir_h = rot90(dir) * sqrtl(h2 / d2);

	return {p + dir_h, p - dir_h};
}

//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<pt> circle_line_intersection(pt c, lf r, pt a, pt b) {
    vector<pt> ret;
    b = b - a; a = a - c;
    lf A = dot(b, b), B = dot(a, b);
    lf C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -EPS) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + EPS)) / A);
    if (D > EPS) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
    return ret;
}

// circle-line inter = 1, inner: 1 = oxo 0 = o=o
vector<pair<pt, pt>> tangents(circle c1, circle c2, bool inner){
	vector<pair<pt, pt>> out;
	if (inner) c2.r = -c2.r; // inner tangent
	pt d = c2.center - c1.center;
	double dr = c1.r - c2.r, d2 = norm2(d), h2 = d2 - dr * dr;
	if (d2 == 0 || h2 < 0) { assert(h2 != 0); return {}; } // (identical)
	for (double s : {-1, 1}) {
		pt v = (d * dr + rot90(d) * sqrt(h2) * s) / d2;
		out.push_back({c1.center + v * c1.r, c2.center + v * c2.r});
	}
```

```cpp
        return out; // if size 1: circle are tangent
}

// circle targent passing through pt p
pair<pt, pt> tangent_through_pt(circle c, pt p){
        pair<pt, pt> out;
        double d = norm2(p - c.center);
        if (d < c.r) return {};
        pt base = c.center - p;
        double w = sqrt(norm2(base) - c.r * c.r);
        pt a = {w, c.r}, b = {w, -c.r};
        pt s = p + base * a / norm2(base) * w;
        pt t = p + base * b / norm2(base) * w;
        out = {s, t};
        return out;
}

lf safeAcos(lf x) {
        if (x < -1.0) x = -1.0;
        if (x > 1.0) x = 1.0;
        return acos(x);
}

lf areaOfIntersectionOfTwoCircles(circle c1, circle c2){
        lf r1 = c1.r, r2 = c2.r, d = dis(c1.center, c2.center);
        if(d >= r1 + r2) return 0.0L;
        if(d <= fabsl(r2 - r1)) return PI * (r1 < r2 ? r1 * r1 : r2 * r2);
        lf alpha = safeAcos((r1 * r1 - r2 * r2 + d * d) / (2.0L * d * r1));
        lf betha = safeAcos((r2 * r2 - r1 * r1 + d * d) / (2.0L * d * r2));
        lf a1 = r1 * r1 * (alpha - sinl(alpha) * cosl(alpha));
        lf a2 = r2 * r2 * (betha - sinl(betha) * cosl(betha));
        return a1 + a2;
};

lf intertriangle(circle& c, pt a, pt b){ // area of intersection with oab
        if(abs(cross((c.center - a), (c.center - b))) <= EPS) return 0.;
        vector<pt> q = {a}, w = inter_cl(c, line(a, b));
        if(w.size() == 2) for(auto p: w) if(dot((a - p), (b - p)) < -EPS) q.push_back(p);
        q.push_back(b);
        if(q.size() == 4 && dot((q[0] - q[1]), (q[2] - q[1])) > EPS) swap(q[1], q[2]);
        lf s = 0;
        for(int i = 0; i < q.size() - 1; ++i){
                if(!c.contains(q[i]) || !c.contains(q[i + 1])) s += c.r * c.r * min_angle((q[i] - c.center), q[i+1] - c.center) / 2;
                else s += abs(cross((q[i] - c.center), (q[i + 1] - c.center)) / 2);
        }
        return s;
}

bool circumcircle_contains(vector<pt> tr, pt D) { // triange CCW
  pt A = tr[0] - D, B = tr[1] - D, C = tr[2] - D;

  lf norm_a = norm2(tr[0]) - norm2(D);
  lf norm_b = norm2(tr[1]) - norm2(D);
  lf norm_c = norm2(tr[2]) - norm2(D);

  lf det1 = A.x * (B.y * norm_c - norm_b * C.y);
  lf det2 = B.x * (C.y * norm_a - norm_c * A.y);
  lf det3 = C.x * (A.y * norm_b - norm_a * B.y);

  return det1 + det2 + det3 > E0;
}

// r[k]: area covered by at least k circles
// O(n^2 log n) (high constant)
vector<lf> intercircles(vector<circle> c){
        vector<lf> r(c.size() + 1);
        for(int i = 0; i < c.size(); ++i){
                int k = 1;  pt O = c[i].center;
                vector<pair<pt, int>> p = {
                        {c[i].center + pt(1,0) * c[i].r, 0},
                        {c[i].center - pt(1,0) * c[i].r, 0}};

                for(int j = 0; j < c.size(); ++j) if(j != i){
                        bool b0 = c[i].in(c[j]), b1 = c[j].in(c[i]);
                        if(b0 && (!b1 || i < j)) ++k;
                        else if(!b0 && !b1){
                                auto v = inter_cc(c[i], c[j]);
                                if(v.size() == 2){
                                        swap(v[0], v[1]);
                                        p.push_back({v[0], 1});
                                        p.push_back({v[1], -1});
                                        if(polar_cmp(v[1] - O, v[0] - O)) ++k;
                                }
                        }
                }

                sort(all(p), [&](auto& a, auto& b){
                   return polar_cmp(a.first - O, b.first - O); });
                for(int j = 0; j < p.size(); ++j){
```

```
                        pt p0 = p[j ? j - 1 : p.size()
                            -1].first, p1 = p[j].first;
                        lf a = min_angle((p0 - c[i].
                            center), (p1 - c[i].center));
                        r[k] += (p0.x - p1.x) * (p0.y +
                            p1.y) / 2 + c[i].r * c[i].r *
                            (a - sin(a)) / 2;
                        k += p[j].second;
                }
        }
        return r;
}
```

5.4 Closest Points

```
// O(nlogn)
pair<pt, pt> closest_points(vector<pt> v){
        sort(v.begin(), v.end());
        pair<pt, pt> ans;
        lf d2 = INF;

        function<void( int, int )> solve = [&](int l, int
            r) {
                if(l == r) return;

                int mid = (l + r) / 2;
                lf x_mid = v[mid].x;
                solve(l, mid);
                solve(mid + 1, r);

                vector<pt> aux;
                int p1 = l, p2 = mid + 1;
                while (p1 <= mid && p2 <= r) {
                        if(v[p1].y < v[p2].y) aux.
                            push_back(v[p1++]);
                        else aux.push_back(v[p2++]);
                }
                while(p1 <= mid) aux.push_back(v[p1++]);
                while(p2 <= r) aux.push_back(v[p2++]);

                vector<pt> nb;
                for(int i = l; i <= r; ++i){
                v[i] = aux[i - l];
                lf dx = (x_mid - v[i].x);
                if(dx * dx < d2)
                        nb.push_back(v[i]);
                }

                for(int i = 0; i < (int) nb.size(); ++i){
                for(int k = i + 1; k < (int) nb.size();
                    ++k){
                        lf dy = (nb[k].y - nb[i].y);
                        if(dy * dy > d2) break;
                        lf nd2 = dis2(nb[i], nb[k]);
                        if(nd2 < d2) d2 = nd2, ans = {nb[
                            i], nb[k]};
                }
                }
        };
        solve(0, v.size() -1);
        return ans;
}
```

5.5 Convex Hull

```
// CCW order
// if colineal are needed, use > in orient and remove
    repeated points
vector<pt> chull(vector<pt>& p){
        if(p.size() < 3) return p;

        vector<pt> r; //r.reserve(p.size());
        sort(p.begin(), p.end()); // first x, then y

        for(int i = 0; i < p.size(); i++){ // lower hull
                while(r.size() >= 2 && orient(r[r.size()
                    - 2], p[i], r.back()) >= 0) r.pop_back
                    ();
                r.pb(p[i]);
        }
        r.pop_back();

        int k = r.size();
        for(int i = p.size() - 1; i >= 0; --i){ // upper
            hull
                while(r.size() >= k + 2 && orient(r[r.
                    size() - 2], p[i], r.back()) >= 0) r.
                    pop_back();
                r.pb(p[i]);
        }
        r.pop_back();

        return r;
}
```

5.6 Dynamic Convex Hull

```
bool ccw(pt p, pt q, pt r) { // counter-clockwise
        return cross((q - p), (r - q)) > 0; // >= if
            colineal are needed
}

struct upper {
        set<pt> se;
        set<pt>::iterator it;

        int is_under(pt p) { // 1 -> inside ; 2 -> border
                it = se.lower_bound(p);
                if (it == se.end()) return 0;
```

```
            if (it == se.begin()) return p == *it ? 2
                : 0;
            if (ccw(p, *it, *prev(it))) return 1;
            return ccw(p, *prev(it), *it) ? 0 : 2;
        }
        void insert(pt p) {
            if (is_under(p)) return;

            if (it != se.end()) while (next(it) != se
                .end() and !ccw(*next(it), *it, p))
                    it = se.erase(it);
            if (it != se.begin()) while (--it != se.
                begin() and !ccw(p, *it, *prev(it)))
                    it = se.erase(it);

            se.insert(p);
        }
};

struct dyn_hull { // 1 -> inside ; 2 -> border
        upper U, L;

        int is_inside(pt p) {
            int u = U.is_under(p), l = L.is_under({-p
                .x, -p.y});
            if (!u or !l) return 0;
            return max(u, l);
        }
        void insert(pt p) {
            U.insert(p);
            L.insert({-p.x, -p.y});
        }
        int size() {
            int ans = U.se.size() + L.se.size();
            return ans <= 2 ? ans/2 : ans-2;
        }
};

// farthest_dynamic

vector<vector<pt>> pols;

// log^2(n) amortized (binary mergin)
void add(pt pi){
    vector<pt> pa = {pi};
        while(sz(pols) && sz(pols.back()) < 2 * sz(pa)){
        for(pt pi : pols.back()) pa.pb(pi);
            pols.pop_back();
        }
        vector<pt> ch = chull(pa);
        pols.pb(ch);
}

// log^2(n)
lf query(pt &dir){
    lf maxi = LONG_LONG_MIN;
        for(vector<pt> &pol : pols){
            lf ans = dot(farthest(pol, dir), dir);
            maxi = max(maxi, ans);
        }
        return maxi;
}
```

5.7 Delaunay

```
// Returns planar graph representing Delaunay's
    triangulation.
// Edges for each vertex are in ccw order.
// Voronoi vertices = the circumcenters of the Delaunay
    triangles.
// O(nlogn)
typedef struct QuadEdge* Q;
struct QuadEdge {
        int id,used;
        pt o;
        Q rot,nxt;
        QuadEdge(int id=-1, pt o=pt(INF,INF)):id(id),used
            (0),o(o),rot(0),nxt(0){}
        Q rev(){return rot->rot;}
        Q next(){return nxt;}
        Q prev(){return rot->next()->rot;}
        pt dest(){return rev()->o;}
};

Q edge(pt a, pt b, int ida, int idb){
        Q e1=new QuadEdge(ida,a);
        Q e2=new QuadEdge(idb,b);
        Q e3=new QuadEdge;
        Q e4=new QuadEdge;
        tie(e1->rot,e2->rot,e3->rot,e4->rot)={e3,e4,e2,e1
            };
        tie(e1->nxt,e2->nxt,e3->nxt,e4->nxt)={e1,e2,e4,e3
            };
        return e1;
}

void splice(Q a, Q b){
        swap(a->nxt->rot->nxt,b->nxt->rot->nxt);
        swap(a->nxt,b->nxt);
}

void del_edge(Q& e, Q ne){
        splice(e,e->prev()); splice(e->rev(),e->rev()->
            prev());
        delete e->rev()->rot; delete e->rev();
        delete e->rot; delete e;
        e=ne;
}

Q conn(Q a, Q b){
        Q e=edge(a->dest(),b->o,a->rev()->id,b->id);
```

```
		splice(e,a->rev()->prev());
		splice(e->rev(),b);
		return e;
}

auto area(pt p, pt q, pt r){ return cross((q-p),(r-q)); }

bool circumcircle_contains(vector<pt> tr, pt D){
		if (orient(tr[0], tr[1], tr[2]) < 0) reverse(all(
			tr));

	pt A = tr[0] - D, B = tr[1] - D, C = tr[2] - D;

	lf norm_a = norm2(tr[0]) - norm2(D);
	lf norm_b = norm2(tr[1]) - norm2(D);
	lf norm_c = norm2(tr[2]) - norm2(D);

	lf det1 = A.x * (B.y * norm_c - norm_b * C.y);
	lf det2 = B.x * (C.y * norm_a - norm_c * A.y);
	lf det3 = C.x * (A.y * norm_b - norm_a * B.y);

	return det1 + det2 + det3 > 0;
}

pair<Q,Q> build_tr(vector<pt>& p, int l, int r){
		if(r-l+1<=3){
				Q a=edge(p[l],p[l+1],l,l+1),b=edge(p[l
					+1],p[r],l+1,r);
				if(r-l+1==2) return {a,a->rev()};
				splice(a->rev(),b);
				auto ar=area(p[l],p[l+1],p[r]);
				Q c=abs(ar)>EPS?conn(b,a):0;
				if(ar>=-EPS) return {a,b->rev()};
				return {c->rev(),c};
		}
		int m=(l+r)/2;
		auto [la,ra]=build_tr(p,l,m);
		auto [lb,rb]=build_tr(p,m+1,r);
		while(1){
				if(orient(lb->o,ra->o, ra->dest()) > 0)
					ra=ra->rev()->prev();
				else if(orient(lb->o,ra->o,lb->dest()) >
					0) lb=lb->rev()->next();
				else break;
		}
		Q b=conn(lb->rev(),ra);
		auto valid=[&](Q e){return orient(e->dest(),b->
			dest(),b->o) > 0;};
		if(ra->o==la->o) la=b->rev();
		if(lb->o==rb->o) rb=b;
		while(1){
				Q L=b->rev()->next();
				if(valid(L)) while(circumcircle_contains
					({b->dest(),b->o,L->dest()},L->next()
					->dest())) del_edge(L,L->next());
				Q R=b->prev();
				if(valid(R)) while(circumcircle_contains
					({b->dest(),b->o,R->dest()},R->prev()
					->dest())) del_edge(R,R->prev());
				if(!valid(L)&&!valid(R)) break;
				if(!valid(L)||(valid(R)&&
					circumcircle_contains({L->dest(),L->o,
					R->o},R->dest()))) b=conn(R,b->rev());
				else b=conn(b->rev(),L->rev());
		}
		return {la,rb};
}

vector<vector<int>> delaunay(vector<pt> v){
		int n=v.size(); auto tmp=v;
		vector<int> id(n); iota(all(id),0);
		sort(all(id),[&](int l, int r){return v[l]<v[r
			];});
		for(int i = 0; i < n; ++i) v[i]=tmp[id[i]];
		assert(unique(all(v))==v.end());
		vector<vector<int>> g(n);
		int col=1;
		for(int i = 2; i < n; ++i) col &= abs(area(v[i],v
			[i-1],v[i-2])) <= EPS;
		if(col){
				for(int i = 1; i < n; i++) g[id[i-1]].pb(
					id[i]),g[id[i]].pb(id[i-1]);
		}
		else{
				Q e=build_tr(v,0,n-1).first;
				vector<Q> edg={e};
				for(int i=0;i<edg.size();e=edg[i++]){
						for(Q at=e;!at->used;at=at->next
							()){
								at->used=1;
								g[id[at->id]].pb(id[at->
									rev()->id]);
								edg.pb(at->rev());
						}
				}
		}
		return g;
}
```

5.8 Halfplanes

```
const lf INF = 1e100;
struct Halfplane {
		pt p, pq; // p: point on line, pq: dir, take left
		lf angle;
		Halfplane(){}
		Halfplane(pt& a, pt& b): p(a), pq(b - a){
				angle = atan2l(pq.y, pq.x);
		}

		bool out(const pt& r){ return cross(pq, r - p) <
```

```
        -EPS;} // checks if p is inside the half plane
    bool operator < (const Halfplane& e) const {
        return angle < e.angle; }

};

// intersection pt of the lines of 2 halfplanes
pt inter(const Halfplane& s, const Halfplane& t){
        if (abs(cross(s.pq, t.pq)) <= EPS) return {INF,
           INF};
        lf alpha = cross((t.p - s.p), t.pq) / cross(s.pq,
            t.pq);
        return s.p + (s.pq * alpha);
}

// O(nlogn) return CCW polygon
vector<pt> hp_intersect(vector<Halfplane>& H) {
        pt box[4] = {pt(INF, INF), pt(-INF, INF), pt(-INF
           , -INF), pt(INF, -INF)};

        for(int i = 0; i < 4; ++i ) {
                Halfplane aux(box[i], box[(i + 1) % 4]);
                H.push_back(aux);
        }

        sort(H.begin(), H.end());
        deque<Halfplane> dq;
        int len = 0;
        for(int i = 0; i < int(H.size()); ++i){

                while (len > 1 && H[i].out(inter(dq[len -
                    1], dq[len - 2]))){
                        dq.pop_back();
                        --len;
                }

                while (len > 1 && H[i].out(inter(dq[0],
                   dq[1]))){
                        dq.pop_front();
                        --len;
                }

                if (len > 0 && fabsl(cross(H[i].pq, dq[
                   len - 1].pq)) < EPS){
                        if (dot(H[i].pq, dq[len - 1].pq)
                           < 0.0) return vector<pt>();

                        if (H[i].out(dq[len - 1].p)){
                                dq.pop_back();
                                --len;
                        } else continue;
                }

                dq.push_back(H[i]);
                ++len;
        }

        while (len > 2 && dq[0].out(inter(dq[len - 1], dq
           [len - 2]))){
                dq.pop_back();
                --len;
        }

        while (len > 2 && dq[len - 1].out(inter(dq[0], dq
           [1]))){
                dq.pop_front();
                --len;
        }

        if (len < 3) return vector<pt>();

        vector<pt> ret(len);
        for(int i = 0; i + 1 < len; ++i) ret[i] = inter(
           dq[i], dq[i + 1]);

        ret.back() = inter(dq[len - 1], dq[0]);
        // remove repeated points if needed
        return ret;
}

// -------------------------------------------------------

// intersection of halfplanes
vector<pt> hp_intersect(vector<halfplane>& b){
        vector<pt> box = {{inf, inf}, {-inf, inf}, {-inf,
            -inf}, {inf, -inf}};
        for(int i = 0; i < 4; i++){
                b.push_back({box[i], box[(i + 1) % 4]});
        }
        sort(b.begin(), b.end());
        int n = b.size(), q = 1, h = 0;
        vector<halfplane> c(n + 10);
        for(int i = 0; i < n; i++){
                while(q < h && b[i].out(inter(c[h], c[h -
                    1]))) h--;
                while(q < h && b[i].out(inter(c[q], c[q +
                    1]))) q++;
                c[++h] = b[i];
                if(q < h && abs(cross(c[h].pq, c[h-1].pq)
                   ) < EPS){
                        if(dot(c[h].pq, c[h - 1].pq) <=
                           0) return {};
                        h--;
                        if(b[i].out(c[h].p)) c[h] = b[i];
                }
        }
        while(q < h - 1 && c[q].out(inter(c[h], c[h - 1])
           )) h--;
        while(q < h - 1 && c[h].out(inter(c[q], c[q + 1])
           )) q++;
        if(h - q <= 1) return {};
        c[h + 1] = c[q];
        vector<pt> s;
        for(int i = q; i < h + 1; i++) s.pb(inter(c[i], c
           [i + 1]));
        return s;
```

```
}
```

5.9 KD Tree

```
const ll INF = 2e18;
const int D = 2; // dimension

struct ptd{
        int p[D];
        bool operator !=(const ptd &a) const {
                bool ok = 1;
                for(int i = 0; i < D; i++) ok &= (p[i] == a.p[i]);
                return !ok;
        }
};

struct kd_node{
        ptd p;
        int axis;
        kd_node *left, *right;
};

struct cmp_points {
        int axis;
        cmp_points(){}
        cmp_points(int x): axis(x){}

        bool operator ()(const ptd &a, const ptd &b) const {
                return a.p[axis] < b.p[axis];
        }
};

ll dis2(ptd a, ptd b) {
        ll ans = 0;
        for(int i = 0; i < D; i++) ans += (a.p[i] - b.p[i]) * 1ll * (a.p[i] - b.p[i]);
        return ans;
}

struct KDTree{
        vector<ptd> arr;
        kd_node* root;

        KDTree(vector<ptd> &vptd): arr(vptd){
                build(root, 0, sz(vptd) - 1);
        }

    // O(nlogn)
        void build(kd_node* &node, int l, int r){
                if(l > r) {
                        node = nullptr;
                        return;
                }

                node = new kd_node();

                if(l == r) {
                        node->p = arr[l];
                        node->left = nullptr;
                        node->right = nullptr;
                        return;
                }

                ll bAxis = 0;
                ll mRange = 0;
                for (int axis = 0; axis < D; ++axis) {
                        ll minVal = INF, maxVal = -INF;
                        for (int i = l; i <= r; ++i){
                                minVal = min(minVal, (ll)arr[i].p[axis]);
                                maxVal = max(maxVal, (ll)arr[i].p[axis]);
                        }

                        if (maxVal - minVal > mRange) {
                                mRange = maxVal - minVal;
                                bAxis = axis;
                        }
                }

                int mid = (l + r) / 2;

                nth_element(arr.begin() + l, arr.begin() + mid, arr.begin() + r + 1, cmp_points(bAxis));
                node->p = arr[mid];
                node->axis = bAxis;
                build(node->left, l, mid);
                build(node->right, mid + 1, r);
        }

        void nearest(kd_node* node, ptd q, pair<ll, ptd> &ans){
                if(node == NULL) return;

                if(node->left == NULL && node->right == NULL) {
                        if(!(q != node->p)) return; // avoid query point as answer

                        if (ans.first > dis2(node->p, q))
                                ans = {dis2(node->p, q), node->p};

                        return;
                }

                int axis = node->axis;
                int value = node->p.p[axis];

                if(q.p[axis] <= value){
                        nearest(node->left, q, ans);
                        ll diff = value - q.p[axis];
```

```
                if(diff * diff <= ans.ff) nearest(node->right, q, ans);
            }else{
                nearest(node->right, q, ans);
                ll diff = q.p[axis] - value;
                if(diff * diff <= ans.ff) nearest(node->left, q, ans);
            }
        }
    // O(logn) Returns {squared distance, nearest point}
        pair<ll, ptd> nearest(ptd q){
            pair<ll, ptd> ans = {INF, ptd()};
            nearest(root, q, ans);
            return ans;
        }
};
```

5.10 Lineas

```
// add points operators
struct line {
        pt v; lf c; // v: dir, c: mov y, cross(v, p) = c
        line(pt v, lf c) : v(v), c(c) {}
        line(lf a, lf b, lf c) : v({b, -a}), c(c) {} // ax + by = c
        line(pt p, pt q) : v(q - p), c(cross(v, p)) {}
        line(lf m, lf b): v({1, m}), c(b) {}

        bool operator < (line l){ return cross(v, l.v) > 0; }
        bool operator == (line l){ return (abs(cross(v, l.v)) <= E0) && c == l.c; } // abs(c) == abs(l.c)

        lf side(pt p){ return cross(v, p) - c; }
        lf dist(pt p){ return abs(side(p)) / norm(v); }
        lf dist2(pt p){ return side(p) * side(p) / (lf)norm2(v); }
        line perp_through(pt p){ return {p, p + rot90(v)}; } // line perp to v passing through p
        bool cmp_proj(pt p, pt q){ return dot(v, p) < dot(v, q); } // order for points over the line
        // use: auto fsort = [&l1](const pt &a, const pt &b){ return l1.cmp_proj(a, b); };
        line translate(pt t){ return {v, c + cross(v, t)}; }
        line shift_left(lf d){ return {v, c + d*norm(v)}; }
        pt proj(pt p){ return p - rot90(v) * side(p) / norm2(v); } // pt proyected on the line
        pt refl(pt p){ return p - rot90(v) * 2 * side(p) / norm2(v); } // pt reflected on the other side of the line
        bool has(pt p){ return abs(cross(v, p) - c) <= E0; }; // pt on line
        lf evalx(lf x){
                assert(fabsl(v.x) > EPS);
                return (c + v.y * x) / v.x;
        }
};

pt inter_ll(line l1, line l2) {
        if (abs(cross(l1.v, l2.v)) <= EPS) return {INF, INF}; // parallel
        return (l2.v * l1.c - l1.v * l2.c) / cross(l1.v, l2.v); // floating points
}

// bisector divides the angle in 2 equal angles
// interior line goes on the same direction as l1 and l2
line bisector(line l1, line l2, bool interior) {
        // assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be parallel
        lf sign = interior ? 1 : -1;
        return {l2.v / norm(l2.v) + l1.v / norm(l1.v) * sign,
                        l2.c / norm(l2.v) + l1.c / norm(l1.v) * sign};
}
```

5.11 Manhattan

```
struct pt {
    ll x, y;
};

// Returns a list of edges in the format (weight, u, v).
// Passing this list to Kruskal algorithm will give the Manhattan MST.
vector<tuple<ll, ll, ll>> manhattan_mst_edges(vector<pt> ps){
    vl ids(sz(ps));
        forx(i, sz(ps)) ids[i] = i;
    vector<tuple<ll, ll, ll>> edges;

    for (ll rot = 0; rot < 4; rot++) {

            sort(ids.begin(), ids.end(), [&](ll i, ll j){
                return (ps[i].x + ps[i].y) < (ps[j].x + ps[j].y);
        });

        map<ll, ll, greater<ll>> active; // (xs, id)
        for(auto i : ids){
                for(auto it = active.lower_bound(ps[i].x); it != active.end();
```

```
                            active.erase(it++)){
                ll j = it->second;
                if (ps[i].x - ps[i].y > ps[j].x - ps[j].y
                    ) break;
                assert(ps[i].x >= ps[j].x && ps[i].y >=
                    ps[j].y);
                edges.push_back({(ps[i].x - ps[j].x) + (
                    ps[i].y - ps[j].y), i, j});
            }
            active[ps[i].x] = i;
        }
        for (auto &p : ps){ // rotate
            if (rot & 1) p.x *= -1;
            else swap(p.x, p.y);
        }
    }
    return edges;
}
```

5.12 Min Circle

```
// minimo circulo que encierra todos los puntos
// Promedio: O(n), Peor: O(n^2)
Circle min_circle(vector<pt> v){
        random_shuffle(v.begin(), v.end()); // shuffle(
            all(vec), rng);
        auto f2 = [&](int a, int b){
                Circle ans(v[a], v[b]);
                for(int i = 0; i < a; ++ i)
                if(ans.contains(v[i]) == OUT) ans =
                    Circle(v[i], v[a], v[b]);
                return ans;
        };
        auto f1 = [&]( int a ){
                Circle ans(v[a], 0.0L);
                for(int i = 0; i < a; ++i)
                if(ans.contains(v[i]) == OUT) ans = f2( i
                    , a );
                return ans;
        };
        Circle ans( v[0], 0.0L );
        for(int i = 1; i < (int) v.size(); ++i)
                if(ans.contains(v[i]) == OUT ) ans = f1(i
                    );
        return ans;
}
```

5.13 Puntos

```
typedef long double lf;
const lf EPS = 1e-9;
const lf E0 = 0.0L; //Keep = 0 for integer coordinates,
    otherwise = EPS
const lf PI = acos(-1);

struct pt {
        lf x, y;
        pt(){}
        pt(lf a, lf b): x(a), y(b){}
        pt(lf ang): x(cos(ang)), y(sin(ang)){}  // Polar
            unit point: ang(RAD)
        pt operator - (const pt &q) const { return {x - q
            .x , y - q.y }; }
        pt operator + (const pt &q) const { return {x + q
            .x , y + q.y }; }
        // pt operator * (pt p){ return {x * p.x - y * p.
            y, x * p.y + y * p.x}; }
        pt operator * (const lf &t) const { return {x * t
             , y * t }; }
        pt operator / (const lf &t) const { return {x / t
             , y / t }; }
        bool operator == (pt p){ return abs(x - p.x) <=
            EPS && abs(y - p.y) <= EPS; }
        bool operator != (pt p){ return !operator==(p); }
        bool operator < (const pt & q) const { // set /
            sort
                if(fabsl(x - q.x) > E0) return x < q.x;
                return y < q.y;
        }
        void print(){ cout << x << " " << y << "\n"; }
};

pt normalize(pt p){
        lf norm = hypotl(p.x, p.y);
        if(fabsl(norm) > EPS) return {p.x /= norm, p.y /=
             norm};
        else return p;
}

int cmp(lf a, lf b){ return (a + EPS < b ? -1 :(b + EPS <
     a ? 1 : 0)); } // float comparator

// rota ccw
pt rot90(pt p){ return {-p.y, p.x}; }
// w(RAD)
pt rot(pt p, lf w){ return {cosl(w) * p.x - sinl(w) * p.y
    , sinl(w) * p.x + cosl(w) * p.y}; }

lf norm2(pt p){ return p.x * p.x + p.y * p.y; }
lf norm(pt p){ return hypotl(p.x, p.y); }

lf dis2(pt p, pt q){ return norm2(p - q); }
lf dis(pt p, pt q){ return norm(p - q); }

lf arg(pt a){return atan2(a.y, a.x); } // ang(RAD) a x-
    pos
lf dot(pt a, pt b){ return a.x * b.x + a.y * b.y; } // x
```

```
    = 90 -> cos = 0
lf cross(pt a, pt b){ return a.x * b.y - a.y * b.x; } //
    x = 180 -> sin = 0
lf orient(pt a, pt b, pt c){ return cross(b - a, c - a);
    } // AB clockwise = -
int sign(lf x){ return (EPS < x) - (x < -EPS); }

// p inside angle abc (center in a)
bool in_angle(pt a, pt b, pt c, pt p) {
        //assert(fabsl(orient(a, b, c)) > E0);
        if(orient(a, b, c) < -E0)
                return orient(a, b, p) >= -E0 || orient(a
                    , c, p) <= E0;
        return orient(a, b, p) >= -E0 && orient(a, c, p)
            <= E0;
}

lf min_angle(pt a, pt b){ return acos(max((lf)-1.0, min((
    lf)1.0, dot(a, b)/norm(a)/norm(b)))); } // ang(RAD)
lf angle(pt a, pt b){ return atan2(cross(a, b), dot(a, b)
    ); } // ang(RAD)
lf angle(pt a, pt b, pt c){ // ang(RAD) AB AC ccw
        lf ang = angle(b - a, c - a);
        if (ang < 0) ang += 2 * PI;
        return ang;
}

bool half(pt p){ // true if is in (0, 180] (line is x
    axis)
        // assert(p.x != 0 || p.y != 0); // the argument
            of (0, 0) is undefined
        return p.y > 0 || (p.y == 0 && p.x < 0);
}

bool half_from(pt p, pt v = {1, 0}) {
        return cross(v,p) < 0 || (cross(v,p) == 0 && dot(
            v,p) < 0);
}

// polar sort
bool polar_cmp(const pt &a, const pt &b){
        return make_tuple(half(a), 0) < make_tuple(half(b
            ), cross(a,b));
}

void polar_sort(vector<pt> &v, pt o){ // sort points in
    counterclockwise with respect to point o
        sort(v.begin(), v.end(), [&](pt a,pt b) {
                return make_tuple(half(a - o), 0.0, norm2
                    ((a - o))) < make_tuple(half(b - o),
                    cross(a - o, b - o), norm2((b - o)));
        });
}

int cuad(pt p){ // REVISAR
        if(p.x > 0 && p.y >= 0) return 0;
        if(p.x <= 0 && p.y > 0) return 1;
        if(p.x < 0 && p.y <= 0) return 2;
        if(p.x >= 0 && p.y < 0) return 3;
        return -1; // x == 0 && y == 0
}

bool cmp(pt p1, pt p2){
        int c1 = cuad(p1), c2 = cuad(p2);
        return c1 == c2 ? p1.y * p2.x < p1.x * p2.y : c1
            < c2;
}

// O(n*2^d*d)
// Return the max manhattan distance between points with
    d-dimension.
ll max_distance_manhattan(vector<vi> p, int d){
        long long ans = 0;
        for (int msk = 0; msk < (1 << d); msk++) {
                long long mx = LLONG_MIN, mn = LLONG_MAX;
                for (int i = 0; i < n; i++) {
                        long long cur = 0;
                        for (int j = 0; j < d; j++) {
                                if (msk & (1 << j)) cur
                                    += p[i][j];
                                else cur -= p[i][j];
                        }
                        mx = max(mx, cur);
                        mn = min(mn, cur);
                }
                ans = max(ans, mx - mn);
        }

        return ans;
}

ll sd_to_ll(string num, int canDec = 6){
        string nnum = "";
        bool ok = 0;
        for(int i = 0; i < sz(num); i++){
                if (num[i] == '.'){
                        ok = 1;
                        continue;
                }

                if (ok) canDec--;

                nnum.pb(num[i]);
        }

        while(canDec--) nnum.pb('0');
        return stoll(nnum);
}
```

5.14 Poligonos

```
// add Points Lines Segments Circles
```

```cpp
// points in polygon(vector<pt>) ccw or cw
enum {OUT, IN, ON};

lf area(vector<pt>& p){
        lf r = 0.;
        for(int i = 0, n = p.size(); i < n; ++i){
                r += cross(p[i], p[(i + 1) % n]);
        }
        return r / 2; // negative if CW, positive if CCW
}

lf perimeter(vector<pt>& p) {
        lf per = 0;
        for (int i = 0, n = p.size(); i < n; ++i){
                per += norm(p[i] - p[(i + 1) % n]);
        }
        return per;
}

bool is_convex(vector<pt>& p) {
        bool pos = 0, neg = 0;
        for (int i = 0, n = p.size(); i < n; i++) {
                int o = orient(p[i], p[(i + 1) % n], p[(i
                    + 2) % n]);
                if (o > 0) pos = 1;
                if (o < 0) neg = 1;
        }
        return !(pos && neg);
}

int point_in_polygon(vector<pt>& pol, pt& p){
        int wn = 0;
        for(int i = 0, n = pol.size(); i < n; ++i) {
                lf c = orient(p, pol[i], pol[(i + 1) % n
                    ]);
                if(fabsl(c) <= E0 && dot(pol[i] - p, pol
                    [(i + 1) % n] - p) <= E0) return ON;
                    // on segment

                if(c > 0 && pol[i].y <= p.y + E0 && pol[(
                    i + 1) % n].y - p.y > E0) ++wn;
                if(c < 0 && pol[(i + 1) % n].y <= p.y +
                    E0 && pol[i].y - p.y > E0) --wn;
        }
        return wn ? IN : OUT;
}

// O(logn) polygon CCW, remove collinear
int point_in_convex_polygon(const vector<pt> &pol, const
    pt &p){
        int low = 1, high = pol.size() - 1;
        while(high - low > 1){
                int mid = (low + high) / 2;
                if(orient(pol[0], pol[mid], p) >= -E0)
                    low = mid;
                else high = mid;
        }
        if(orient(pol[0], pol[low], p) < -E0) return OUT;
        if(orient(pol[low], pol[high], p) < -E0) return
            OUT;
        if(orient(pol[high], pol[0], p) < -E0) return OUT
            ;

        if(low == 1 && orient(pol[0], pol[low], p) <= E0)
             return ON;
        if(orient(pol[low], pol[high], p) <= E0) return
            ON;
        if(high == (int) pol.size() -1 && orient(pol[high
            ], pol[0], p) <= E0) return ON;
        return IN;
}

// convex polygons in some order (CCW, CW)
vector<pt> minkowski(vector<pt> P, vector<pt> Q) {
        rotate(P.begin(), min_element(P.begin(), P.end())
            , P.end());
        rotate(Q.begin(), min_element(Q.begin(), Q.end())
            , Q.end());

        P.push_back(P[0]), P.push_back(P[1]);
        Q.push_back(Q[0]), Q.push_back(Q[1]);

        vector<pt> ans;
        size_t i = 0, j = 0;
        while(i < P.size() - 2 || j < Q.size() - 2){
                ans.push_back(P[i] + Q[j]);
                lf dt = cross(P[i + 1] - P[i], Q[j + 1] -
                    Q[j]);
                if(dt >= E0 && i < P.size() - 2) ++i;
                if(dt <= E0 && j < Q.size() - 2) ++j;
        }
        return ans;
}

pt centroid(vector<pt>& p){
        pt c{0, 0};
        lf scale = 6. * area(p);
        for (int i = 0, n = p.size(); i < n; ++i){
                c = c + (p[i] + p[(i + 1) % n]) * cross(p
                    [i], p[(i + 1) % n]);
        }
        return c / scale;
}

void normalize(vector<pt>& p) { // polygon CCW
        int bottom = min_element(p.begin(), p.end()) - p.
            begin();
        vector<pt> tmp(p.begin() + bottom, p.end());
        tmp.insert(tmp.end(), p.begin(), p.begin()+bottom
            );
        p.swap(tmp);
        bottom = 0;
}

void remove_col(vector<pt>& p){
```

```
	vector<pt> s;
	for(int i = 0, n = p.size(); i < n; i++){
		if(!on_segment(p[(i - 1 + n) % n], p[(i +
		   1) % n], p[i])) s.push_back(p[i]);
	}
	p.swap(s);
}

void delete_repetead(vector<pt>& p){
	vector<pt> aux;
	sort(p.begin(), p.end());
	for (pt &pi : p){
		if (aux.empty() || aux.back() != pi) aux.
		    push_back(pi);
	}
	p.swap(aux);
}

pt farthest(vector<pt>& p, pt v){ // O(log(n)) only
    CONVEX, v: dir
	int n = p.size();
	if(n < 10){
		int k = 0;
		for(int i = 1; i < n; i++) if(dot(v, (p[i
		    ] - p[k])) > EPS) k = i;
		return p[k];
	}
	pt a = p[1] - p[0];
	int s = 0, e = n, ua = dot(v, a) > EPS;
	if(!ua && dot(v, (p[n - 1] - p[0])) <= EPS)
	    return p[0];
	while(1){
		int m = (s + e) / 2;
		pt c = p[(m + 1) % n] - p[m];
		int uc = dot(v, c) > EPS;
		if(!uc && dot(v, (p[(m - 1 + n) % n] - p[
		    m])) <= EPS) return p[m];
		if(ua && (!uc || dot(v, (p[s] - p[m])) >
		    EPS)) e = m;
		else if(ua || uc || dot(v, (p[s] - p[m]))
		   >= -EPS) s = m, a = c, ua = uc;
		else e = m;
		assert(e > s + 1);
	}
}

vector<pt> cut(vector<pt>& p, line l){
	// cut CONVEX polygon by line l
	// returns part at left of l.pq
	vector<pt> q;
	for(int i = 0, n = p.size(); i < n; i++) {
		int d0 = sign(l.side(p[i]));
		int d1 = sign(l.side(p[(i + 1) % n]));
		if(d0 >= 0) q.push_back(p[i]);

		line m(p[i], p[(i + 1) % n]);
			if(d0 * d1 < 0 && !(abs(cross(l.v, m.v))
			    <= EPS)){
				q.push_back((inter_ll(l, m)));
			}
	}
	return q;
}

// O(n)
vector<pair<int, int>> antipodal(vector<pt>& p){
	vector<pair<int, int>> ans;
	int n = p.size();
	if (n == 2) ans.push_back({0, 1});
	if (n < 3) return ans;
	auto nxt = [&](int x){ return (x + 1 == n ? 0 : x
	     + 1); };
	auto area2 = [&](pt a, pt b, pt c){ return cross(
	    b - a, c - a); };
	int b0 = 0;
	while (abs(area2(p[n - 1], p[0], p[nxt(b0)])) >
	    abs(area2(p[n - 1], p[0], p[b0]))) ++b0;
	for (int b = b0, a = 0; b != 0 && a <= b0; ++a) {
		ans.push_back({a, b});
		while (abs(area2(p[a], p[nxt(a)], p[nxt(b
		    )])) > abs(area2(p[a], p[nxt(a)], p[b
		    ]))){
			b = nxt(b);
			if (a != b0 || b != 0) ans.
			    push_back({a, b});
			else return ans;
		}
		if (abs(area2(p[a], p[nxt(a)], p[nxt(b)])
		    ) == abs(area2(p[a], p[nxt(a)], p[b]))
		    ){
			if (a != b0 || b != n - 1) ans.
			    push_back({a, nxt(b)});
			else ans.push_back({nxt(a), b});
		}
	}
	return ans;
}

// O(n)
// square distance of most distant points, prereq: convex
    , ccw, NO COLLINEAR POINTS
lf callipers(vector<pt>& p){
	int n = p.size();
	lf r = 0;
	for(int i = 0, j = n < 2 ? 0 : 1; i < j; ++i){
		for(;;j = (j + 1) % n){
			r = max(r, norm2(p[i] - p[j]));
			if(cross((p[(i + 1) % n] - p[i]),
			     (p[(j + 1) % n] - p[j])) <=
			    EPS) break;
		}
	}
```

```
        return r;
}

// O(n + m) max_dist between 2 points (pa, pb) of 2
   Convex polygons (a, b)
lf rotating_callipers(vector<pt>& a, vector<pt>& b){ //
   REVISAR
        if (a.size() > b.size()) swap(a, b); // <- del or
            add
        pair<ll, int> start = {-1, -1};
        if(a.size() == 1) swap(a, b);
        for(int i = 0; i < a.size(); i++) start = max(
            start, {norm2(b[0] - a[i]), i});
        if(b.size() == 1) return start.first;

        lf r = 0;
        for(int i = 0, j = start.second; i < b.size(); ++
            i){
                for(;; j = (j + 1) % a.size()){
                        r = max(r, norm2(b[i] - a[j]));
                        if(cross((b[(i + 1) % b.size()] -
                            b[i]), (a[(j + 1) % a.size()]
                            - a[j])) <= EPS) break;
                }
        }
        return r;
}

lf intercircle(vector<pt>& p, circle c){ // area of
   intersection with circle
        lf r=0.;
        for(int i = 0, n = p.size(); i < n; i++){
                int j = (i + 1) % n;
                lf w = intertriangle(c, p[i], p[j]);
                if(cross((p[j] - c.center), (p[i] - c.
                    center)) > 0) r += w;
                else r -= w;
        }
        return abs(r);
}

ll pick(vector<pt>& p){
        ll boundary = 0;
        for (int i = 0, n = p.size(); i < n; i++) {
                int j = (i + 1 == n ? 0 : i + 1);
                boundary += __gcd((ll)abs(p[i].x - p[j].x
                    ), (ll)abs(p[i].y - p[j].y));
        }
        return abs(area(p)) + 1 - boundary / 2;
}

// minimum distance between two parallel lines (non
   necessarily axis parallel)
// such that the polygon can be put between the lines
// O(n) CCW polygon
lf width(vector<pt> &p) {
    int n = (int)p.size();
    if (n <= 2) return 0;
    lf ans = inf;
    int i = 0, j = 1;
    while (i < n){
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n
            ] - p[j]) >= 0) j = (j + 1) % n;
        line l1(p[i], p[(i + 1) % n]);
        ans = min(ans, l1.dist(p[j]));
        i++;
    }
    return ans;
}

// O(n) {minimum perimeter, minimum area} CCW polygon
pair<ld, ld> minimum_enclosing_rectangle(vector<pt> &p) {
        int n = p.size();
        if (n <= 2) return {perimeter(p), 0};
        int mndot = 0;
    lf tmp = dot(p[1] - p[0], p[0]);
        for (int i = 1; i < n; i++) {
                if (dot(p[1] - p[0], p[i]) <= tmp) {
                        tmp = dot(p[1] - p[0], p[i]);
                        mndot = i;
                }
        }
        ld ansP = inf;
        ld ansA = inf;
        int i = 0, j = 1, mxdot = 1;
        while (i < n) {
                pt cur = p[(i + 1) % n] - p[i];
        while (cross(cur, p[(j + 1) % n] - p[j]) >= 0) j
            = (j + 1) % n;
        while (dot(p[(mxdot + 1) % n], cur) >= dot(p[
            mxdot], cur)) mxdot = (mxdot + 1) % n;
        while (dot(p[(mndot + 1) % n], cur) <= dot(p[
            mndot], cur)) mndot = (mndot + 1) % n;
        line l1(p[i], p[(i + 1) % n]);

        // minimum perimeter
        ansP = min(ansP, 2.0 * ((dot(p[mxdot], cur) /
            norm(cur) - dot(p[mndot], cur) / norm(cur)) +
            l1.dist(p[j])));
        // minimum area
        ansA = min(ansA, (dot(p[mxdot], cur) / norm(cur)
            - dot(p[mndot], cur) / norm(cur)) * l1.dist(p[
            j]));
        i++;
    }

    return {ansP, ansA};
}

// maximum distance from a convex polygon to another
   convex polygon
lf maximum_dist_from_polygon_to_polygon(vector<pt> &u,
```

```
    vector<pt> &v){ //O(n)
    int n = (int)u.size(), m = (int)v.size();
    lf ans = 0;
    if (n < 3 || m < 3) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) ans = max(ans,
                dis2(u[i], v[j]));
        }
        return sqrt(ans);
    }
    if (u[0].x > v[0].x) swap(n, m), swap(u, v);
    int i = 0, j = 0, step = n + m + 10;
    while (j + 1 < m && v[j].x < v[j + 1].x) j++ ;
    while (step--) {
        if (cross(u[(i + 1) % n] - u[i], v[(j + 1) % m] -
            v[j]) >= 0) j = (j + 1) % m;
        else i = (i + 1) % n;
        ans = max(ans, dis2(u[i], v[j]));
    }
    return sqrt(ans);
}
// -------------------------------------------------------------

pt project_from_point_to_seg(pt a, pt b, pt c) {
    double r = dis2(a, b);
    if (sign(r) == 0) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
// minimum distance from point c to segment ab
lf pt_to_seg(pt a, pt b, pt c) {
    return dis(c, project_from_point_to_seg(a, b, c));
}

pair<pt, int> point_poly_tangent(vector<pt> &p, pt Q, int
    dir, int l, int r) {
    while (r - l > 1) {
        int mid = (l + r) >> 1;
        bool pvs = sign(orient(Q, p[mid], p[mid - 1])) !=
            -dir;
        bool nxt = sign(orient(Q, p[mid], p[mid + 1])) !=
            -dir;
        if (pvs && nxt) return {p[mid], mid};
        if (!(pvs || nxt)) {
            auto p1 = point_poly_tangent(p, Q, dir, mid +
                1, r);
            auto p2 = point_poly_tangent(p, Q, dir, l,
                mid - 1);
            return sign(orient(Q, p1.first, p2.first)) ==
                dir ? p1 : p2;
        }
        if (!pvs) {
            if (sign(orient(Q, p[mid], p[l])) == dir)  r
                = mid - 1;
            else if (sign(orient(Q, p[l], p[r])) == dir)
                r = mid - 1;
            else l = mid + 1;
        }
        if (!nxt) {
            if (sign(orient(Q, p[mid], p[l])) == dir)  l
                = mid + 1;
            else if (sign(orient(Q, p[l], p[r])) == dir)
                r = mid - 1;
            else l = mid + 1;
        }
    }
    pair<pt, int> ret = {p[l], l};
    for (int i = l + 1; i <= r; i++) ret = sign(orient(Q,
        ret.first, p[i])) != dir ? make_pair(p[i], i) :
        ret;
    return ret;
}
// (ccw, cw) tangents from a point that is outside this
    convex polygon
// returns indexes of the points
// ccw means the tangent from Q to that point is in the
    same direction as the polygon ccw direction
pair<int, int> tangents_from_point_to_polygon(vector<pt>
    &p, pt Q){
    int ccw = point_poly_tangent(p, Q, 1, 0, (int)p.size
        () - 1).second;
    int cw = point_poly_tangent(p, Q, -1, 0, (int)p.size
        () - 1).second;
    return make_pair(ccw, cw);
}

// minimum distance from a point to a convex polygon
// it assumes point lie strictly outside the polygon
lf dist_from_point_to_polygon(vector<pt> &p, pt z) {
    lf ans = inf;
    int n = p.size();
    if (n <= 3) {
        for(int i = 0; i < n; i++) ans = min(ans,
            pt_to_seg(p[i], p[(i + 1) % n], z));
        return ans;
    }
    pair<int, int> dum = tangents_from_point_to_polygon(p
        , z);
    int r = dum.first;
    int l = dum.second;
    if(l > r) r += n;
    while (l < r) {
        int mid = (l + r) >> 1;
        lf left = dis2(p[mid % n], z), right= dis2(p[(mid
            + 1) % n], z);
        ans = min({ans, left, right});
```

```cpp
            if(left < right) r = mid;
            else l = mid + 1;
        }
        ans = sqrt(ans);
        ans = min(ans, pt_to_seg(p[l % n], p[(l + 1) % n], z));
        ans = min(ans, pt_to_seg(p[l % n], p[(l - 1 + n) % n], z));
        return ans;
}

// minimum distance from a convex polygon to another convex polygon
// the polygon doesnot overlap or touch
lf dist_from_polygon_to_polygon(vector<pt> &p1, vector<pt> &p2) { // O(n log n)
        lf ans = inf;
        for (int i = 0; i < p1.size(); i++) {
            ans = min(ans, dist_from_point_to_polygon(p2, p1[i]));
        }
        for (int i = 0; i < p2.size(); i++) {
            ans = min(ans, dist_from_point_to_polygon(p1, p2[i]));
        }
        return ans;
}

// it returns a point such that the sum of distances
// from that point to all points in p  is minimum
// O(n log^2 MX)
PT geometric_median(vector<PT> p) {
        auto tot_dist = [&](PT z) {
            double res = 0;
            for (int i = 0; i < p.size(); i++) res += dist(p[i], z);
            return res;
        };
        auto findY = [&](double x) {
            double yl = -1e5, yr = 1e5;
            for (int i = 0; i < 60; i++) {
                double ym1 = yl + (yr - yl) / 3;
                double ym2 = yr - (yr - yl) / 3;
                double d1 = tot_dist(PT(x, ym1));
                double d2 = tot_dist(PT(x, ym2));
                if (d1 < d2) yr = ym2;
                else yl = ym1;
            }
            return pair<double, double> (yl, tot_dist(PT(x, yl)));
        };
    double xl = -1e5, xr = 1e5;
    for (int i = 0; i < 60; i++) {
        double xm1 = xl + (xr - xl) / 3;
        double xm2 = xr - (xr - xl) / 3;
        double y1, d1, y2, d2;
        auto z = findY(xm1); y1 = z.first; d1 = z.second;
        z = findY(xm2); y2 = z.first; d2 = z.second;
        if (d1 < d2) xr = xm2;
        else xl = xm1;
    }
    return {xl, findY(xl).first };
}

// ear decomposition, O(n^3) but faster
vector<vector<pt>> triangulate(vector<pt> p) {
        vector<vector<pt>> v;

        while (p.size() >= 3){
                for (int i = 0, n = p.size(); i < n; i++) {
                        int pre = i == 0 ? n - 1 : i - 1;;
                        int nxt = i == n - 1 ? 0 : i + 1;;
                        lf ori = orient(p[i], p[pre], p[nxt]);

                        if (ori < 0){
                                int ok = 1;
                                for (int j = 0; j < n; j++){
                                        if (j == i || j == pre || j == nxt)continue;
                                        vector<pt> tr = {p[i], p[pre], p[nxt]};
                                        if (point_in_polygon(tr , p[j]) != OUT){
                                                ok = 0;
                                                break;
                                        }
                                }

                                if (ok){
                                        v.push_back({p[pre], p[i], p[nxt]});
                                        p.erase(p.begin() + i);
                                        break;
                                }
                        }
                }
        }
        return v;
}

// CH DP
```

```
// vecs contains all vectors u -> v that lie within the
   original polygon
sort(vecs.begin(), vecs.end(), cmp);
for (auto [u, v] : vecs)
    for (int s = 0; s < n; s++)
        dp[s][v] += dp[s][u];

// Suma del area de todos los triangulos de un poligono
int main() {
        int n; cin >> n;

        vector<pt> pts(n);

        forx(i, n) cin >> pts[i].x >> pts[i].y;

        vector<pt> pref(n + 1);

        for(int i = 1; i < n + 1; ++i){
                pref[i] = pref[i - 1] + pts[i - 1];
        }

        ld totalT = 0;

        for(ll i = 0; i < n; i++){
                pt p1 = (pts[i] * (ld)i) - pref[i];
                pt p2 = (pref[n] - pref[i + 1]) - (pts[i]
                    * (ld)(n - 1 - i));

                totalT += cross(p1, p2);
        }

        // area = totalT / 2

}
```

5.15 Segmentos

```
// add Lines Points

bool in_disk(pt a, pt b, pt p){ // pt p inside ab disk
        return dot(a - p, b - p) <= E0;
}

bool on_segment(pt a, pt b, pt p) { // p on ab
        return orient(a, b, p) == 0 && in_disk(a, b, p);
}

// ab crossing cd
bool proper_inter(pt a, pt b, pt c, pt d, pt& out) {
        lf oa = orient(c, d, a);
        lf ob = orient(c, d, b);
        lf oc = orient(a, b, c);
        lf od = orient(a, b, d);
        // Proper intersection exists iff opposite signs
        if (oa * ob < 0 && oc * od < 0) {
                out = (a * ob - b * oa) / (ob - oa);
                return true;
        }
                return false;
}

// intersection bwn segments
set<pt> inter_ss(pt a, pt b, pt c, pt d) {
        pt out;
        if (proper_inter(a, b, c, d, out)) return {out};
        set<pt> s;
        if (on_segment(c, d, a)) s.insert(a); // a in cd
        if (on_segment(c, d, b)) s.insert(b); // b in cd
        if (on_segment(a, b, c)) s.insert(c); // c in ab
        if (on_segment(a, b, d)) s.insert(d); // d in ab
        return s;
}

lf pt_to_seg(pt a, pt b, pt p) { // p to ab
        if (a != b) {
                line l(a, b);
                if (l.cmp_proj(a, p) && l.cmp_proj(p, b))
                    // if closest to projection = (a, p,
                    b)
                        return l.dist(p); // output
                            distance to line
        }
        return min(norm(p - a), norm(p - b)); //
           otherwise distance to A or B
}

lf seg_to_seg(pt a, pt b, pt c, pt d) {
        pt dummy;
        if (proper_inter(a, b, c, d, dummy)) return 0; //
            ab intersects cd
        return min({pt_to_seg(a, b, c), pt_to_seg(a, b, d
           ), pt_to_seg(c, d, a), pt_to_seg(c, d, b)});
           // try the 4 pts
}

int length_union(vector<pt>& a){ // REVISAR
        int n = a.size();
        vector<pair<int, bool>> x(n * 2);
        for (int i = 0; i < n; i++) {
                x[i * 2] = {a[i].x, false};
                x[i * 2 + 1] = {a[i].y, true};
        }
        sort(x.begin(), x.end());
        int result = 0;
        int c = 0;
        for (int i = 0; i < n * 2; i++) {
                if (i > 0 && x[i].first > x[i - 1].first
                    && c > 0) result += x[i].first - x[i -
                     1].first;
                if (x[i].second) c--;
                else c++;
        }
        return result;
}
```

5.16 Triangle Union

```cpp
// Area of the union of a set of n triangles
// T(n^2 logn) M(n)
typedef double dbl;
const dbl eps = 1e-9;
inline bool eq(dbl x, dbl y){
    return fabs(x - y) < eps;
}
inline bool lt(dbl x, dbl y){
    return x < y - eps;
}
inline bool gt(dbl x, dbl y){
    return x > y + eps;
}
inline bool le(dbl x, dbl y){
    return x < y + eps;
}
inline bool ge(dbl x, dbl y){
    return x > y - eps;
}
struct ptT{
    dbl x, y;
    ptT(){}
    ptT(dbl x, dbl y): x(x), y(y){}
    inline ptT operator - (const ptT & p)const{
        return ptT{x - p.x, y - p.y};
    }
    inline ptT operator + (const ptT & p)const{
        return ptT{x + p.x, y + p.y};
    }
    inline ptT operator * (dbl a)const{
        return ptT{x * a, y * a};
    }
    inline dbl cross(const ptT & p)const{
        return x * p.y - y * p.x;
    }
    inline dbl dot(const ptT & p)const{
        return x * p.x + y * p.y;
    }
    inline bool operator == (const ptT & p)const{
        return eq(x, p.x) && eq(y, p.y);
    }
};
struct LineT{
    ptT p[2];
    LineT(){}
    LineT(ptT a, ptT b):p{a, b}{}
    ptT vec()const{
        return p[1] - p[0];
    }
    ptT& operator [](size_t i){
        return p[i];
    }
};
inline bool lexComp(const ptT & l, const ptT & r){
    if(fabs(l.x - r.x) > eps){
        return l.x < r.x;
    }
    else return l.y < r.y;
}

vector<ptT> interSegSeg(LineT l1, LineT l2){
    if(eq(l1.vec().cross(l2.vec()), 0)){
        if(!eq(l1.vec().cross(l2[0] - l1[0]), 0))
            return {};
        if(!lexComp(l1[0], l1[1]))
            swap(l1[0], l1[1]);
        if(!lexComp(l2[0], l2[1]))
            swap(l2[0], l2[1]);
        ptT l = lexComp(l1[0], l2[0]) ? l2[0] : l1[0];
        ptT r = lexComp(l1[1], l2[1]) ? l1[1] : l2[1];
        if(l == r)
            return {l};
        else return lexComp(l, r) ? vector<ptT>{l, r} :
           vector<ptT>();
    }
    else{
        dbl s = (l2[0] - l1[0]).cross(l2.vec()) / l1.vec
           ().cross(l2.vec());
        ptT inter = l1[0] + l1.vec() * s;
        if(ge(s, 0) && le(s, 1) && le((l2[0] - inter).dot
           (l2[1] - inter), 0))
            return {inter};
        else
            return {};
    }
}

inline char get_segtype(LineT segment, ptT other_point){
    if(eq(segment[0].x, segment[1].x))
        return 0;
    if(!lexComp(segment[0], segment[1]))
        swap(segment[0], segment[1]);
    return (segment[1] - segment[0]).cross(other_point -
       segment[0]) > 0 ? 1 : -1;
}

dbl union_area(vector<tuple<ptT, ptT, ptT> > triangles){
    vector<LineT> segments(3 * triangles.size());
    vector<char> segtype(segments.size());
    for(size_t i = 0; i < triangles.size(); i++){
        ptT a, b, c;
```

```
        tie(a, b, c) = triangles[i];
        segments[3 * i] = lexComp(a, b) ? LineT(a, b) : LineT(b, a);
        segtype[3 * i] = get_segtype(segments[3 * i], c);
        segments[3 * i + 1] = lexComp(b, c) ? LineT(b, c) : LineT(c, b);
        segtype[3 * i + 1] = get_segtype(segments[3 * i + 1], a);
        segments[3 * i + 2] = lexComp(c, a) ? LineT(c, a) : LineT(a, c);
        segtype[3 * i + 2] = get_segtype(segments[3 * i + 2], b);
    }
    vector<dbl> k(segments.size()), b(segments.size());
    for(size_t i = 0; i < segments.size(); i++){
        if(segtype[i]){
            k[i] = (segments[i][1].y - segments[i][0].y) / (segments[i][1].x - segments[i][0].x);
            b[i] = segments[i][0].y - k[i] * segments[i][0].x;
        }
    }
    dbl ans = 0;
    for(size_t i = 0; i < segments.size(); i++){
        if(!segtype[i])
            continue;
        dbl l = segments[i][0].x, r = segments[i][1].x;
        vector<pair<dbl, int> > evts;
        for(size_t j = 0; j < segments.size(); j++){
            if(!segtype[j] || i == j)
                continue;
            dbl l1 = segments[j][0].x, r1 = segments[j][1].x;
            if(ge(l1, r) || ge(l, r1))
                continue;
            dbl common_l = max(l, l1), common_r = min(r, r1);
            auto pts = interSegSeg(segments[i], segments[j]);
            if(pts.empty()){
                dbl yl1 = k[j] * common_l + b[j];
                dbl yl = k[i] * common_l + b[i];
                if(lt(yl1, yl) == (segtype[i] == 1)){
                    int evt_type = -segtype[i] * segtype[j];
                    evts.emplace_back(common_l, evt_type);
                    evts.emplace_back(common_r, -evt_type);
                }
            }
            else if(pts.size() == 1u){
                dbl yl = k[i] * common_l + b[i], yl1 = k[j] * common_l + b[j];
                int evt_type = -segtype[i] * segtype[j];
                if(lt(yl1, yl) == (segtype[i] == 1)){
                    evts.emplace_back(common_l, evt_type);
                    evts.emplace_back(pts[0].x, -evt_type);
                }
                yl = k[i] * common_r + b[i], yl1 = k[j] * common_r + b[j];
                if(lt(yl1, yl) == (segtype[i] == 1)){
                    evts.emplace_back(pts[0].x, evt_type);
                    evts.emplace_back(common_r, -evt_type);
                }
            }
            else{
                if(segtype[j] != segtype[i] || j > i){
                    evts.emplace_back(common_l, -2);
                    evts.emplace_back(common_r, 2);
                }
            }
        }
        evts.emplace_back(l, 0);
        sort(evts.begin(), evts.end());
        size_t j = 0;
        int balance = 0;
        while(j < evts.size()){
            size_t ptr = j;
            while(ptr < evts.size() && eq(evts[j].first, evts[ptr].first)){
                balance += evts[ptr].second;
                ++ptr;
            }
            if(!balance && !eq(evts[j].first, r)){
                dbl next_x = ptr == evts.size() ? r : evts[ptr].first;
                ans -= segtype[i] * (k[i] * (next_x + evts[j].first) + 2 * b[i]) * (next_x - evts[j].first);
            }
            j = ptr;
        }
    }
    return ans/2;
}
```

6 Grafos

6.1 2sat

```
// O(n+m)
// (x1 or y1) and (x2 or y2) and ... and (xn or yn)
```

```cpp
struct sat2{
	vector<vector<vi>> g;
	vector<bool> vis, val;
	stack<int> st;
	vi comp;
	int n;

	sat2(int n):n(n),g(2, vector<vi>(2*n)),vis(2*n),
	    val(2*n),comp(2*n){}

	int neg(int x){return 2*n-x-1;} // get not x

	void make_true(int u){add_edge(neg(u), u);}
	void make_false(int u){make_true(neg(u));}

	void add_or(int u, int v){implication(neg(u),v);}
	     // (u or v)
	void diff(int u, int v){eq(u, neg(v));} // u != v
	void eq(int u, int v){
		implication(u, v);
		implication(v, u);
	}

	void implication(int u,int v){
		add_edge(u, v);
		add_edge(neg(v),neg(u));
	}

	void add_edge(int u, int v){
		g[0][u].push_back(v);
		g[1][v].push_back(u);
	}

	void dfs(int id, int u, int t=0){
		vis[u]=true;
		for(auto &v:g[id][u])
			if(!vis[v])dfs(id, v, t);
		if(id)comp[u]=t;
		else st.push(u);
	}

	void kosaraju() {
		for(int u=0;u<n;++u){
			if(!vis[u])dfs(0, u);
			if(!vis[neg(u)])dfs(0, neg(u));
		}
		vis.assign(2*n, false);
		int t=0;
		while(!st.empty()){
			int u=st.top();st.pop();
			if(!vis[u])dfs(1, u, t++);
		}
	}

	// return true if satisfiable, fills val[]
	bool check(){
		kosaraju();
		for(int i=0;i<n;++i){
			if(comp[i]==comp[neg(i)])return
			    false;
			val[i]=comp[i]>comp[neg(i)];
		}
		return true;
	}
};

int m,n;cin>>m>>n;
sat2 s(n);
char c1,c2;
for(int a,b,i=0;i<m;++i){
	cin>>c1>>a>>c2>>b;
	a--;b--;
	if(c1=='-')a=s.neg(a);
	if(c2=='-')b=s.neg(b);
	s.add_or(a,b);
}
if(s.check()){
	for(int i=0;i<n;++i)cout<<(s.val[i]?'+':'-')<<" "
	    ;
	cout<<"\n";
}else cout<<"IMPOSSIBLE\n";
```

6.2 Bellman Ford

```cpp
// O(V*E)
vi bellman_ford(vector<vii> &adj, int s, int n){
	vi dist(n, INF); dist[s] = 0;
	for (int i = 0; i<n-1; i++){
		bool modified = false;
		for (int u = 0; u<n; u++)
			if (dist[u] != INF)
				for (auto &[v, w] : adj[u
				    ]){
					if (dist[v] <=
					    dist[u] + w)
					    continue;
					dist[v] = dist[u]
					     + w;
					modified = true;
				}
		if (!modified) break;
	}

	bool negativeCicle = false;
	for (int u = 0; u<n; u++)
		if (dist[u] != INF)
			for (auto &[v, w] : adj[u]){
				if (dist[v] > dist[u] + w
				    ) negativeCicle = true
				    ;
			}

	return dist;
}
```

6.3 Block Cut Tree

```
// O(n) build
// bi_connected save the edges
const int maxn = 1e5+5;
int lowLink[maxn] , dfn[maxn];
vector<vector<ii>> bi_connected;
stack<ii> comps;
int ndfn;

void tarjan(vector<vi>& adj, int u=0, int par=-1){
        dfn[u] = lowLink[u] = ndfn++;
        for(auto &v : adj[u]){
                if (v != par && dfn[v] < dfn[u])
                        comps.push({u, v});
                if(dfn[v] == -1){
                        tarjan(adj, v, u);
                        lowLink[u] = min(lowLink[u] ,
                            lowLink[v]);
                        if (lowLink[v] >= dfn[u]){
                                bi_connected.emplace_back
                                    (vector<ii>());
                                ii edge;
                                do{
                                        edge = comps.top
                                            ();
                                        comps.pop();
                                        bi_connected.back
                                            ().
                                            emplace_back(
                                            edge);

                                }while(edge.first != u ||
                                     edge.second != v);
                                reverse(all(bi_connected.
                                    back()));
                        }
                }else if(v != par){
                        lowLink[u] = min(lowLink[u] , dfn
                            [v]);
                }
        }
}

void init(vector<vi>& adj){
        for(int i=0;i<sz(adj);++i)
                dfn[i]=-1;
        bi_connected.clear();
        comps=stack<ii>();
        ndfn=0;
        tarjan(adj);
}
```

6.4 Bridges Online

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
        par.resize(n);
        dsu_2ecc.resize(n);
        dsu_cc.resize(n);
        dsu_cc_size.resize(n);
        lca_iteration = 0;
        last_visit.assign(n, 0);
        for (int i=0; i<n; ++i) {
                dsu_2ecc[i] = i;
                dsu_cc[i] = i;
                dsu_cc_size[i] = 1;
                par[i] = -1;
        }
        bridges = 0;
}

int find_2ecc(int v) {
        if (v == -1)
                return -1;
        return dsu_2ecc[v] == v ? v : dsu_2ecc[v] =
            find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
        v = find_2ecc(v);
        return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(
            dsu_cc[v]);
}

void make_root(int v) {
        int root = v;
        int child = -1;
        while (v != -1) {
                int p = find_2ecc(par[v]);
                par[v] = child;
                dsu_cc[v] = root;
                child = v;
                v = p;
        }
        dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
        ++lca_iteration;
        vector<int> path_a, path_b;
        int lca = -1;
        while (lca == -1) {
                if (a != -1) {
                        a = find_2ecc(a);
```

```
                                path_a.push_back(a);
                                if (last_visit[a] ==
                                    lca_iteration){
                                        lca = a;
                                        break;
                                        }
                                last_visit[a] = lca_iteration;
                                a = par[a];
                        }
                        if (b != -1) {
                                b = find_2ecc(b);
                                path_b.push_back(b);
                                if (last_visit[b] ==
                                    lca_iteration){
                                        lca = b;
                                        break;
                                        }
                                last_visit[b] = lca_iteration;
                                b = par[b];
                        }

                }

                for (int v : path_a) {
                        dsu_2ecc[v] = lca;
                        if (v == lca)
                                break;
                        --bridges;
                }
                for (int v : path_b) {
                        dsu_2ecc[v] = lca;
                        if (v == lca)
                                break;
                        --bridges;
                }
}

void add_edge(int a, int b) {
        a = find_2ecc(a);
        b = find_2ecc(b);
        if (a == b)
                return;

        int ca = find_cc(a);
        int cb = find_cc(b);

        if (ca != cb) {
                ++bridges;
                if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
                        swap(a, b);
                        swap(ca, cb);
                }
                make_root(a);
                par[a] = dsu_cc[a] = b;
                dsu_cc_size[cb] += dsu_cc_size[a];
        } else {
                merge_path(a, b);
```

```
                }
        }
```

6.5 Camino Mas Corto De Longitud Fija

```
/*
Modificar operacion * de matrix de esta forma:
En la exponenciacion binaria inicializar matrix ans = b
*/
const ll INFL = 2e18;
matrix operator * (const matrix &b){
        matrix ans(this->r, b.c, vector<vl>(this->r, vl(b
            .c, INFL)));

        for (int i = 0; i<this->r; i++) {
                for (int k = 0; k<b.r; k++){
                        for (int j = 0; j<b.c; j++){
                                ans.m[i][j] = min(ans.m[i
                                    ][j], m[i][k] + b.m[k
                                    ][j]);
                        }
                }
        }
        return ans;
}

int main() {

        int n, m, k; cin >> n >> m >> k;

        vector<vl> adj(n, vl(n, INFL));

        for (int i = 0; i<m; i++){
                ll a, b, c; cin >> a >> b >> c; a--; b--;
                adj[a][b] = min(adj[a][b], c);
        }

        matrix graph(n, n, adj);
        graph = pow(graph, k-1);

        cout << (graph.m[0][n-1]==INFL ? -1 : graph.m[0][
            n-1]) << "\n";

        return 0;
}
```

6.6 Clique

```
/**
 * Credit: kactl
 * Given a graph as a symmetric bitset matrix (without
   any self edges)
 * Finds the maximum clique
 * Can be used to find the maximum independent set by
   finding a clique of the complement graph.
```

```
 * Runs in about 1s for n=155, and faster for sparse
   graphs
 * 0 indexed
 */
const int N = 40;
typedef vector<bitset<N>> graph;
struct Maxclique {
  double limit = 0.025, pk = 0;
  struct Vertex {
    int i, d = 0;
  };
  typedef vector<Vertex> vv;
  graph e;
  vv V;
  vector<vector<int>> C;
  vector<int> qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i
        ];
    sort(r.begin(), r.end(), [](auto a, auto b) {
      return a.d > b.d;
    });
    int mxD = r[0].d;
    for (int i = 0; i < sz(r); i++) r[i].d = min(i, mxD)
        + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v : R) if (e[R.back().i][v.i]) T.push_back
          ({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) +
            1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) {
            return e[v.i][i];
          };
          while (any_of(C[k].begin(), C[k].end(), f)) k
              ++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        for (int k = mnk; k <= mxk; k++) for (int i : C[k
            ])
              T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  Maxclique(graph g) : e(g), C(sz(e) + 1), S(sz(C)), old(
      S) {
    for (int i = 0; i < sz(e); i++) V.push_back({i});
  }
  vector<int> solve() { // returns the clique
    init(V), expand(V);
    return qmax;
  }
};
```

6.7 Cycle Directed

```
vector<vi> adj;
vi parent,color;
int cy0, cy1;

bool dfs(int v){
        color[v]=1;
        for(int u:adj[v]){
                if(color[u]==0){
                        parent[u]=v;
                        if(dfs(u))return true;
                }else if(color[u]==1){
                        cy1=v;
                        cy0=u;
                        return true;
                }
        }
        color[v]=2;
        return false;
}

// O(m)
void find_cycle(int n){
        color.assign(n, 0);
        parent.assign(n, -1);
        cy0=-1;
        for(int v=0;v<n;++v){
                if(color[v]==0){
                        if(dfs(v))break;
                }
        }
        if(cy0==-1){
                cout<<"IMPOSSIBLE\n";
                return;
        }
        vi cycle;
        cycle.push_back(cy0);
```

```
        for(int v=cy1;v!=cy0;v=parent[v])cycle.push_back(
            v);
        cycle.push_back(cy0);
        reverse(cycle.begin(),cycle.end());
        print(cycle);
}
```

6.8 Cycle Undirected

```
vector<vi> adj;
vector<bool> visited;
int cy0,cy1;
vi parent;

bool dfs(int v, int par){
        visited[v]=true;
        for(int u:adj[v]){
                if(u==par)continue;
                if(visited[u]){
                        cy1=v;
                        cy0=u;
                        return true;
                }
                parent[u]=v;
                if(dfs(u,parent[u]))return true;
        }
        return false;
}

// O(m)
void find_cycle(int n){
        visited.assign(n, false);
        parent.assign(n, -1);
        cy0=-1;
        for(int v=0;v<n;++v){
                if(!visited[v]){
                if(dfs(v, parent[v]))break;
                }
        }
        if(cy0==-1){
                cout<<"IMPOSSIBLE\n";
                return;
        }
        vi cycle;
        cycle.push_back(cy0);
        for(int v=cy1;v!=cy0;v=parent[v])cycle.push_back(
            v);
        cycle.push_back(cy0);
        print(cycle);
}
```

6.9 Dial Algorithm

```
const int maxn = 2e5+5;
vector<ii> adj[maxn];

// O(n*k+m)
// bfs for edge weights in the range [0, k]
void bfs(int s, int n, int k){
        vector<queue<int>> qs(k+1, queue<int>());
        vector<bool> vis(n, false);
        vector<int> dist(n, 1e9);
        qs[0].push(s);
        dist[s]=0;
        int pos=0,num=1;
        while(num){
                while(qs[pos%(k+1)].empty())pos++;
                int u=qs[pos%(k+1)].front();
                qs[pos%(k+1)].pop();
                num--;
                if(vis[u])continue;
                vis[u]=true;
                for(auto [w,v]:adj[u]){
                        if(dist[v]>dist[u]+w){
                                dist[v]=dist[u]+w;
                                qs[dist[v]%(k+1)].push(v)
                                    ;
                                num++;
                        }
                }
        }
}
```

6.10 Dijkstra

```
// O ((V+E)*log V)
vl dijkstra(vector<vector<pll>> &adj, int s, int n){
        vl dist(n, INFL); dist[s] = 0;
        priority_queue<pll, vector<pll>, greater<pll> >
            pq; pq.push(pll(0, s));
        while(!pq.empty()){
                pll front = pq.top(); pq.pop();
                ll d = front.first, u = front.second;
                if (d > dist[u]) continue;

                for (auto &[v, w] : adj[u]){
                        if (dist[u] + w < dist[v]){
                                dist[v] = dist[u] + w;
                                pq.push(pll(dist[v], v));
                        }
                }
        }
        return dist;
}
```

6.11 Dijkstra Sparse Graphs

```
// O(E*log(V))
vl dijkstra(vector<vector<pll>> &adj, int s, int n){
	vl dist(n, INFL); dist[s] = 0;
	set<pll> pq;
	pq.insert(pll(0, s));
	while(!pq.empty()){
		pll front = *pq.begin(); pq.erase(pq.begin());
		ll d = front.first, u = front.second;
		for (auto &[v, w] : adj[u]){
			if (dist[u] + w < dist[v]){
				pq.erase(pll(dist[v], v));
				dist[v] = dist[u] + w;
				pq.insert(pll(dist[v], v));
			}
		}
	}
	return dist;
}
```

6.12 Eulerian Path Directed

```
const int maxn = 1e5+5;
vector<int> adj[maxn],path;
int out[maxn],in[maxn]; // remember

void dfs(int v){
	while(!adj[v].empty()){
		int u=adj[v].back();
		adj[v].pop_back();
		dfs(u);
	}
	path.push_back(v);
}

// n -> nodes, m -> edges, s -> start, e -> end
void eulerian_path(int n, int m, int s, int e){
	for(int i=0;i<n;++i){
		if(i==s || i==e)continue;
		if(in[i]!=out[i]){
			cout<<"IMPOSSIBLE\n";
			return;
		}
	}
	if(out[s]-in[s]!=1 || in[e]-out[e]!=1){
		cout<<"IMPOSSIBLE\n";
		return;
	}
	dfs(s);
	reverse(path.begin(), path.end());
	if(sz(path)!=m+1 || path.back()!=e)cout<<"IMPOSSIBLE\n";
	else print(path);
}
```

6.13 Eulerian Path Undirected

```
const int maxn = 1e5+5;
const int maxm = 2e5+5;
vector<ii> adj[maxn]; // adj[a].push_back({b, i});
vector<int> path;
int grade[maxn]; // remember
bool vis[maxm];

void dfs(int v){
	while(!adj[v].empty()){
		ii x=adj[v].back();
		adj[v].pop_back();
		if(vis[x.second])continue;
		vis[x.second]=true;
		dfs(x.first);
	}
	path.push_back(v+1);
}

// check if end is equal to start
void eulerian_path(int n, int m, int s){
	for(int i=0;i<n;++i){
		if(grade[i]%2!=0){
			cout<<"IMPOSSIBLE\n";
			return;
		}
	}
	dfs(s);
	if(sz(path)!=m+1)cout<<"IMPOSSIBLE\n";
	else print(path);
}
```

6.14 Floyd Warshall

```
// O(n^3)
vector<vi> adjMat(n+1, vi(n+1));
//Condicion previa: adjMat[i][j] contiene peso de la arista (i, j)
//o INF si no existe esa arista y adjMat[i][i] = 0
for (int k = 0; k < n; ++k) {
	for (int i = 0; i < n; ++i) {
		for (int j = 0; j < n; ++j) {
			if (adjMat[i][k] < INF && adjMat[k][j] < INF)
				adjMat[i][j] = min(adjMat[i][j], adjMat[i][k] + adjMat[k][j]);
```

```
                }
        }
}
```

6.15 Kosaraju

```
const int maxn = 1e5+5;
// construir el grafo inverso
// remember adj[a]->b, adj_rev[b]->a
vi adj_rev[maxn],adj[maxn];
bool used[maxn];
int idx[maxn]; // componente de cada nodo
vi order,comp;

// O(n+m)
void dfs1(int v){
        used[v]=true;
        for(int u:adj[v])
                if(!used[u])dfs1(u);
        order.push_back(v);
}

void dfs2(int v){
        used[v]=true;
        comp.push_back(v);
        for(int u:adj_rev[v])
                if(!used[u])dfs2(u);
}

// returna el numero de componentes
int init(int n){
        for(int i=0;i<n;++i)if(!used[i])dfs1(i);
        for(int i=0;i<n;++i)used[i]=false;
        reverse(all(order));
        int j=0;
        for(int v:order){
                if(!used[v]){
                        dfs2(v);
                        for(int u:comp)idx[u]=j;
                        comp.clear();
                        j++;
                }
        }
        return j;
}
```

6.16 kruskal

```
// peso, nodo a, node b
vector<tuple<int,int,int>> edges;
struct DSU{};

// O(m*log(m))
void kruskal(int n){
        sort(all(edges));
        DSU dsu(n);
        ll ans=0;
        for(auto& [w,u,v]:edges){
                if(dsu.get(u)!=dsu.get(v)){
                        dsu.unite(u, v);
                        ans+=w;
                }
        }
        if(dsu.sets!=1)cout<<"IMPOSSIBLE\n";
        else cout<<ans<<"\n";
}
```

6.17 Prim

```
// O(E * log V)
// check: primer parametro de prim
// check: cuando no hay mst
vector<vii> adj;
vi tomado;
priority_queue<ii> pq;
void process(int u){
        tomado[u] = 1;
        for (auto &[v, w] : adj[u]){
                if (!tomado[v]) pq.emplace(-w, -v);
        }
}

int prim(int v, int n){
        tomado.assign(n, 0);
        process(0);
        int mst_costo = 0, tomados = 0;
        while (!pq.empty()){
                auto [w, u] = pq.top(); pq.pop();
                w = -w; u = -u;
                if (tomado[u]) continue;
                mst_costo += w;
                process(u);
                tomados++;
                if (tomados == n-1) break;
        }
        return mst_costo;
}
```

6.18 Puentes y Puntos

```
const int maxn = 1e5+5;
vector<bool> vis;
vi adj[maxn]; // undirected
vi tin, low;
int timer;

void dfs(int u,int p=-1){
```

```
		vis[u]=true;
		tin[u]=low[u]=timer++;
		int children=0;
		for(int v:adj[u]){
			if(v==p)continue;
			if(vis[v])low[u]=min(low[u],tin[v]);
			else{
				dfs(v,u);
				low[u]=min(low[u], low[v]);
				if(low[v]>tin[u]); // u-v puente
				if(low[v]>=tin[u] && p!=-1); // u
				    punto de articulacion
				++children;
			}
		}
		if(p==-1 && children>1); // u punto de
		    articulacion
	}
	// O(n+m)
	void init(int n){
		timer=0;
		vis.assign(n,false);
		tin.assign(n,-1);low.assign(n,-1);
		for(int i=0;i<n;++i){
			if(!vis[i])dfs(i);
		}
	}
```

6.19 Shortest Path Faster Algorithm

```
	//Algoritmo mas rapido de ruta minima
	//O(V*E) peor caso, O(E) en promedio.
	bool spfa(vector<vii> &adj, vector<int> &d, int s, int n)
	    {
		d.assign(n, INF);
		vector<int> cnt(n, 0);
		vector<bool> inqueue(n, false);
		queue<int> q;

		d[s] = 0;
		q.push(s);
		inqueue[s] = true;
		while (!q.empty()) {
			int v = q.front();
			q.pop();
			inqueue[v] = false;

			for (auto& [to, len] : adj[v]) {

				if (d[v] + len < d[to]) {
					d[to] = d[v] + len;
					if (!inqueue[to]) {
						q.push(to);
						inqueue[to] =
						    true;
						cnt[to]++;
						if (cnt[to] > n)
							return
							    false;
							    //
							    ciclo
							    negativo
					}
				}
			}
		}
		return true;
	}
```

6.20 Tarjan

```
	// O(n+m) build graph in g[] and callt()
	const int maxn = 2e5 + 5;
	vi low, num, comp, g[maxn];
	int scc, timer;
	stack<int> st;
	void tjn(int u) {
		low[u] = num[u] = timer++; st.push(u); int v;
		for(int v: g[u]) {
			if(num[v]==-1) tjn(v);
			if(comp[v]==-1) low[u] = min(low[u], low[
			    v]);
		}
		if(low[u]==num[u]) {
			do{ v = st.top(); st.pop(); comp[v]=scc;
			}while(u != v);
			++scc;
		}
	}
	void callt(int n) {
		timer = scc= 0;
		num = low = comp = vector<int>(n,-1);
		for(int i = 0; i<n; i++) if(num[i]==-1) tjn(i);
	}
```

7 Matematicas

7.1 Bruijn sequences

```
	// Given alphabet [0, k) constructs a cyclic string
	// of length k^n that contains every length n string as
	    substr.
	vi deBruijnSeq(int k, int n, int lim){
```

```
        if (k == 1) return {0};
        vi seq, aux(n + 1);
        int cont = 0;
        function<void(int,int)> gen = [&](int t, int p) {
                if (t > n){
                        if (n % p == 0) for(int i = 1; i
                            < p + 1; i++){
                                if (cont >= lim) return;
                                seq.push_back(aux[i]);
                                cont++;
                        }
                } else {
                        aux[t] = aux[t - p];
                        gen(t + 1, p);
                        while (++aux[t] < k){
                                if (cont >= lim) return;
                                gen(t + 1, t);
                        }
                }
        };
        gen(1, 1);
    // for (int i = 0; i<n-1; i++) seq.push_back(0);
        return seq;
}
```

7.2 Convoluciones

```
// c[k] = sumatoria (i&j = k, += a[i]*b[j]) AND
   convolution
// c[k] = sumatoria (i|j = k, += a[i]*b[j]) OR
   convolution
// c[k] = sumatoria (i^j = k, += a[i]*b[j]) XOR
   convolution
// c[k] = sumatoria (gcd(i,j) = k, += a[i]*b[j]) GCD
   convolution
// c[k] = sumatoria (lcm(i,j) = k, += a[i]*b[j]) LCM
   convolution
// todas las funciones tienen operaciones con modulo
// si es indexando en 1 entonces se pone un cero al
   principio y listo

template<int MOD> struct mint {
        static const int mod = MOD;
        int v;
        explicit operator int() const { return v; }
        mint():v(0) {}
        mint(ll _v):v(int(_v%MOD)) { v += (v<0)*MOD; }
        void build(ll _v) { v=int(_v%MOD), v+=(v<0)*MOD;
            }
        mint& operator+=(mint o) {
                if ((v += o.v) >= MOD) v -= MOD;
                return *this; }
        mint& operator-=(mint o) {
                if ((v -= o.v) < 0) v += MOD;
                return *this; }
        mint& operator*=(mint o) {
                v = int((ll)v*o.v%MOD); return *this; }
        friend mint pow(mint a, ll p) { assert(p >= 0);
                return p==0?1:pow(a*a,p/2)*(p&1?a:1); }
        friend mint inv(mint a) { assert(a.v != 0);
            return pow(a,MOD-2); }
        friend mint operator+(mint a, mint b) { return a
            += b; }
        friend mint operator-(mint a, mint b) { return a
            -= b; }
        friend mint operator*(mint a, mint b) { return a
            *= b; }
};
using mi = mint<998244353>;

template<typename T>
void SubsetZetaTransform(vector<T>& v) {
        const int n = v.size(); // n must be a power of 2
        for (int j = 1; j < n; j <<= 1) {
                for (int i = 0; i < n; i++)
                        if (i & j) v[i] += v[i ^ j];
        }
}

template<typename T>
void SubsetMobiusTransform(vector<T>& v) {
        const int n = v.size(); // n must be a power of 2
        for (int j = 1; j < n; j <<= 1) {
                for (int i = 0; i < n; i++)
                        if (i & j) v[i] -= v[i ^ j];
        }
}

template<typename T>
void SupersetZetaTransform(vector<T>& v) {
        const int n = v.size(); // n must be a power of 2
        for (int j = 1; j < n; j <<= 1) {
                for (int i = 0; i < n; i++)
                        if (i & j) v[i ^ j] += v[i];
        }
}

template<typename T>
void SupersetMobiusTransform(vector<T>& v) {
        const int n = v.size(); // n must be a power of 2
        for (int j = 1; j < n; j <<= 1) {
                for (int i = 0; i < n; i++)
                        if (i & j) v[i ^ j] -= v[i];
        }
}

vector<int> PrimeEnumerate(int n) {
        vector<int> P; vector<bool> B(n + 1, 1);
        for (int i = 2; i <= n; i++) {
                if (B[i]) P.push_back(i);
```

```
                for (int j : P) { if (i * j > n) break; B
                    [i * j] = 0; if (i % j == 0) break; }
        }
        return P;
}

template<typename T>
void DivisorZetaTransform(vector<T>& v) {
        const int n = sz(v) - 1;
        for (int p : PrimeEnumerate(n)) {
                for (int i = 1; i * p <= n; i++)
                        v[i * p] += v[i];
        }
}

template<typename T>
void DivisorMobiusTransform(vector<T>& v) {
        const int n = sz(v) - 1;
        for (int p : PrimeEnumerate(n)) {
                for (int i = n / p; i; i--)
                        v[i * p] -= v[i];
        }
}

template<typename T>
void MultipleZetaTransform(vector<T>& v) {
        const int n = sz(v) - 1;
        for (int p : PrimeEnumerate(n)) {
                for (int i = n / p; i; i--)
                        v[i] += v[i * p];
        }
}

template<typename T>
void MultipleMobiusTransform(vector<T>& v) {
        const int n = sz(v) - 1;
        for (int p : PrimeEnumerate(n)) {
                for (int i = 1; i * p <= n; i++)
                        v[i] -= v[i * p];
        }
}

template<typename T>
vector<T> AndConvolution(vector<T> A, vector<T> B) {
        SupersetZetaTransform(A);
        SupersetZetaTransform(B);
        for (int i = 0; i < sz(A); i++) A[i] *= B[i];
        SupersetMobiusTransform(A);
        return A;
}

template<typename T>
vector<T> OrConvolution(vector<T> A, vector<T> B) {
        SubsetZetaTransform(A);
        SubsetZetaTransform(B);
        for (int i = 0; i < sz(A); i++) A[i] *= B[i];
        SubsetMobiusTransform(A);
        return A;
}

template<typename T>
vector<T> GCDConvolution(vector<T> A, vector<T> B) {
        MultipleZetaTransform(A);
        MultipleZetaTransform(B);
        for (int i = 0; i < sz(A); i++) A[i] *= B[i];
        MultipleMobiusTransform(A);
        return A;
}

template<typename T>
vector<T> LCMConvolution(vector<T> A, vector<T> B) {
        DivisorZetaTransform(A);
        DivisorZetaTransform(B);
        for (int i = 0; i < sz(A); i++) A[i] *= B[i];
        DivisorMobiusTransform(A);
        return A;
}

template<typename T>
vector<T> XORConvolution(vector<T> A, vector<T> B) {
        const int n = sz(A);
        auto FWT = [&](vector<T>& v) {
                for (int len = 1; len < n; len <<= 1) {
                        for (int i = 0; i < n; i += (len
                            << 1)) {
                                for (int j = 0; j < len;
                                    j++) {
                                        T u(v[i + j]);
                                        T w(v[i + j + len
                                            ]);
                                        v[i + j] = u + w;
                                         v[i + j + len
                                            ] = u - w;
                                }
                        }
                }
        };
        FWT(A); FWT(B);
        for (int i = 0; i < n; i++) A[i] *= B[i];
        FWT(A);
        T inv_n(inv(T(n)));
        for (int i = 0; i < n; i++) A[i] *= inv_n;
        return A;
}

void main2(){
        int n;
        cin>>n;
        vector<mi> a(1<<n),b(1<<n);
        for(int x,i=0;i<sz(a);++i){cin>>x;a[i].build(x);}
        for(int x,i=0;i<sz(b);++i){cin>>x;b[i].build(x);}
        vector<mi> ans=XORConvolution(a,b);
        for(int i=0;i<sz(ans);++i)cout<<ans[i].v<<" ";
```

```
}
```

7.3 Criba

```
// O(n*log(log(n)))
vector<ll> primes;
vector<bool> is_prime;
void criba(ll n){
        is_prime.assign(n+1,true);
        for(ll i=2;i<=n;++i){
                if(!is_prime[i])continue;
                for(ll j=i*i;j<=n;j+=i)is_prime[j]=false;
                primes.push_back(i);
        }
}

// O(sqrt(n)/log(sqrt(n)))
void fact(ll n, map<ll, int>& f){
        for(int i=0;i<sz(primes) && primes[i]*primes[i]<=
            n;++i)
                while(n%primes[i]==0ll)f[primes[i]]++,n/=
                    primes[i];
        if(n>1)f[n]++;
}

// O((R-L+1)log(log(R))+sqrt(R)log(log(sqrt(R)))
// R-L+1 <= 1e7, R <= 1e14
void segmentedSieve(long long L, long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R)+3;
    vector<bool> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }

    vector<bool> isPrime(R - L + 1, true);
    for (long long i : primes)
        for (long long j = max(i * i, (L + i - 1) / i * i
            ); j <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
}
```

7.4 Chinese Remainder Theorem

```
/// Complexity: |N|*log(|N|)
/// Tested: Not yet.
/// finds a suitable x that meets: x is congruent to a_i
    mod n_i
/** Works for non-coprime moduli.
 Returns {-1,-1} if solution does not exist or input is
    invalid.
Otherwise, returns {x,L}, where x is the solution unique
    to mod L = LCM of mods
*/
pll crt( vl A, vl M ) {
        ll n = A.size(), a1 = A[0], m1 = M[0];
        for(ll i = 1; i < n; i++) {
                ll a2 = A[i], m2 = M[i];
                ll g = __gcd(m1, m2);
                if( a1 % g != a2 % g ) return {-1,-1};
                ll p, q;
                extended_euclid(m1/g, m2/g, p, q);
                ll mod = m1 / g * m2;
                q %= mod; p %= mod;
                ll x = ((1ll*(a1%mod)*(m2/g))%mod*q + (1
                    ll*(a2%mod)*(m1/g))%mod*p) % mod; //
                    if WA there is overflow
                a1 = x;
                if (a1 < 0) a1 += mod;
                m1 = mod;
        }
        return {a1, m1};
}
```

7.5 Divisors

```
// d(n) = (a1+1)*(a2+1)*...*(ak+1)
ll numDiv(map<ll, ll>& f){
        ll ans=1;
        for(auto [_,pot]:f)ans=mul(ans, (pot+1ll));
        return ans;
}

// sigma(n) = (p1^(a1+1)-1)/(p1-1) * (p2^(a2+1)-1)/(p2-1)
    * ... * (pk^(ak+1)-1)/(pk-1)
// suma divisores a la xth potencia
ll sumDiv(map<ll, ll>& f){
        ll ans=1,potencia=1;
        for(auto [num, pot]:f){
                ll p=binpow(num,(pot+1ll)*potencia)-1ll;
                ans=mul(ans, mul(p, inv(num-1ll)));
        }
        return ans;
}

ll productDiv(map<ll, ll>& f){
        ll pi=1,res=1;
        for(auto [num, pot]:f){
                ll p=binpow(num, pot*(pot+1ll)/2ll);
```

```
                res=mul(binpow(res, pot+1ll),binpow(p, pi
                    ));
                pi=mul(pi, pot+1ll, MOD-1ll);
        }
        return res;
}

// si a y b son coprimos, entonces:
// sigma(a*b) = sigma(a)*sigma(b)
// d(a*b) = d(a)*d(b)
```

7.6 Ecuaciones Diofanticas

```
// O(log(n))
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
        ll xx = y = 0;
        ll yy = x = 1;
        while (b) {
                ll q = a / b;
                ll t = b; b = a % b; a = t;
                t = xx; xx = x - q * xx; x = t;
                t = yy; yy = y - q * yy; y = t;
        }
        return a;
}
// a*x+b*y=c. returns valid x and y if possible.
// all solutions are of the form (x0 + k * b / g, y0 - k
    * b / g)
bool find_any_solution (ll a, ll b, ll c, ll &x0, ll &y0,
     ll &g) {
        if (a == 0 and b == 0) {
                if (c) return false;
                x0 = y0 = g = 0;
                return true;
        }
        g = extended_euclid (abs(a), abs(b), x0, y0);
        if (c % g != 0) return false;
        x0 *= c / g;
        y0 *= c / g;
        if (a < 0) x0 *= -1;
        if (b < 0) y0 *= -1;
        return true;
}
void shift_solution(ll &x, ll &y, ll a, ll b, ll cnt) {
        x += cnt * b;
        y -= cnt * a;
}
// returns the number of solutions where x is in the
    range[minx, maxx] and y is in the range[miny, maxy]
ll find_all_solutions(ll a, ll b, ll c, ll minx, ll maxx,
     ll miny,ll maxy) {
        ll x, y, g;
        if (find_any_solution(a, b, c, x, y, g) == 0)
            return 0;
        if (a == 0 and b == 0) {
                assert(c == 0);
                return 1LL * (maxx - minx + 1) * (maxy -
                    miny + 1);
        }
        if (a == 0) {
                return (maxx - minx + 1) * (miny <= c / b
                     and c / b <= maxy);
        }
        if (b == 0) {
                return (maxy - miny + 1) * (minx <= c / a
                     and c / a <= maxx);
        }
        a /= g, b /= g;
        ll sign_a = a > 0 ? +1 : -1;
        ll sign_b = b > 0 ? +1 : -1;
        shift_solution(x, y, a, b, (minx - x) / b);
        if (x < minx) shift_solution(x, y, a, b, sign_b);
        if (x > maxx) return 0;
        ll lx1 = x;
        shift_solution(x, y, a, b, (maxx - x) / b);
        if (x > maxx) shift_solution (x, y, a, b, -sign_b
            );
        ll rx1 = x;
        shift_solution(x, y, a, b, -(miny - y) / a);
        if (y < miny) shift_solution (x, y, a, b, -sign_a
            );
        if (y > maxy) return 0;
        ll lx2 = x;
        shift_solution(x, y, a, b, -(maxy - y) / a);
        if (y > maxy) shift_solution(x, y, a, b, sign_a);
        ll rx2 = x;
        if (lx2 > rx2) swap (lx2, rx2);
        ll lx = max(lx1, lx2);
        ll rx = min(rx1, rx2);
        if (lx > rx) return 0;
        return (rx - lx) / abs(b) + 1;
}

///finds the first k | x + b * k / gcd(a, b) >= val
ll greater_or_equal_than(ll a, ll b, ll x, ll val, ll g)
    {
        ld got = 1.0 * (val - x) * g / b;
        return b > 0 ? ceil(got) : floor(got);
}
```

7.7 Exponenciacion binaria

```
ll binpow(ll b, ll n, ll m) {
        b %= m;
        ll res = 1;
        while (n > 0) {
                if (n & 1)
                        res = res * b % m;
                b = b * b % m;
```

```
            n >>= 1;
        }
        return res % m;
}
```

7.8 Exponenciacion matricial

```
struct matrix {
        int r, c; vector<vl> m;
        matrix(int r, int c, const vector<vl> &m) : r(r), c(c), m(m){}

        matrix operator * (const matrix &b){
                matrix ans(this->r, b.c, vector<vl>(this->r, vl(b.c, 0)));

                for (int i = 0; i<this->r; i++) {
                        for (int k = 0; k<b.r; k++){
                                if (m[i][k] == 0)
                                    continue;
                                for (int j = 0; j<b.c; j++){
                                        ans.m[i][j] += mod(m[i][k], MOD) * mod(b.m[k][j], MOD);
                                        ans.m[i][j] = mod(ans.m[i][j], MOD);
                                }
                        }
                }
                return ans;
        }
};

matrix pow(matrix &b, ll p){
        matrix ans(b.r, b.c, vector<vl>(b.r, vl(b.c, 0)));
        for (int i = 0; i<b.r; i++) ans.m[i][i] = 1;
        while (p){
                if (p&1){
                        ans = ans*b;
                }
                b = b*b;
                p >>= 1;
        }
        return ans;
}
```

7.9 Fast Fourier Transform

```
///Complexity: O(N log N)
///tested: https://codeforces.com/gym/104373/problem/E
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define sz(v) ((int)v.size())
#define trav(a, x) for(auto& a : x)
#define all(v) v.begin(),v.end()
typedef vector<ll> vl;
typedef vector<int> vi;
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
        int n = sz(a), L = 31 - __builtin_clz(n);
        static vector<complex<long double>> R(2, 1);
        static vector<C> rt(2, 1);  // (^ 10% faster if double)
        for (static int k = 2; k < n; k *= 2) {
                R.resize(n); rt.resize(n);
                auto x = polar(1.0L, acos(-1.0L) / k);
                rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
        }
        vi rev(n);
        rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
        rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
                for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
                        // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)  /// include-line
                        auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];        /// exclude-line
                        C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);           /// exclude-line
                        a[i + j + k] = a[i + j] - z;
                        a[i + j] += z;
                }
}
vl conv(const vl& a, const vl& b) {
        if (a.empty() || b.empty()) return {};
        vd res(sz(a) + sz(b) - 1);
        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n);
        copy(all(a), begin(in));
        rep(i,0,sz(b)) in[i].imag(b[i]);
        fft(in);
        trav(x, in) x *= x;
        rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
        fft(out);
        vector<ll> resp(sz(res));
        rep(i,0,sz(res)) resp[i] = round(imag(out[i]) /
```

```
		(4.0 * n));
	return resp;
}
```

7.10 Fibonacci Fast Doubling

```
// O(log n) muy rapido
// F(2n+1) = F(n)^2 + F(n+1)^2
// F(2n) = F(n+1)^2 - F(n-1)^2
pair<int, int> fib (int n) {
	if (n == 0)
		return {0, 1};

	auto p = fib(n >> 1);
	int c = p.first * (2 * p.second - p.first);
	int d = p.first * p.first + p.second * p.second;
	if (n & 1)
		return {d, c + d};
	else
		return {c, d};
}
```

7.11 Fraction

```
typedef __int128 T;
struct Fraction{
	T num,den;
	Fraction():num(0),den(1){}
	Fraction(T n):num(n),den(1){}
	Fraction(T n,T d):num(n),den(d){reduce();}
	void reduce(){
		// assert(den!=0);
		T gcd=__gcd(num,den); // <-
		num/=gcd,den/=gcd;
		if(den<0)num=-num,den=-den;
	}
	Fraction fractional_part()const{ // x - floor(x)
		Fraction fp=Fraction(num%den,den);
		if(fp<Fraction(0))fp+=Fraction(1);
		return fp;
	}
	T compare(Fraction f)const{return num*f.den-den*f
	    .num;}
	Fraction operator + (const Fraction& f){return
	    Fraction(num*f.den+den*f.num,den*f.den);}
	Fraction operator - (const Fraction& f){return
	    Fraction(num*f.den-den*f.num,den*f.den);}
	Fraction operator * (const Fraction& f){
		Fraction a=Fraction(num,f.den);
		Fraction b=Fraction(f.num,den);
		return Fraction(a.num*b.num,a.den*b.den);
	}
	Fraction operator / (const Fraction& f){return *
	    this*Fraction(f.den,f.num);}
	Fraction operator += (const Fraction& f){return *
	    this=*this+f;}
	Fraction operator -= (const Fraction& f){return *
	    this=*this-f;}
	Fraction operator *= (const Fraction& f){return *
	    this=*this*f;}
	Fraction operator /= (const Fraction& f){return *
	    this=*this/f;}
	bool operator == (const Fraction& f)const{return
	    compare(f)==0;}
	bool operator != (const Fraction& f)const{return
	    compare(f)!=0;}
	bool operator >= (const Fraction& f)const{return
	    compare(f)>=0;}
	bool operator <= (const Fraction& f)const{return
	    compare(f)<=0;}
	bool operator > (const Fraction& f)const{return
	    compare(f)>0;}
	bool operator < (const Fraction& f)const{return
	    compare(f)<0;}
};
Fraction operator - (const Fraction& f){return Fraction(-
    f.num,f.den);}
ostream& operator << (ostream& os, const Fraction& f){
    return os<<"("<<(ll)f.num<<"/"<<(ll)f.den<<")";}
```

7.12 Freivalds algorithm

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch
    ().count());
// check if two n*n matrix a*b=c within complexity (
    iteration*n^2)
// probability of error 2^(-iteration)
// O(iter*n^2)
int Freivalds(matrix &a, matrix &b, matrix &c) {
	int n = a.r, iteration = 20;
	matrix zero(n, 1), r(n, 1);
	while (iteration--) {
		for(int i = 0; i < n; i++) r.m[i][0] =
		    rnd() % 2;
		matrix ans = (a * (b * r)) - (c * r);
		if(ans.m != zero.m) return 0;
	}
	return 1;
}
```

7.13 Gauss Jordan

```
const int EPS = 1;
```

```
int gauss (vector<vector<int>> a, vector<int> &ans) {
  int n = a.size(), m = a[0].size()-1;
  vector<int> where(m, -1);
  for(int col = 0, row = 0; col < m && row < n; ++col) {
    int sel = row;
    for(int i = row; i < n; ++i)
      if(abs(a[i][col]) > abs(a[sel][col])) sel = i;
    if(abs(a[sel][col]) < EPS) continue;
    swap(a[sel], a[row]);
    where[col] = row;
    for(int i = 0; i < n; ++i)
      if(i != row) {
        int c = divide(a[i][col], a[row][col]); ///
            precalc inverses
        for(int j = col; j <= m; ++j)
          a[i][j] = sbt(a[i][j], mul(a[row][j], c));
      }
    ++row;
  }
  ans.assign(m, 0);
  for(int i = 0; i < m; ++i)
    if(where[i] != -1) ans[i] = divide(a[where[i]][m], a[
        where[i]][i]);
  for(int i = 0; i < n; ++i) {
    int sum = 0;
    for(int j = 0; j < m; ++j)
      sum = add(sum, mul(ans[j], a[i][j]));
    if(sum != a[i][m]) return 0;
  }
  for(int i = 0; i < m; ++i)
    if(where[i] == -1) return -1; /// infinite solutions
  return 1;
}
```

7.14 Gauss Jordan mod 2

```
// O(min(n, m)*n*m)
int gauss (vector < bitset<N> > &a, int n, int m, bitset<
    N> & ans) {
        vector<int> where (m, -1);
        for (int col=0, row=0; col<m && row<n; ++col) {
                for (int i=row; i<n; ++i)
                        if (a[i][col]) {
                                swap (a[i], a[row]);
                                break;
                        }
                if (! a[row][col])
                        continue;
                where[col] = row;

                for (int i=0; i<n; ++i)
                        if (i != row && a[i][col])
                                a[i] ^= a[row];
                ++row;
        }
        for (int i=0; i<m; ++i)
                if (where[i] != -1)
                        ans[i] = a[where[i]][m] / a[where
                            [i]][i];
        for (int i=0; i<n; ++i) {
                double sum = 0;
                for (int j=0; j<m; ++j)
                        sum += ans[j] * a[i][j];
                if (abs (sum - a[i][m]) > EPS)
                        return 0;
        }

        for (int i=0; i<m; ++i)
                if (where[i] == -1)
                        return INF;
        return 1;
}
```

7.15 GCD y LCM

```
//O(log10 n) n == max(a, b)
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a%b);
    }
int lcm(int a, int b) { return a / gcd(a, b) * b; }
//gcd(a, b, c) = gcd(a, gcd(b, c))
//gcd(a, b) = gcd(a, b-a)

// O(log(min(a, b)) - a*x+b*y=gcd(a,b)
ll gcd(ll a, ll b, ll &x, ll &y){
        x=1,y=0;
        ll x1=0,y1=1,a1=a,b1=b;
        while(b1){
                ll q=a1/b1;
                tie(x, x1)=make_tuple(x1, x-q*x1);
                tie(y, y1)=make_tuple(y1, y-q*y1);
                tie(a1, b1)=make_tuple(b1, a1-q*b1);
        }return a1;
}
```

7.16 Integral Definida

```
const int steps = 1e6; // %2==0
double f(double x);
double simpson(double a, double b){
        double h=(b-a)/steps;
        double s=f(a)+f(b);
        for(int i=1;i<=steps-1;i++){
                double x=a+h*i;
                s+=f(x)*((i&1)?4:2);
        }
```

```
		s*=h/3;
		return s;
}
```

7.17 Inverso modular

```
ll mod(ll a, ll m){
		return ((a%m) + m) % m;
}

ll modInverse(ll b, ll m){
		ll x, y;
		ll d = extEuclid(b, m, x, y);  //obtiene b*x + m*
		    y == d
		if (d != 1) return -1;         //indica error
		// b*x + m*y == 1, ahora aplicamos (mod m) para
		    obtener b*x == 1 (mod m)
		return mod(x, m);
}

// Otra forma
// O(log MOD)
ll inv (ll a){
		return binpow(a, MOD-2, MOD);
}

//Modulo constante
inv[1] = 1;
for(int i = 2; i < p; ++i)
		inv[i] = (p - (p / i) * inv[p % i] % p) % p;
```

7.18 Lagrange

```
const int N = 1e6;
int f[N], fr[N];
void initC(){
  f[0] = 1;
  for(int i=1; i<N; i++) f[i] = mul(f[i-1], i);
  fr[N-1] = inv(f[N-1]);
  for(int i=N-1; i>=1; --i) fr[i-1] = mul(fr[i], i);
}
// mint C(int n, int k) { return k<0 || k>n ? 0 : f[n] *
    fr[k] * fr[n-k]; }
struct LagrangePol {
  int n;
  vi y, den, l, r;
  LagrangePol(vector<int> f): n(sz(f)), y(f), den(n), l(n
      ), r(n){// f[i] := f(i)
    // Calcula interpol. pol P in O(n) := deg(P) = sz(v)
        -1
    initC();
    for (int i = 0; i<n; i++) {
      den[i] = mul(fr[n-1-i], fr[i]);
      if((n-1-i) & 1) den[i] = mod(-den[i]);
    }
  }
  int eval(int x){ // Evaluate LagrangePoly P(x) in O(n)
    l[0] = r[n-1] = 1;
    for (int i = 1; i<n; i++) l[i] = mul(l[i-1], mod(x -
        i + 1));
    for(int i=n-2; i>=0; --i) r[i] = mul(r[i+1], mod(x -
        i - 1));
    int ans = 0;
    for (int i = 0; i<n; i++) ans = add(ans, mul(mul(l[i
        ], r[i]), mul(y[i], den[i])));
    return ans;
  }
};
/////////////////////////////////////////
// Para Xs que no sean de [0, N]
/// Complexity: O(|N|^2)
/// Tested: https://tinyurl.com/y23sh38k
vector<lf> X, F;
lf f(lf x) {
  lf answer = 0;
  for(int i = 0; i < sz(X); i++) {
    lf prod = F[i];
    for(int j = 0; j < sz(X); j++) {
      if(i == j) continue;
      prod = mul(prod, divide(sbt(x, X[j]), sbt(X[i], X[j
          ])));
    }
    answer = add(answer, prod);
  }
  return answer;
}

//given y=f(x) for x [0,degree]
vi interpolation( vi &y ) {
  int n = sz(y);
  vi u = y, ans( n ), sum( n );
  ans[0] = u[0], sum[0] = 1;
  for( int i = 1; i < n; ++i )
  {
    int inv = binpow( i, MOD - 2 );
    for( int j = n - 1; j >= i; --j )
      u[j] = 1LL * (u[j] - u[j - 1] + MOD) * inv % MOD;

    for( int j = i; j > 0; --j )
    {
      sum[j] = (sum[j - 1] - 1LL * (i - 1) * sum[j] % MOD
          + MOD) % MOD;
      ans[j] = (ans[j] + 1LL * sum[j] * u[i]) % MOD;
    }
    sum[0] = 1LL * (i - 1) * (MOD - sum[0]) % MOD;
    ans[0] = (ans[0] + 1LL * sum[0] * u[i]) % MOD;
  }
  return ans;
```

```
}
```

7.19 Logaritmo Discreto

```
// O(sqrt(m))
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
        // if (a == 0) return b == 0 ? 1 : -1; Casos 0^x
            = b
        a %= m, b %= m;
        int k = 1, add = 0, g;
        while ((g = gcd(a, m)) > 1) {
                if (b == k)
                        return add;
                if (b % g)
                        return -1;
                b /= g, m /= g, ++add;
                k = (k * 1ll * a / g) % m;
        }

        int n = sqrt(m) + 1;
        int an = 1;
        for (int i = 0; i < n; ++i)
                an = (an * 1ll * a) % m;

        unordered_map<int, int> vals;
        for (int q = 0, cur = b; q <= n; ++q) {
                vals[cur] = q;
                cur = (cur * 1ll * a) % m;
        }

        for (int p = 1, cur = k; p <= n; ++p) {
                cur = (cur * 1ll * an) % m;
                if (vals.count(cur)) {
                        int ans = n * p - vals[cur] + add
                            ;
                        return ans;
                }
        }
        return -1;
}
```

7.20 Miller Rabin

```
ll mul (ll a, ll b, ll mod) {
        ll ret = 0;
        for(a %= mod, b %= mod; b != 0;
                b >>= 1, a <<= 1, a = a >= mod ? a - mod
                    : a) {
                if (b & 1) {
                        ret += a;
                        if (ret >= mod) ret -= mod;
                }
        }
        return ret;
}
ll fpow (ll a, ll b, ll mod) {
        ll ans = 1;
        for (; b; b >>= 1, a = mul(a, a, mod))
                if (b & 1)
                        ans = mul(ans, a, mod);
        return ans;
}
bool witness (ll a, ll s, ll d, ll n) {
        ll x = fpow(a, d, n);
        if (x == 1 || x == n - 1) return false;
        for (int i = 0; i < s - 1; i++) {
                x = mul(x, x, n);
                if (x == 1) return true;
                if (x == n - 1) return false;
        }
        return true;
}
ll test[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 0};
bool is_prime (ll n) {
        if (n < 2) return false;
        if (n == 2) return true;
        if (n % 2 == 0) return false;
        ll d = n - 1, s = 0;
        while (d % 2 == 0) ++s, d /= 2;
        for (int i = 0; test[i] && test[i] < n; ++i)
                if (witness(test[i], s, d, n))
                        return false;
        return true;
}
```

7.21 Miller Rabin Probabilistico

```
using u64 = uint64_t;
using u128 = __uint128_t;

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

bool check_composite(u64 n, u64 a, u64 d, int s) {
        u64 x = binpower(a, d, n);
        if (x == 1 || x == n - 1)
                return false;
```

```cpp
		for (int r = 1; r < s; r++) {
			x = (u128)x * x % n;
			if (x == n - 1)
				return false;
		}
		return true;
};
bool MillerRabin(u64 n, int iter=5) { // returns true if
	n is probably prime, else returns false.
		if (n < 4)
			return n == 2 || n == 3;

		int s = 0;
		u64 d = n - 1;
		while ((d & 1) == 0) {
			d >>= 1;
			s++;
		}

		for (int i = 0; i < iter; i++) {
			int a = 2 + rand() % (n - 3);
			if (check_composite(n, a, d, s))
				return false;
		}
		return true;
}
```

7.22 Mobius

```cpp
// 1 if n is 1
// 0 if n has a squared prime factor
// (-1)^k if n is a product of k distinct prime factors
const int N = 1e6+1;
int mob[N];
void mobius() {
		mob[1] = 1;
		for (int i = 2; i < N; i++){
			mob[i]--;
			for (int j = i + i; j < N; j += i) {
				mob[j] -= mob[i];
			}
		}
}

// to count coprime pairs
// ans=n*(n-1)/2
// for(int x:a){
//			for(int y:divisors[a])cnt[y]++;
// }
// ans+=(mobius[v]*cnt[v]*(cnt[v]-1))/2
```

7.23 Number Theoretic Transform

```cpp
const int N = 1 << 20;
const int mod = 469762049; //998244353
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
int POW(int x, int y, int ans = 1) {
		for (; y; y >>= 1, x = (long long) x * x % mod)
			if (y & 1) ans = (long long) ans * x % mod;
		return ans;
}
void precompute(int len) {
		lim = wn[0] = 1; int s = -1;
		while (lim < len) lim <<= 1, ++s;
		for (int i = 0; i < lim; ++i) rev[i] = rev[i >>
			1] >> 1 | (i & 1) << s;
		const int g = POW(root, (mod - 1) / lim);
		inv_lim = POW(lim, mod - 2);
		for (int i = 1; i < lim; ++i) wn[i] = (long long)
			wn[i - 1] * g % mod;
}
void ntt(vector<int> &a, int typ) {
		for (int i = 0; i < lim; ++i) if (i < rev[i])
			swap(a[i], a[rev[i]]);
		for (int i = 1; i < lim; i <<= 1) {
			for (int j = 0, t = lim / i / 2; j < i;
				++j) w[j] = wn[j * t];
			for (int j = 0; j < lim; j += i << 1) {
				for (int k = 0; k < i; ++k) {
					const int x = a[k + j], y
						= (long long) a[k + j
						+ i] * w[k] % mod;
					reduce(a[k + j] += y -
						mod), reduce(a[k + j +
						i] = x - y);
				}
			}
		}
		if (!typ) {
			reverse(a.begin() + 1, a.begin() + lim);
			for (int i = 0; i < lim; ++i) a[i] = (
				long long) a[i] * inv_lim % mod;
		}
}
vector<int> multiply(vector<int> &f, vector<int> &g) {
		int n=(int)f.size() + (int)g.size() - 1;
		precompute(n);
		vector<int> a = f, b = g;
		a.resize(lim); b.resize(lim);
		ntt(a, 1), ntt(b, 1);
		for (int i = 0; i < lim; ++i) a[i] = (long long)
			a[i] * b[i] % mod;
		ntt(a, 0);
		a.resize(n + 1);
		return a;
```

```
}
```

7.24 Pollard Rho

```
//O(n^(1/4)) (?)
ll pollard_rho(ll n, ll c) {
        ll x = 2, y = 2, i = 1, k = 2, d;
        while (true) {
                x = (mul(x, x, n) + c);
                if (x >= n)     x -= n;
                d = __gcd(x - y, n);
                if (d > 1) return d;
                if (++i == k) y = x, k <<= 1;
        }
        return n;
}
void factorize(ll n, vector<ll> &f) {
        if (n == 1) return;
        if (is_prime(n)) {
                f.push_back(n);
                return;
        }
        ll d = n;
        for (int i = 2; d == n; i++)
                d = pollard_rho(n, i);
        factorize(d, f);
        factorize(n/d, f);
}
```

7.25 Simplex

```
// Maximizar c1*x1 + c2*x2 + c3*x3 ...
// Restricciones a11*x1 + a12*x2 <= b1, a22*x2 + a23*x3
    <= b2 ...
// Retorna valor optimo y valores de las variables
// O(c^2*b), O(c*b) - variables c, restricciones b
typedef double lf;
const lf EPS = 1e-9;
struct Simplex{
        vector<vector<lf>> A;
        vector<lf> B,C;
        vector<int> X,Y;
        lf z;
        int n,m;

        Simplex(vector<vector<lf>> _a, vector<lf> _b,
            vector<lf> _c){
                A=_a;B=_b;C=_c;
                n=B.size();m=C.size();z=0.;
                X=vector<int>(m);Y=vector<int>(n);
                for(int i=0;i<m;++i)X[i]=i;
                for(int i=0;i<n;++i)Y[i]=i+m;
        }
        void pivot(int x,int y){
                swap(X[y],Y[x]);
                B[x]/=A[x][y];
                for(int i=0;i<m;++i)if(i!=y)A[x][i]/=A[x
                    ][y];
                A[x][y]=1/A[x][y];
                for(int i=0;i<n;++i)if(i!=x&&abs(A[i][y])
                    >EPS){
                        B[i]-=A[i][y]*B[x];
                        for(int j=0;j<m;++j)if(j!=y)A[i][
                            j]-=A[i][y]*A[x][j];
                        A[i][y]=-A[i][y]*A[x][y];
                }
                z+=C[y]*B[x];
                for(int i=0;i<m;++i)if(i!=y)C[i]-=C[y]*A[
                    x][i];
                C[y]=-C[y]*A[x][y];
        }

        pair<lf, vector<lf>> maximize(){
                while(1){
                        int x=-1,y=-1;
                        lf mn=-EPS;
                        for(int i=0;i<n;++i)if(B[i]<mn)mn
                            =B[i],x=i;
                        if(x<0)break;
                        for(int i=0;i<m;++i)if(A[x][i]<-
                            EPS){y=i;break;}
                        // assert(y>=0) -> y<0, no
                            solution to Ax<=B
                        pivot(x,y);
                }
                while(1){
                        lf mx=EPS;
                        int x=-1,y=-1;
                        for(int i=0;i<m;++i)if(C[i]>mx)mx
                            =C[i],y=i;
                        if(y<0)break;
                        lf mn=1e200;
                        for(int i=0;i<n;++i)if(A[i][y]>
                            EPS&&B[i]/A[i][y]<mn)mn=B[i]/A
                            [i][y],x=i;
                        // assert(x>=0) -> x<0, unbounded
                        pivot(x,y);
                }
                vector<lf> r(m);
                for(int i=0;i<n;++i)if(Y[i]<m)r[Y[i]]=B[i
                    ];
                return {z,r};
        }
};
```

7.26 Simplex Int

```
// Maximizar c1*x1 + c2*x2 + c3*x3 ...
// Restricciones a11*x1 + a12*x2 <= b1, a22*x2 + a23*x3
    <= b2 ...
// Retorna valor optimo y valores de las variables

// O(c^2*b), O(c*b) - variables c, restricciones b (tle)
struct Fraction{};
typedef Fraction lf;
const lf ZERO(0),INF(1e18);
struct Simplex{
        vector<vector<lf>> A;
        vector<lf> B,C;
        vector<int> X,Y;
        lf z;
        int n,m;

        Simplex(vector<vector<lf>> _a, vector<lf> _b,
            vector<lf> _c){
                A=_a;B=_b;C=_c;
                n=B.size();m=C.size();z=ZERO;
                X=vector<int>(m);Y=vector<int>(n);
                for(int i=0;i<m;++i)X[i]=i;
                for(int i=0;i<n;++i)Y[i]=i+m;
        }

        void pivot(int x,int y){
                swap(X[y],Y[x]);
                B[x]/=A[x][y];
                for(int i=0;i<m;++i)if(i!=y)A[x][i]/=A[x
                    ][y];
                A[x][y]=Fraction(1)/A[x][y];
                for(int i=0;i<n;++i)if(i!=x && A[i][y]!=
                    ZERO){
                        B[i]-=A[i][y]*B[x];
                        for(int j=0;j<m;++j)if(j!=y)A[i][
                            j]-=A[i][y]*A[x][j];
                        A[i][y]=-A[i][y]*A[x][y];
                }
                z+=C[y]*B[x];
                for(int i=0;i<m;++i)if(i!=y)C[i]-=C[y]*A[
                    x][i];
                C[y]=-C[y]*A[x][y];
        }

        pair<lf, vector<lf>> maximize(){
                while(1){
                        int x=-1,y=-1;
                        lf mn=ZERO;
                        for(int i=0;i<n;++i)if(B[i]<mn)mn
                            =B[i],x=i;
                        if(x<0)break;
                        for(int i=0;i<m;++i)if(A[x][i]<
                            ZERO){y=i;break;}
                        // assert(y>=0) -> y<0, no
                            solution to Ax<=B
                        pivot(x,y);
                }
                while(1){
                        lf mx=ZERO;
                        int x=-1,y=-1;
                        for(int i=0;i<m;++i)if(C[i]>mx)mx
                            =C[i],y=i;
                        if(y<0)break;
                        lf mn=INF;
                        for(int i=0;i<n;++i)if(A[i][y]>
                            ZERO && B[i]/A[i][y]<mn)mn=B[i
                            ]/A[i][y],x=i;
                        // assert(x>=0) -> x<0, unbounded
                        pivot(x,y);
                }
                vector<lf> r(m);
                for(int i=0;i<n;++i)if(Y[i]<m)r[Y[i]]=B[i
                    ];
                return {z,r};
        }

        pair<Fraction, vector<Fraction>> maximize_int(){
                while(1){
                        auto sol=maximize();
                        bool all_int=true;
                        for(auto &x:sol.second)all_int&=x
                            .fractional_part()==ZERO;
                        if(all_int)return sol;
                        Fraction nw_b=ZERO;
                        int id=-1;
                        for(int i=0;i<n;++i){
                                Fraction fp=B[i].
                                    fractional_part();
                                if(fp>=nw_b)nw_b=fp,id=i;
                        }
                        vector<Fraction> nw_a;
                        for(auto &x:A[id])nw_a.push_back
                            (-x.fractional_part());
                        A.push_back(nw_a);
                        B.push_back(-nw_b);
                        Y.push_back(n+m);n++;
                }
        }
};
```

7.27 Totient y Divisores

```
vector<int> count_divisors_sieve() {
        bitset<mx> is_prime; is_prime.set();
        vector<int> cnt(mx, 1);
        is_prime[0] = is_prime[1] = 0;
        for(int i = 2; i < mx; i++) {
```

```
                if(!is_prime[i]) continue;
                cnt[i]++;
                for(int j = i+i; j < mx; j += i) {
                        int n = j, c = 1;
                        while( n%i == 0 ) n /= i, c++;
                        cnt[j] *= c;
                        is_prime[j] = 0;
                }
        }
        return cnt;
}
vector<int> euler_phi_sieve() {
        bitset<mx> is_prime; is_prime.set();
        vector<int> phi(mx);
        iota(phi.begin(), phi.end(), 0);
        is_prime[0] = is_prime[1] = 0;
        for(int i = 2; i < mx; i++) {
                if(!is_prime[i]) continue;
                for(int j = i; j < mx; j += i) {
                        phi[j] -= phi[j]/i;
                        is_prime[j] = 0;
                }
        }
        return phi;
}
ll euler_phi(ll n) {
        ll ans = n;
        for(ll i = 2; i * i <= n; ++i) {
                if(n % i == 0) {
                        ans -= ans / i;
                        while(n % i == 0) n /= i;
                }
        }
        if(n > 1) ans -= ans / n;
        return ans;
}
```

7.28 Xor Basis

```
template<typename T = int, int B = 31>
struct Basis {
        T a[B];
        Basis() {
                memset(a, 0, sizeof a);
        }
        void insert(T x){
                for (int i = B - 1; i >= 0; i--) {
                        if (x >> i & 1) {
                                if (a[i]) x ^= a[i];
                                else {
                                        a[i] = x;
                                        break;
                                }
                        }
                }
        }
        bool can(T x) {
                for(int i = B - 1; i >= 0; i--) {
                        x = min(x, x ^ a[i]);
                }
                return x == 0;
        }
        T max_xor(T ans = 0) {
                for(int i = B - 1; i >= 0; i--) {
                        ans = max(ans, ans ^ a[i]);
                }
                return ans;
        }
};

// Basis<long long, 63> B;
// Cantidad de xor diferentes es 2^sz(base)
// Cantidad de subsets xor = 0 es 2^(n-sz(base))
```

8 Programacion dinamica

8.1 Aliens Trick

```
// We binary search for A and find the highest A such
   that c(A) >= K.
// Let the optimal value be Aopt. Then our answer is v(
   Aopt) + AoptK

int main() {
        int n, k;
        cin >> n >> k;

        int a[n];
        for (int &i : a) { cin >> i; }

        /**
         * the maximum sum along with the number of
             subarrays used
         * if creating a subarray penalizes the sum by "
             lmb" and
         * there is no limit to the number of subarrays
             you can create
         */
        auto solve_lambda = [&](ll lmb) {
                pair<ll, ll> dp[n][2];

                dp[0][0] = {0, 0};
                dp[0][1] = {a[0] - lmb, 1};

                for (int i = 1; i < n; i++) {
                        dp[i][0] = max(dp[i - 1][0], dp[i
                             - 1][1]);

                        dp[i][1] =
```

```
                                max(make_pair(dp[i - 1][0].
                                    first + a[i] - lmb, dp[i -
                                    1][0].second + 1),
                                    make_pair(dp[i - 1][1].
                                        first + a[i], dp[i -
                                        1][1].second));
                }

                return max(dp[n - 1][0], dp[n - 1][1]);
        };

        ll lo = 0;
        ll hi = 1e18;
        while (lo < hi) {
                ll mid = (lo + hi + 1) / 2;
                solve_lambda(mid).second >= k ? lo = mid
                    : hi = mid - 1;
        }

        cout << solve_lambda(lo).first + lo * k << endl;
}
```

8.2 Bin Packing

```
int main() {
        ll n, capacidad;
        cin >> n >> capacidad;
        vl pesos(n, 0);
        forx(i, n) cin >> pesos[i];

        vector<pll> dp((1 << n));
        dp[0] = {1, 0};
        // dp[X] = {#numero de paquetes, peso de min
            paquete}

        // La idea es probar todos los subset y en cada
            uno preguntarnos
        // quien es mejor para subirse de ultimo buscando
             minimizar
        // primero el numero de paquetes
        for (int subset = 1; subset < (1 << n); subset++)
             {
                dp[subset] = {21, 0};

                for (int iPer = 0; iPer < n; iPer++) {
                        if ((subset >> iPer) & 1) {
                                pll ant = dp[subset ^ (1
                                    << iPer)];
                                ll k = ant.ff;
                                ll w = ant.ss;

                                if (w + pesos[iPer] >
                                    capacidad) {
                                        k++;
                                        w = min(pesos[
                                            iPer], w);
                                } else {
                                        w += pesos[iPer];
                                }

                                dp[subset] = min(dp[
                                    subset], {k, w});
                        }
                }
        }

        cout << dp[(1 << n) - 1].ff << ln;
}
```

8.3 Convex Hull Trick

```
// - Me dan las pendientes ordenadas
// Caso 1: Me hacen las querys ordenadas
// O(N + Q)
// Caso 2: Me hacen querys arbitrarias
// O(N + QlogN)

struct CHT {
    // funciona tanto para min como para max, depende del
         orden en que pasamos las lineas

    struct Line {
        int slope, yIntercept;

        Line(int slope, int yIntercept) : slope(slope),
            yIntercept(yIntercept){}
        int val(int x){ return slope * x + yIntercept; }
        int intersect(Line y) {
            return (y.yIntercept - yIntercept + slope - y
                .slope - 1) / (slope - y.slope);
        }
    };

    deque<pair<Line, int>> dq;

    void insert(int slope, int yIntercept){
        // lower hull si m1 < m2 < m3
        // upper hull si si m1 > m2 > m3
        Line newLine(slope, yIntercept);
        while (!dq.empty() && dq.back().second >= dq.back
            ().first.intersect(newLine)) dq.pop_back();
        if (dq.empty()) {
            dq.emplace_back(newLine, 0);
            return;
        }
        dq.emplace_back(newLine, dq.back().first.
            intersect(newLine));
    }

    int query(int x) { // cuando las consultas son
        crecientes
        while (dq.size() > 1) {
```

```
                if (dq[1].second <= x) dq.pop_front();
                else break;
            }
            return dq[0].first.val(x);
        }
        int query2(int x) { // cuando son arbitrarias
            auto qry = *lower_bound(dq.rbegin(), dq.rend(),
                                    make_pair(Line(0, 0), x),
                                    [&](const pair<Line, int>
                                        &a, const pair<Line,
                                        int> &b) {
                                        return a.second > b.
                                            second;
                                    });
            return qry.first.val(x);
        }
    };
```

8.4 CHT Dynamic

```
// O((N+Q) log N) <- usando set para add y bs para q
// lineas de la forma mx + b
#pragma once

struct Line {
        mutable ll m, b, p;
        bool operator<(const Line& o) const { return m <
            o.m; }
        bool operator<(ll x) const { return p < x; }
};

struct CHT : multiset<Line, less<>> {
        // (for doubles, use inf = 1/.0, div(a,b) = a/b)
        static const ll inf = LLONG_MAX;
        static const bool mini = 0; // <---- 1 FOR MIN
        ll div(ll a, ll b){ // floored division
                return a / b - ((a ^ b) < 0 && a % b); }
        bool isect(iterator x, iterator y){
                if (y == end()) return x->p = inf, 0;
                if (x->m == y->m) x->p = x->b > y->b ?
                    inf : -inf;
                else x->p = div(y->b - x->b, x->m - y->m)
                    ;
                return x->p >= y->p;
        }
        void add(ll m, ll b){
                if (mini){ m *= -1, b *= -1; }
                auto z = insert({m, b, 0}), y = z++, x =
                    y;
                while (isect(y, z)) z = erase(z);
                if (x != begin() && isect(--x, y)) isect(
                    x, y = erase(y));
                while ((y = x) != begin() && (--x)->p >=
                    y->p)
                        isect(x, erase(y));
        }
        ll query(ll x) {
                assert(!empty());
                auto l = *lower_bound(x);
                if (mini) return -l.m * x + -l.b;
                else return l.m * x + l.b;
        }
};
```

8.5 Digit DP

```
// dp[pos][count of d][limit]
ll dp[20][20][2];
int k,d;

// count numbers <= c with k occurrences of d
ll dfs(string& c, int x=0, int y=0, bool z=0){
        if(dp[x][y][z]!=-1)return dp[x][y][z];
        dp[x][y][z]=(y==k);
        if(x==(int)c.size()){
                return dp[x][y][z];
        }
        int limit=9;
        if(!z){
                limit=c[x]-'0';
        }
        dp[x][y][z]=0;
        for(int i=0;i<=limit;++i){
                if(z)dp[x][y][z]+=dfs(c, x+1, y+(i==d), z
                    );
                else dp[x][y][z]+=dfs(c, x+1, y+(i==d), i
                    <limit);
        }
        return dp[x][y][z];
}

// count(0,m) - count(0,n-1) = count(n,m)
ll query(ll n, ll m){
        string s1=to_string(m);
        string s2=to_string(n-1ll);
        memset(dp, -1, sizeof(dp));
        ll ans=dfs(s1);
        if(n<=0ll)return ans; // check
        memset(dp, -1, sizeof(dp));
        return ans-dfs(s2);
}
```

8.6 Divide Conquer

```
// C[a][c] + C[b][d] <= C[a][d] + C[b][c] where a < b < c
    < d.
int m, n;
```

```
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
        if (l > r)
                return;

        int mid = (l + r) >> 1;
        pair<long long, int> best = {LLONG_MAX, -1};

        for (int k = optl; k <= min(mid, optr); k++) {
                best = min(best, {(k ? dp_before[k - 1] :
                     0) + C(k, mid), k});
        }

        dp_cur[mid] = best.first;
        int opt = best.second;

        compute(l, mid - 1, optl, opt);
        compute(mid + 1, r, opt, optr);
}

int solve() {
        for (int i = 0; i < n; i++)
                dp_before[i] = C(0, i);

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

        return dp_before[n - 1];
}
```

8.7 Edit Distances

```
int editDistances(string& wor1,string& wor2){
        // O(tam1*tam2)
        // minimo de letras que debemos insertar, elminar
            o reemplazar
        // de wor1 para obtener wor2
        ll tam1=wor1.size();
        ll tam2=wor2.size();
        vector<vl> dp(tam2+1,vl(tam1+1,0));
        for(int i=0;i<=tam1;i++)dp[0][i]=i;
        for(int i=0;i<=tam2;i++)dp[i][0]=i;
        dp[0][0]=0;
        for(int i=1;i<=tam2;i++){
                for(int j=1;j<=tam1;j++){
                        ll op1 = min(dp[i-1][j],dp[i][j
                            -1])+1;
                        // el minimo entre eliminar o
                            insertar
                        ll op2 = dp[i-1][j-1]; //
                            reemplazarlo
                        if(wor1[j-1]!=wor2[i-1])op2++;
                        // si el reemplazo tiene efecto o
                            quedo igual
                        dp[i][j]=min(op1,op2);
                }
        }
        return dp[tam2][tam1];
}
```

8.8 Kadane 2D

```
int main() {
        ll fil,col;cin>>fil>>col;
        vector<vl> grid(fil,vl(col,0));

// Algoritmo de Kadane/DP para suma maxima de una matriz
    2D en o(n^3)
        for(int i=0;i<fil;i++){
                for(int e=0;e<col;e++){
                        ll num;cin>>num;
                        if (e>0) grid[i][e]=num+grid[i][e
                            -1];
                        else grid[i][e]=num;
                }
        }

        ll maxGlobal = LONG_LONG_MIN;
        for(int l=0;l<col;l++){
                for(int r=l;r<col;r++){
                        ll maxLoc=0;
                        for(int row=0;row<fil;row++){
                                if (l>0) maxLoc+=grid[row
                                    ][r]-grid[row][l-1];
                                else maxLoc+=grid[row][r
                                    ];
                                if (maxLoc<0) maxLoc=0;
                                maxGlobal= max(maxGlobal,
                                    maxLoc);
                        }
                }
        }
}
```

8.9 Knuth

```
// C[b][c] <= C[a][d]
// C[a][c] + C[b][d] <= C[a][d] + C[b][c] where a < b < c
    < d.
int solve() {
        int N;
        ... // read N and input
        int dp[N][N], opt[N][N];
```

```
	auto C = [&](int i, int j) {
		... // Implement cost function C.
	};

	for (int i = 0; i < N; i++) {
		opt[i][i] = i;
		... // Initialize dp[i][i] according to
		    the problem
	}

	for (int i = N-2; i >= 0; i--) {
		for (int j = i+1; j < N; j++) {
			int mn = INT_MAX;
			int cost = C(i, j);
			for (int k = opt[i][j-1]; k <=
			    min(j-1, opt[i+1][j]); k++) {
				if (mn >= dp[i][k] + dp[k
				    +1][j] + cost) {
					opt[i][j] = k;
					mn = dp[i][k] +
					    dp[k+1][j] +
					    cost;
				}
			}
			dp[i][j] = mn;
		}
	}

	cout << dp[0][N-1] << endl;
}
```

8.10 LIS

```
// O(n*log(n))
// retorna los indices de un lis
// cambiar el tipo y revisar si permite iguales
typedef int T;
vi lis(vector<T>& a, bool equal){
	vi prev(sz(a));
	typedef pair<T, int> p;
	vector<p> res;
	for(int i=0;i<sz(a);++i){
		auto it=lower_bound(all(res), p{a[i],(
		    equal?i:0)});
		if(it==res.end())res.emplace_back(),it=
		    res.end()-1;
		*it={a[i],i};
		prev[i]=(it==res.begin())?0:(it-1)->
		    second;
	}
	int l=sz(res),act=res.back().second;
	vi ans(l);
	while(l--)ans[l]=act,act=prev[act];
	return ans;
}
```

8.11 SOS

```
const int bits = 23;
int dp[1<<bits];

// O(n*2^n)
void SOS(){
	for(int i = 0; i < (1 << bits); ++i) dp[i] = A[i
	    ];
	// top - down (informacion de las submascaras)
	for(int i = 0; i < bits; ++i){
		for(int s = 0; s < (1 << bits); ++s){
			if(s & (1 << i)){
				dp[s] += dp[s ^ (1 << i)
				    ];
			}
		}
	}
	// bottom - up (informacion de las supermascaras)
	for(int i = 0; i < bits; ++i){
		for(int s = (1 << bits) - 1; s >= 0; --s)
		    {
			if(s & (1 << i)){
				dp[s ^ (1 << i)] += dp[s
				    ];
			}
		}
	}
}

int dp2[1<<bits][bits+1];

// O(2^n*n^2)
void cnt(){
	vector<int> a;
	for(int x:a)dp2[x][0]++;
	// dp[s][c] = number of s^ai with c bits
	for(int i=0;i<bits;++i){
		for(int c=i;c>=0;--c){
			for(int s=0;s<(1<<bits);++s){
				dp2[s^(1<<i)][c+1]+=dp2[s
				    ][c];
			}
		}
	}
}
```

9 Strings

9.1 Aho Corasick

```cpp
// 1) init() trie and add() strings
// 2) build() aho-corasick
// 3) process the text
// 4) dfs to calculate dp

// suf: longest proper suffix that's also in the trie
// dad: closest suffix link that is terminal
// cnt: number of strings that end exactly at node v
const int maxn = 2e5+5;
const int alpha = 26;
vector<int> adj[maxn];
int to[maxn][alpha],cnt[maxn],dad[maxn],suf[maxn],act; // not to change
int conv(char ch){return ((ch>='a' && ch<='z')?ch-'a':ch-'A'+26);}

void init(){
        for(int i=0;i<=act;++i){
                suf[i]=cnt[i]=dad[i]=0;
                adj[i].clear();
                memset(to[i], 0, sizeof(to[i]));
        }
        act=0;
}

int add(string& s){
        int u=0;
        for(char ch:s){
                int c=conv(ch);
                if(!to[u][c])to[u][c]=++act;
                u=to[u][c];
        }
        cnt[u]++;
        return u;
}

// O(sum(|s|)*alpha)
void build(){
        queue<int> q{{0}};
        while(!q.empty()){
                int u=q.front();q.pop();
                for(int i=0;i<alpha;++i){
                        int v=to[u][i];
                        if(!v)to[u][i]=to[suf[u]][i];
                        else q.push(v);
                        if(!u || !v)continue;
                        suf[v]=to[suf[u]][i];
                        dad[v]=cnt[suf[v]]?suf[v]:dad[suf[v]];
                }
        }
        for(int i=1;i<=act;++i){
                adj[i].push_back(dad[i]);
                adj[dad[i]].push_back(i);
        }
}
```

9.2 Hashing

```cpp
// O(n) build - O(1) get
// 1. prepare() in the main
// 2. hashing<string> hs("hello");
// 3. hs.get(l,r);

// Chars are in [1, BASE)
// BASE is prime or random(lim, MOD-lim)
// If chars are in [0, BASE) then compare the hashes for length
// 1000234999, 1000567999, 1000111997, 1000777121, 1001265673, 1001864327, 999727999, 1070777777
// if hash(multiset 1)==hash(multiset 2) then (r+a1)*(r+a2)...(r+an)==(r+b1)*(r+b2)...(r+bn) // (Collision n/MOD)

const ii BASE(257, 367);
const int MOD[2] = { 1001864327, 1001265673 };

int add(int a, int b, int m){return a+b>=m?a+b-m:a+b;}
int sbt(int a, int b, int m){return a-b<0?a-b+m:a-b;}
int mul(int a, int b, int m){return ll(a)*b%m;}
ll operator ! (const ii a) { return (ll(a.first) << 32) | a.second; }
ii operator + (const ii& a, const ii& b){return {add(a.first, b.first, MOD[0]), add(a.second, b.second, MOD[1])};}
ii operator - (const ii& a, const ii& b){return {sbt(a.first, b.first, MOD[0]), sbt(a.second, b.second, MOD[1])};}
ii operator * (const ii& a, const ii& b){return {mul(a.first, b.first, MOD[0]), mul(a.second, b.second, MOD[1])};}

const int maxn = 1e6+5;
ii pot[maxn];
void prepare(){ // remember!!!
        pot[0] = ii{1,1};
        rep(i,1,maxn) pot[i] = pot[i-1] * BASE;
}

template <class type>
struct Hashing{
        vector<ii> h;
        Hashing(type& t){
                h.assign(sz(t)+1, ii{0,0});
                rep(i,1,sz(h)) h[i] = h[i-1] * BASE + ii{t[i-1], t[i-1]};
```

```
		}
		ii get(int l, int r){
			return h[r+1] - h[l] * pot[r-l+1];
		}
};
ii combine(ii a, ii b, int lenb){
		return a * pot[lenb] + b;
}
```

9.3 Hashing 2D

```
// Revisar primero los comentarios en hashing!!!
// 1-indexed
const ii BX(3731, 3731), BY(2999, 2999);
const int MOD[2] = { 998244353, 1001265673 };
int add(int a, int b, int m){return a+b>=m?a+b-m:a+b;}
int sbt(int a, int b, int m){return a-b<0?a-b+m:a-b;}
int mul(int a, int b, int m){return ll(a)*b%m;}
ll operator ! (const ii a) { return (ll(a.first) << 32) |
	 a.second; }
ii operator + (const ii& a, const ii& b){return {add(a.
	first, b.first, MOD[0]), add(a.second, b.second, MOD
	[1])};}
ii operator - (const ii& a, const ii& b){return {sbt(a.
	first, b.first, MOD[0]), sbt(a.second, b.second, MOD
	[1])};}
ii operator * (const ii& a, const ii& b){return {mul(a.
	first, b.first, MOD[0]), mul(a.second, b.second, MOD
	[1])};}
const int maxn = 1e6+5;
ii PX[maxn], PY[maxn];
void prepare(){ // remember!!!
		PX[0] = PY[0] = ii{1,1};
		rep(i,1,maxn) {
			PX[i] = PX[i-1] * BX;
			PY[i] = PY[i-1] * BY;
		}
}
template <class type>
struct Hashing2D { // 1-indexed
		vector<vector<ii>> hs;
		int n, m;
		Hashing2D(vector<type>& s) {
			n = sz(s), m = sz(s[0]);
			hs.assign(n + 1, vector<ii>(m + 1, {0,0})
				);
			rep(i, 0, n) rep(j, 0, m)
				hs[i + 1][j + 1] = {s[i][j], s[i
					][j]};
			rep(i, 0, n+1) rep(j, 0, m)
				hs[i][j + 1] = hs[i][j + 1] + hs[
					i][j] * BY;
			rep(i, 0, n) rep(j, 0, m+1)
				hs[i + 1][j] = hs[i + 1][j] + hs[
					i][j] * BX;
		}
		ii get(int x1, int y1, int x2, int y2) {
			assert(1 <= x1 && x1 <= x2 && x2 <= n);
			assert(1 <= y1 && y1 <= y2 && y2 <= m);
			x1--;y1--;
			int dx = x2 - x1, dy = y2 - y1;
			return (hs[x2][y2] - hs[x2][y1] * PY[dy])
				- (hs[x1][y2] - hs[x1][y1] * PY[dy
					]) * PX[dx];
		}
};
```

9.4 KMP

```
// O(n)
vector<int> phi(string& s){
		int n=sz(s);
		vector<int> tmp(n);
		for(int i=1,j=0;i<n;++i){
			while(j>0 && s[j]!=s[i])j=tmp[j-1];
			if(s[i]==s[j])j++;
			tmp[i]=j;
		}
		return tmp;
}
// O(n+m)
int kmp(string& s, string& p){
		int n=sz(s),m=sz(p),cnt=0;
		vector<int> pi=phi(p);
		for(int i=0,j=0;i<n;++i){
			while(j && s[i]!=p[j])j=pi[j-1];
			if(s[i]==p[j])j++;
			if(j==m){
				cnt++;
				j=pi[j-1];
			}
		}
		return cnt;
}
```

9.5 KMP Automaton

```
const int maxn = 1e5+5;
const int alpha = 26;
int to[maxn][alpha];
```

```
int conv(char ch){return ((ch>='a' && ch<='z')?ch-'a':ch-
    'A'+26);}

// O(n*alpha)
void build(string& s){
        to[0][conv(s[0])]=1;
        int n=sz(s);
        for(int i=1,p=0;i<n+1;++i){
                for(int j=0;j<alpha;++j)to[i][j]=to[p][j
                    ];
                if(i<n){
                        to[i][conv(s[i])]=i+1;
                        p=to[p][conv(s[i])];
                }
        }
}
```

9.6 Lyndon Factorization

```
// A string is called simple if it is strictly smaller
    than all its nontrivial cyclic shifts.
// The Lyndon factorization of the string is s = w1 w2
    ... wk
// where all strings wi are simple, and they are in non-
    increasing order
// w1 >= w2 >= ... >= wk
// this factorization exists and it is unique

// O(n)
vector<string> duval(string& s){
        vector<string> factorization;
        int n=sz(s),i=0;
        while(i<n){
                int j=i+1,k=i;
                while(j<n && s[k]<=s[j]){
                        if(s[k]<s[j])k=i;
                        else k++;
                        j++;
                }
                while(i<=k){
                        factorization.push_back(s.substr(
                            i, j-k));
                        i+=j-k;
                }
        }
        return factorization;
}
```

9.7 Manacher

```
// O(n), par (raiz, izq, der) 1 - impar 0
vi manacher(string& s, int par){
        int l=0,r=-1,n=sz(s);
        vi m(n,0);
        for(int i=0;i<n;++i){
                int k=(i>r?(1-par):min(m[l+r-i+ par], r-i
                    +par))+par;
                while(i+k-par<n && i-k>=0 && s[i+k-par]==
                    s[i-k])++k;
                m[i]=k-par;--k;
                if(i+k-par>r)l=i-k,r=i+k-par;
        }
        for(int i=0;i<n;++i)m[i]=(m[i]-1+par)*2+1-par;
        return m;
}
```

9.8 Minimum Expression

```
// O(n)
int minimum_expression(string s){
        s=s+s;int n=sz(s),i=0,j=1,k=0;
        while(i+k<n && j+k<n){
                if(s[i+k]==s[j+k])k++;
                else if(s[i+k]>s[j+k])i=i+k+1,k=0; // <
                    max
                else j=j+k+1,k=0;
                if(i==j)j++;
        }
        return min(i, j);
}
```

9.9 Next Permutation

```
// O(n)
// 1) find the last i such that ai<ai+1
// 2) find the last j such that ai<aj
// 3) swap i and j, then reverse the segment [i+1, n-1]
string nextPermutation(string& s){
        string ans(s);
        int n=sz(s);
        int i=n-2;
        while(i>=0 && ans[i]>=ans[i+1])i--;
        if(i<0)return "no permutation";
        int j=n-1;
        while(ans[i]>=ans[j])j--;
        swap(ans[i], ans[j]);
        int l=i+1,r=n-1;
        while(r>l)swap(ans[r--], ans[l++]);
        return ans;
}
```

9.10 Palindromic Tree

```
const int alpha = 26;
const char mini = 'a';

// tree.suf: the longest suffix-palindrome link
// tree.dad - tree.to: the parent palindrome by removing
    the first and last character
// node 0 = root with len -1 for odd
// node 1 = root with len 0 for even
struct Node {
    int to[alpha], suf, len, cnt, dad;
    Node(int x, int l = 0, int c = 1): len(x), suf(l),
        cnt(c) {
        memset(to, 0, sizeof(to));
    }
    int& operator [] (int i) { return to[i]; }
};

struct PalindromicTree {
    vector<Node> tree;
        vector<int> palo; // longest suffix-palindrome in
            the position i
    string s;
    int n,last; // max suffix palindrome

    PalindromicTree(string t = "") {
        n = last = 0;
        tree.push_back(Node(-1));
        tree.push_back(Node(0));
                for(char& c:t) add_char(c);
                // Propagate counts up the suffix links
                for(int i=sz(tree)-1;i>=2;i--){
                        tree[tree[i].suf].cnt+=tree[i].
                            cnt;
                }
    }

    int getsuf(int p) {
        while (n - tree[p].len - 1 < 0 || s[n - tree[p].
            len - 1] != s[n])
                        p = tree[p].suf;
        return p;
    }

    void add_char(char ch) {
        s.push_back(ch);
        int p = getsuf(last), c = ch - mini;
        if (!tree[p][c]) {
            int suf = getsuf(tree[p].suf);
            suf = max(1, tree[suf][c]);
            tree[p][c] = sz(tree);
            tree.push_back(Node(tree[p].len + 2, suf, 0))
                ;
        }
        last = tree[p][c];
        tree[last].dad = p;
        tree[last].cnt++;n++;
                palo.push_back(tree[last].len);
        }
};
```

9.11 Suffix Array

```
// O(n*log(n)) - char in [1, lim)
// sa: is the starting position of the i-th lex smallest
    suffix
// rnk: is the rank (position in SA) of the suffix
    starting at i
// lcp: is the longest common prefix between sa[i] and sa
    [i+1]
auto SuffixArray(string s, int lim=256) {
        s.push_back(0); int n = sz(s), k = 0, a, b;
        vi sa, lcp, rnk(all(s)), y(n), ws(max(n, lim));
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2),
            lim = p) {
                p = j, iota(all(y), n - j);
                rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i]
                    - j;
                fill(all(ws), 0);
                rep(i,0,n) ws[rnk[i]]++;
                rep(i,1,lim) ws[i] += ws[i - 1];
                for (int i = n; i--;) sa[--ws[rnk[y[i]]]]
                    = y[i];
                swap(rnk, y), p = 1, rnk[sa[0]] = 0;
                rep(i,1,n) a = sa[i - 1], b = sa[i], rnk[
                    b] =
                        (y[a] == y[b] && y[a + j] == y[b
                            + j]) ? p - 1 : p++;
        }
        for (int i = 0, j; i < n - 1; lcp[rnk[i++]] = k)
                for (k && k--, j = sa[rnk[i] - 1]; s[i +
                    k] == s[j + k]; k++);
        reverse(all(lcp));lcp.pop_back();reverse(all(lcp)
            );
        return tuple{sa, rnk, lcp};
}
```

9.12 Suffix Automaton

```
// O(n*log(alpha))
// suf: suffix link (like aho if not match)
// len: length of the longest string in this state
// minlen: smallest string of node v = (v.suf==-1?0:v.suf
    .len) + 1
// end: if this state is terminal
// count different strings [i.suf.len+1, i.len]
// para saber cuantos substrings itnee a en b, ir
    procesando los
```

```
// prefijos y al marcarlos procesar la cantidad visitando
    los sufijos de los nodos
// contribucion es u.len - u.suf.len, tener en cuenta con
    que len se llego
// puede ser un len2 para manejar eso min(u.len, u.len2)-
   u.suf.len

// a->b->c->b->c
// b->c
template<int alpha = 26>
struct SuffixAutomaton {
        struct Node {
                // array<int, alpha> to; TLE, add -> int
                    conv(char ch)
                map<char, int> to;
                int len = 0, suf = 0;
                bool end = false;
        };

        vector<Node> sa;
        int last = 0;
        ll substrs = 0;

        SuffixAutomaton(string &s) {
                sa.reserve(sz(s)*2);
                last = add_node();
                sa[0].suf = -1;
                for (char &c : s) add_char(c);
                for (int p = last; p; p = sa[p].suf) sa[p
                    ].end = 1;
        }

        int add_node() { sa.push_back({}); return sz(sa)
            -1; }

        void add_char(char c) {
                int u = add_node(), p = last;
                sa[u].len = sa[last].len + 1;
                while (p != -1 && !sa[p].to.count(c)) {
                        sa[p].to[c] = u;
                        substrs += p != 0 ? sa[p].len -
                            sa[sa[p].suf].len : 1;
                        p = sa[p].suf;
                }
                if (p != -1) {
                        int q = sa[p].to[c];
                        if (sa[p].len + 1 != sa[q].len) {
                                int clone = add_node();
                                sa[clone] = sa[q];
                                sa[clone].len = sa[p].len
                                    + 1;
                                sa[q].suf = sa[u].suf =
                                    clone;
                                while (p != -1 && sa[p].
                                    to[c] == q) {
                                        sa[p].to[c] =
                                            clone;
                                        p = sa[p].suf;
                                }
                        } else sa[u].suf = q;
                }
                last = u;
        }

        // Aplicaciones

        int dfs(int u){ // count
                if(sa[u].cnt!=-1)return sa[u].cnt;
                sa[u].cnt=sa[u].end;
                for(auto [_,v]:sa[u].to){
                        sa[u].cnt+=dfs(v);
                }
                return sa[u].cnt;
        }

        void dfs2(int u){ // grade primero
                sa[u].pre--;
                if(sa[u].pre>0)return;
                for(auto [_,v]:sa[u].to){
                        sa[v].cnt2+=sa[u].cnt2;
                        dfs2(v);
                }
        }

        void dfs2(){
                vector<int> order(sz(sa)-1);
                for(int i=1;i<sz(sa);++i)order[i-1]=i;
                sort(order.begin(), order.end(), [&](int
                    a, int b) { return sa[a].len > sa[b].
                    len; });
                for(auto &i : order){
                        // suf.cnt += i.cnt
                }
        }

        int lcs(string& t){
                int u=0,l=0,ans=0;
                for(char c:t){
                        while(u && !sa[u].to.count(c)){
                                u=sa[u].suf;
                                l=sa[u].len;
                        }
                        if(sa[u].to.count(c)){
                                u=sa[u].to[c];
                                l++;
                        }
                        ans=max(ans, l);
                }
                return ans;
        }

        bool query(string& t){
                int u=0;
                for(char c:t){
```

```
				if(!sa[u].to.count(c))return
				    false;
				u=sa[u].to[c];
			}
			return true;
		}

		void cyclic(string& t){ // dfs(0) primero
			int u=0,l=0;
			int m=sz(t);
			t+=t;
			unordered_set<int> s; // vector<bool>
			for(char ch:t){
				int c=conv(ch);
				while(u && !sa[u].to[c]){
					u=sa[u].suf;
					l=sa[u].len;
				}
				if(sa[u].to[c]){
					u=sa[u].to[c];
					l++;
				}
				if(l==m){
					s.insert(u);
					if(sa[u].minlen==m){
						u=sa[u].suf;
						l=sa[u].len;
					}else{
						l--;
					}
				}
			}
			ll ans=0;
			for(int u:s)ans+=sa[u].cnt;
			cout<<ans<<"\n";
		}
	};
```

9.13 Suffix Tree

```
	// O(n)
	// pos: start of the edge
	// len: edge length
	// link: suffix link
	struct SuffixTree{
		vector<map<char,int>> to;
		vector<int> pos,len,link;
		int size=0,inf=1e9;
		string s;

		int make(int _pos, int _len){
			to.push_back(map<char,int>());
			pos.push_back(_pos);
			len.push_back(_len);
			link.push_back(-1);
			return size++;
		}

		void add(int& p, int& lef, char c){
			s+=c;++lef;int lst=0;
			for(;lef;p?p=link[p]:lef--){
				while(lef>1 && lef>len[to[p][s[sz
				    (s)-lef]]]){
					p=to[p][s[sz(s)-lef]],lef
					    -=len[p];
				}
				char e=s[sz(s)-lef];
				int& q=to[p][e];
				if(!q){
					q=make(sz(s)-lef,inf),
					    link[lst]=p,lst=0;
				}else{
					char t=s[pos[q]+lef-1];
					if(t==c){link[lst]=p;
					    return;}
					int u=make(pos[q],lef-1);
					to[u][c]=make(sz(s)-1,inf
					    );
					to[u][t]=q;
					pos[q]+=lef-1;
					if(len[q]!=inf)len[q]-=
					    lef-1;
					q=u,link[lst]=u,lst=u;
				}
			}
		}

		SuffixTree(string& _s){
			make(-1,0);int p=0,lef=0;
			for(char c:_s)add(p,lef,c);
			add(p,lef,'$'); // smallest char
			s.pop_back();
		}

		int query(string& p){
			for(int i=0,u=0,n=sz(p);;){
				if(i==n || !to[u].count(p[i]))
				    return i;
				u=to[u][p[i]];
				for(int j=0;j<len[u];++j){
					if(i==n || s[pos[u]+j]!=p
					    [i])return i;
					i++;
				}
			}
		}

		vector<int> sa;
		void genSA(int x=0, int Len=0){
			if(!sz(to[x]))sa.push_back(pos[x]-Len);
			else for(auto t:to[x])genSA(t.second,Len+
```

```
                len[x]);
        }
};
```

9.14 Trie

```
const int maxn = 2e6+5;
const int alpha = 26;

int to[maxn][alpha]; // to[u][c]: node u edge with the
    letter c
int cnt[maxn]; // count of word ending in this node
int act; // trie node cound
int conv(char ch){return ((ch>='a' && ch<='z')?ch-'a':ch-
    'A'+26);}

void init(){
        for(int i=0;i<=act;++i){
                memset(to[i],0,sizeof(to[i]));
                cnt[i]=0;
        }
        act=0;
}

void add(string& s){
        int u=0;
        for(char ch:s){
                int c=conv(ch);
                if(!to[u][c])to[u][c]=++act;
                u=to[u][c];
        }
        cnt[u]++;
}
```

9.15 Trie Bit

```
const int maxn = 5e5+5;
const int bits = 30;
const int alpha = 2;

int to[maxn*bits][alpha]; // to[u][c]: node u edge with
    the letter c
int cnt[maxn*bits]; // count of word ending in this node
int act; // trie node cound
int conv(int x, int i){return ((x&(1<<i))?1:0);}

void init(){
        for(int i=0;i<=act;++i){
                memset(to[i],0,sizeof(to[i]));
                cnt[i]=0;
        }
        act=0;
}

void add(int x){
        int u=0;
        for(int i=bits;i>=0;--i){
                int c=conv(x,i);
                if(!to[u][c])to[u][c]=++act;
                cnt[u]++;
                u=to[u][c];
        }
        cnt[u]++;
}

int mini(int x){
        int u=0,ans=0;
        for(int i=bits;i>=0;--i){
                int c=conv(x,i);
                if(!to[u][c] || cnt[to[u][c]]==0){
                        u=to[u][!c];
                        ans+=(1<<i);
                }else{
                        u=to[u][c];
                }
        }
        return ans;
}
```

9.16 Z Algorithm

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

9.17 El especial

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

const int N = 1e5 + 9;

struct ST {
  #define lc (n << 1)
  #define rc ((n << 1) | 1)
```

```
  long long t[4 * N], lazy[4 * N];
  ST() {
    memset(t, 0, sizeof t);
    memset(lazy, 0, sizeof lazy);
  }
  inline void push(int n, int b, int e) {
    if (lazy[n] == 0) return;
    t[n] = t[n] + lazy[n] * (e - b + 1);
    if (b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline long long combine(long long a,long long b) {
    return a + b;
  }
  inline void pull(int n) {
    t[n] = t[lc] + t[rc];
  }
  void upd(int n, int b, int e, int i, int j, int v) {
    push(n, b, e);
    if (j < b || e < i) return;
    if (i <= b && e <= j) {
      lazy[n] = v; //set lazy
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  long long query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if (i > e || b > j) return 0; //return null
    if (i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid
        + 1, e, i, j));
  }
}st;

struct node {
  int len, link, firstpos;
  map<char, int> nxt;
};
vector<node> t;
struct SuffixAutomaton {
  int sz, last;
  vector<int> terminal;
  vector<int> dp;
  vector<vector<int>> g;
  SuffixAutomaton() {}
  SuffixAutomaton(int n) {
    t.clear(); t.resize(2 * n);
    terminal.resize(2 * n, 0);
    dp.resize(2 * n, -1); sz = 1; last = 0;
    g.resize(2 * n);
    t[0].len = 0; t[0].link = -1; t[0].firstpos = 0;
  }
  void extend(char c) {
    int p = last;
    int cur = sz++;
    t[cur].len = t[last].len + 1;
    t[cur].firstpos = t[cur].len;
    p = last;
    while (p != -1 && !t[p].nxt.count(c)) {
      t[p].nxt[c] = cur;
      p = t[p].link;
    }
    if (p == -1) t[cur].link = 0;
    else {
      int q = t[p].nxt[c];
      if (t[p].len + 1 == t[q].len) t[cur].link = q;
      else {
        int clone = sz++;
        t[clone] = t[q];
        t[clone].len = t[p].len + 1;
        while (p != -1 && t[p].nxt[c] == q) {
          t[p].nxt[c] = clone;
          p = t[p].link;
        }
        t[q].link = t[cur].link = clone;
      }
    }
    last = cur;
  }
};

pair<int, int> modifies[N * 2];
int cnt;
namespace lct {
  int par[N * 2], lazy[N * 2], last[N * 2], c[N * 2][2];
  void mark(int x, int v) {
    lazy[x] = last[x] = v;
  }
  void push(int x) {
    if (lazy[x]) {
      if (c[x][0]) {
        mark(c[x][0], lazy[x]);
      }
      if (c[x][1]) {
        mark(c[x][1], lazy[x]);
      }
      lazy[x] = 0;
    }
  }
  bool is_root(int x) {
    return c[par[x]][0] != x && c[par[x]][1] != x;
```

```cpp
  }
  void rotate(int x) {
    int y = par[x], z = par[y], k = c[y][1] == x;
    if (!is_root(y)) {
      c[z][c[z][1] == y] = x;
    }
    par[c[y][k] = c[x][!k]] = y;
    par[par[c[x][!k] = y] = x] = z;
  }
  void splay(int x) {
    static int st[N];
    int top = 0;
    st[++top] = x;
    for (int i = x; !is_root(i); i = par[i]) {
      st[++top] = par[i];
    }
    while (top) {
      push(st[top--]);
    }
    while (!is_root(x)) {
      int y = par[x], z = par[y];
      if (!is_root(y)) {
        rotate((c[y][1] == x) == (c[z][1] == y) ? y : x);
      }
      rotate(x);
    }
  }
  void access(int x, int v) {
    int z = 0;
    cnt = 0;
    while (x) {
      splay(x);
      modifies[++cnt] = make_pair(t[x - 1].len, last[x]);
      c[x][1] = z;
      mark(x, v);
      z = x;
      x = par[x];
    }
  }
}

int pos[N];
vector<pair<int, int>> Q[N];
long long ans[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, q; cin >> n >> q;
  string s; cin >> s;
  SuffixAutomaton sa(n);
  for (int i = 1; i <= q; i++) {
    int l, r; cin >> l >> r;
    ++l; ++r;
    Q[r].push_back({l, i});
  }
  s = "." + s;
  pos[0] = 1;
  for (int i = 1; i <= n; ++i) {
    sa.extend(s[i]);
    pos[i] = sa.last + 1;
  }
  for (int i = 1; i <= sa.sz; ++i) {
    lct::par[i] = t[i - 1].link + 1;
  }
  for (int i = 1; i <= n; ++i) {
    st.upd(1, 1, n, 1, i, 1);
    lct::access(pos[i], i);
    int last = 0;
    for (int j = cnt; j > 1; --j) {
      pair<int, int> p = modifies[j];
      if (p.first) {
        if (p.second) {
          st.upd(1, 1, n, p.second - p.first + 1, p.
              second - last, -1);
        }
        last = p.first;
      }
    }
    // st.query(l, l) = number of distinct substrings
        which lastly occured in starting position l for
        prefix [1, i]
    for (auto [l, id]: Q[i]) {
      ans[id] = st.query(1, 1, n, l, i);
    }
  }
  for (int i = 1; i <= q; i++) {
    cout << ans[i] << '\n';
  }
  return 0;
}
```

10 Misc

10.1 Counting Sort

```cpp
// O(n+k)
void counting_sort(vi& a){
        int n=sz(a);
        int maxi=*max_element(all(a));
        int mini=*min_element(all(a));
        int k=maxi-mini+1;
        vi cnt(k,0);
        for(int i=0;i<n;++i)++cnt[a[i]-mini];
        for(int i=0,j=0;i<k;++i)
                while(cnt[i]--)a[j++]=i+mini;
}
```

10.2 Dates

```cpp
int dateToInt(int y, int m, int d){
        return 1461*(y+4800+(m-14)/12)/4+367*(m-2-(m-14)
            /12*12)/12-
                3*((y+4900+(m-14)/12)/100)/4+d-32075;
}
void intToDate(int jd, int& y, int& m, int& d){
        int x,n,i,j;x=jd+68569;
        n=4*x/146097;x-=(146097*n+3)/4;
        i=(4000*(x+1))/1461001;x-=1461*i/4-31;
        j=80*x/2447;d=x-2447*j/80;
        x=j/11;m=j+2-12*x;y=100*(n-49)+i+x;
}
int DayOfWeek(int d, int m, int y){     //starting on
    Sunday
        static int ttt[]={0, 3, 2, 5, 0, 3, 5, 1, 4, 6,
            2, 4};
        y-=m<3;
        return (y+y/4-y/100+y/400+ttt[m-1]+d)%7;
}
```

10.3 Expression Parsing

```cpp
// O(n) - eval() de python
bool delim(char c){return c==' ';}
bool is_op(char c){return c=='+' || c=='-' || c=='*' || c
    =='/';}
bool is_unary(char c){return c=='+' || c=='-';}

int priority(char op){
        if(op<0)return 3;
        if(op=='+' || op=='-')return 1;
        if(op=='*' || op=='/')return 2;
        return -1;
}

void process_op(stack<int>& st, char op){
        if(op<0){
                int l=st.top();st.pop();
                switch(-op){
                        case '+':st.push(l);break;
                        case '-':st.push(-l);break;
                }
        }else{
                int r=st.top();st.pop();
                int l=st.top();st.pop();
                switch(op){
                        case '+':st.push(l+r);break;
                        case '-':st.push(l-r);break;
                        case '*':st.push(l*r);break;
                        case '/':st.push(l/r);break;
                }
        }
}

int evaluate(string& s){
        stack<int> st;
        stack<char> op;
        bool may_be_unary=true;
        for(int i=0;i<sz(s);++i){
                if(delim(s[i]))continue;
                if(s[i] == '('){
                        op.push('(');
                        may_be_unary=true;
                }else if(s[i]==')'){
                        while(op.top()!='('){
                                process_op(st, op.top());
                                op.pop();
                        }
                        op.pop();
                        may_be_unary=false;
                }else if(is_op(s[i])){
                        char cur_op=s[i];
                        if(may_be_unary && is_unary(
                            cur_op))cur_op=-cur_op;
                        while(!op.empty() && ((cur_op >=
                            0 && priority(op.top()) >=
                            priority(cur_op)) || (cur_op <
                             0 && priority(op.top()) >
                            priority(cur_op)))){
                                process_op(st, op.top());
                                op.pop();
                        }
                        op.push(cur_op);
                        may_be_unary=true;
                }else{
                        int number=0;
                        while(i<sz(s) && isalnum(s[i]))
                            number=number*10+s[i++]-'0';
                        --i;
                        st.push(number);
                        may_be_unary=false;
                }
        }

        while(!op.empty()){
                process_op(st, op.top());
                op.pop();
        }
        return st.top();
}
```

10.4 Hanoi

```cpp
// hanoi(n) = 2 * hanoi(n-1) + 1
// hanoi(n, 1, 3)
```

```
vector<int> ans;
void hanoi(int x, int start, int end){
        if(!x)return;
        hanoi(x-1, start, 6-start-end);
        ans.push_back({start, end});
        hanoi(x-1, 6-start-end, end);
}
```

10.5 K mas frecuentes

```
// los k numeros mas frecuentes
// el cero es un valor neutral dentro del vector
// no usarlo en el array original (a[i] > 0, i e [0,n-1])
// el vector guarda {valor, contador}
// pero contador es para el algo, no es la cantidad real

// algoritmo de misra-gries O(k^2)
vector<ii> null(k, {0,0});
vector<ii> init(int v){
        vector<ii> a=null;
        a[0]={v,1};
        return a;
}
vector<ii> oper(vector<ii> a, vector<ii> b, int k) {
        for (int i = 0; i < k; ++i) if (b[i].first) {
                int p = -1, q = -1;
                for (int j = 0; j < k; ++j) {
                        if (b[i].first == a[j].first) p =
                            j;
                        if (!a[j].first) q = j;
                }
                if (p != -1) {
                        a[p].second += b[i].second;
                } else if (q != -1) {
                        a[q] = b[i];
                } else {
                        int mn = b[i].second;
                        for (int j = 0; j < k; ++j) mn =
                            min(mn, a[j].second);
                        for (int j = 0; j < k; ++j) a[j].
                            second -= mn;
                        b[i].second -= mn;
                        for (int j = 0; j < k; ++j) if (!
                            a[j].second) {
                                if (b[i].second > 0) {
                                        a[j] = b[i], b[i
                                            ].second = 0;
                                } else {
                                        a[j].first = 0;
                                }
                        }
                }
        }
        return a;
```



```
}
```

10.6 Prefix3D

```
const int N = 100;
int A[N][N][N];
int preffix[N + 1][N + 1][N + 1];

void build(int n){
        for (int x = 1; x <= n; x++){
                for (int y = 1; y <= n; y++){
                        for (int z = 1; z <= n; z++){
                                preffix[x][y][z] = A[x -
                                    1][y - 1][z - 1]
                                        + preffix[x - 1][
                                            y][z] +
                                            preffix[x][y -
                                            1][z] +
                                            preffix[x][y][
                                            z - 1]
                                        - preffix[x - 1][
                                            y - 1][z] -
                                            preffix[x -
                                            1][y][z - 1] -
                                             preffix[x][y
                                            - 1][z - 1]
                                        + preffix[x - 1][
                                            y - 1][z - 1];
                        }

                }

        }
}

ll query(int lx, int rx, int ly, int ry, int lz, int rz){
        ll ans = preffix[rx][ry][rz]
                - preffix[lx - 1][ry][rz] - preffix[rx][
                    ly - 1][rz] - preffix[rx][ry][lz - 1]
                + preffix[lx - 1][ly - 1][rz] + preffix[
                    lx - 1][ry][lz - 1] + preffix[rx][ly -
                     1][lz - 1]
                - preffix[lx - 1][ly - 1][lz - 1];

        return ans;
}
```

10.7 Ternary Search

```
// O(log((r-l)/eps))
// returna el maximo valor de f(x) en [l,r]
const double eps = 1e-9;
double f(double x);
double ternary(){
```

```
    double l,r;
    while(r-l>eps){
        double m1=l+(r-l)/3.0;
        double m2=r-(r-l)/3.0;
        if(f(m1)<f(m2))l=m1;
        else r=m2;
    }return max(f(l),f(r));
}
// ternary search para enteros
// O(log((r-l)/eps))
// returna el maximo valor de f(x) en [l,r]
int f(int x);
int ternary(){
    int l,r;
    while(r-l>6){
        int m1=l+(r-l)/3;
        int m2=r-(r-l)/3;
        if(f(m1)<f(m2))l=m1; // revisar desempate
        else r=m2;
    }
    int ans=l,val=f(l);
    for(int i=l+1;i<=r;++i){
        int val2=f(i);
        if(val2>val){
            val=val2;
            ans=i;
        }
    }
    return val;
}
```

11 Teoría y miscelánea

11.1 Sumatorias

- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$
- $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- $\sum_{i=1}^{n} i^5 = \frac{(n(n+1))^2(2n^2+2n-1)}{12}$
- $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ para $x \neq 1$

11.2 Teoría de Grafos

11.2.1 Teorema de Euler

En un grafo conectado planar, se cumple que $V - E + F = 2$, donde V es el número de vértices, E es el número de aristas y F es el número de caras. Para varios componentes la formula es: $V - E + F = 1 + C$, siendo C el número de componentes.

11.2.2 Planaridad de Grafos

Un grafo es planar si y solo si no contiene un subgrafo homeomorfo a K_5 (grafo completo con 5 vértices) ni a $K_{3,3}$ (grafo bipartito completo con 3 vértices en cada conjunto).

11.2.3 Truco del Cow Game

Dadas restricciones de la forma:

$$x_a - x_b \leq d$$

podemos transformar cada desigualdad en una arista dirigida:

$$b \to a \quad \text{con peso } d$$

Luego, ejecutando un algoritmo de camino más corto desde un nodo inicial s, obtenemos:

$$\text{dist}[i] = \max(x_i - x_s)$$

Nota: Pueden aparecer pesos negativos, por lo que se debe usar Bellman-Ford o SPFA, no Dijkstra.

11.3 Teoría de Números

11.3.1 Ecuaciones Diofánticas Lineales

Una ecuación diofántica lineal es una ecuación en la que se buscan soluciones enteras x e y que satisfagan la relación lineal $ax+by = c$, donde a, b y c son constantes dadas.

Para encontrar soluciones enteras positivas en una ecuación diofántica lineal, podemos seguir el siguiente proceso:

1. Encontrar una solución particular: Encuentra una solución particular (x_0, y_0) de la ecuación. Esto puede hacerse utilizando el algoritmo de Euclides extendido.

2. Encontrar la solución general: Una vez que tengas una solución particular, puedes obtener la solución general utilizando la fórmula:

$$x = x_0 + \frac{b}{\text{mcd}(a, b)} \cdot t$$

$$y = y_0 - \frac{a}{\text{mcd}(a, b)} \cdot t$$

donde t es un parámetro entero.

3. Restringir a soluciones positivas: Si deseas soluciones positivas, asegúrate de que las soluciones generales satisfagan $x \geq 0$ y $y \geq 0$. Puedes ajustar el valor de t para cumplir con estas restricciones.

11.3.2 Pequeño Teorema de Fermat

Si p es un número primo y a es un entero no divisible por p, entonces $a^{p-1} \equiv 1 \pmod{p}$.

11.3.3 Teorema de Euler

Para cualquier número entero positivo n y un entero a coprimo con n, se cumple que $a^{\phi(n)} \equiv 1 \pmod{n}$, donde $\phi(n)$ es la función phi de Euler, que representa la cantidad de enteros positivos menores que n y coprimos con n.

11.4 Geometría

11.4.1 Teorema de Pick

Sea un poligono simple cuyos vertices tienen coordenadas enteras. Si B es el numero de puntos enteros en el borde, I el numero de puntos enteros en el interior del poligono, entonces el area A del poligono se puede calcular con la formula:

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

11.4.2 Fórmula de Herón

Si los lados del triángulo tienen longitudes a, b y c, y s es el semiperímetro (es decir, $s = \frac{a+b+c}{2}$), entonces el área A del triángulo está dada por:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

11.4.3 Relación de Existencia Triangular

Para un triángulo con lados de longitud a, b, y c, la relación de existencia triangular se expresa como:

$$b - c < a < b + c, \quad a - c < b < a + c, \quad a - b < c < a + b$$

11.5 Combinatoria

11.5.1 Permutaciones

El número de permutaciones de n objetos distintos tomados de a r a la vez (sin repetición) se denota como $P(n, r)$ y se calcula mediante:

$$P(n, r) = \frac{n!}{(n-r)!}$$

11.5.2 Combinaciones

El número de combinaciones de n objetos distintos tomados de a r a la vez (sin repetición) se denota como $C(n, r)$ o $\binom{n}{r}$ y se calcula mediante:

$$C(n, r) = \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

11.5.3 Permutaciones con Repetición

El número de permutaciones de n objetos tomando en cuenta repeticiones se denota como $P_{\text{rep}}(n; n_1, n_2, \ldots, n_k)$ y se calcula mediante:

$$P_{\text{rep}}(n; n_1, n_2, \ldots, n_k) = \frac{n!}{n_1! n_2! \cdots n_k!}$$

11.5.4 Combinaciones con Repetición

El número de combinaciones de n objetos tomando en cuenta repeticiones se denota como $C_{\text{rep}}(n; n_1, n_2, \ldots, n_k)$ y se calcula mediante:

$$C_{\text{rep}}(n; n_1, n_2, \ldots, n_k) = \binom{n+k-1}{n} = \binom{n+k-1}{k-1}$$

11.5.5 Números de Catalan

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Los números de Catalan también pueden calcularse utilizando la siguiente fórmula recursiva:

$$C_0 = 1$$
$$C_{n+1} = \frac{4n+2}{n+2} C_n$$

Usos:

- $\text{Cat}(n)$ cuenta el número de árboles binarios distintos con n vértices.

- $\mathrm{Cat}(n)$ cuenta el número de expresiones que contienen n pares de paréntesis correctamente emparejados.

- $\mathrm{Cat}(n)$ cuenta el número de formas diferentes en que se pueden colocar $n+1$ factores entre paréntesis, por ejemplo, para $n = 3$ y $3 + 1 = 4$ factores: a, b, c, d, tenemos: $(ab)(cd), a(b(cd)), ((ab)c)d$ y $a((bc)d)$.

- Los números de Catalan cuentan la cantidad de caminos no cruzados en una rejilla n x n que se pueden trazar desde una esquina de un cuadrado o rectángulo a la esquina opuesta, moviéndose solo hacia arriba y hacia la derecha.

- Los números de Catalan representan el número de árboles binarios completos con $n + 1$ hojas.

- $\mathrm{Cat}(n)$ cuenta el número de formas en que se puede triangular un poligono convexo de $n + 2$ lados. Otra forma de decirlo es como la cantidad de formas de dividir un polígono convexo en triángulos utilizando diagonales no cruzadas.

11.5.6 Estrellas y barras

Número de soluciones de la ecuación $x_1 + x_2 + \cdots + x_k = n$.

- Con $x_i \geq 0$: $\binom{n+k-1}{n}$

- Con $x_i \geq 1$: $\binom{n-1}{k-1}$

Número de sumas de enteros con límite inferior:

Esto se puede extender fácilmente a sumas de enteros con diferentes límites inferiores. Es decir, queremos contar el número de soluciones para la ecuación:

$$x_1 + x_2 + \cdots + x_k = n$$

con $x_i \geq a_i$.

Después de sustituir $x'_i := x_i - a_i$ recibimos la ecuación modificada:

$$(x'_1 + a_i) + (x'_2 + a_i) + \cdots + (x'_k + a_k) = n$$

$$\Leftrightarrow \quad x'_1 + x'_2 + \cdots + x'_k = n - a_1 - a_2 - \cdots - a_k$$

con $x'_i \geq 0$. Así que hemos reducido el problema al caso más simple con $x'_i \geq 0$ y nuevamente podemos aplicar el teorema de estrellas y barras.

11.6 DP Optimization Theory

Name	Original Recurrence	Sufficient Condition	From	To
CH 1	$dp[i] = min_{j<i}\{dp[j] + b[j] * a[i]\}$	$b[j] \geq b[j+1]$ Optionally $a[i] \leq a[i+1]$	$O(n^2)$	$O(n)$
CH 2	$dp[i][j] = min_{k<j}\{dp[i-1][k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ Optionally $a[j] \leq a[j+1]$	$O(kn^2)$	$O(kn)$
D&Q	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(kn \log n)$
Knuth	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i, j-1] \leq A[i, j] \leq A[i+1, j]$	$O(n^3)$	$O(n^2)$

Notes:

- $A[i][j]$ - the smallest k that gives the optimal answer, for example in $dp[i][j] = dp[i-1][k] + C[k][j]$

- $C[i][j]$ - some given cost function

- We can generalize a bit in the following way $dp[i] = \min_{j<i}\{F[j] + b[j] * a[i]\}$, where $F[j]$ is computed from $dp[j]$ in constant time

11.7 Lista de números con mayor cantidad de divisores hasta 10^n

n	Número	# divisores
1	6	4
2	60	12
3	840	32
4	7560	64
5	83160	128
6	720720	240
7	8648640	448
8	73513440	768
9	735134400	1344
10	6983776800	2304
11	97772875200	4032
12	963761198400	6720
13	9316358251200	10752
14	97821761637600	17280
15	866421317361600	26880
16	8086598962041600	41472
17	74801040398884800	64512
18	897612484786617600	103680

11.8 Games

Grundy numbers. For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \text{mex}(\{G(y) : (x, y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games.

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.

- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.

- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.

- *Player must move in all games, and loses if can't move in some game.* A position is losing if any of the games is in a losing position.

Misère Nim. A position with pile sizes $a_1, a_2, \ldots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

11.9 2-SAT

2-SAT Rules

$p \to q \equiv \neg p \vee q$

$p \to q \equiv \neg q \to \neg p$

$p \vee q \equiv \neg p \to q$

$p \wedge q \equiv \neg(p \to \neg q)$

$\neg(p \to q) \equiv p \wedge \neg q$

$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$

$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$

$(p \to r) \wedge (q \to r) \equiv (p \wedge q) \to r$

$(p \to r) \vee (q \to r) \equiv (p \vee q) \to r$

$(p \wedge q) \vee (r \wedge s) \equiv (p \vee r) \wedge (p \vee s) \wedge (q \vee r) \wedge (q \vee s)$