Notebook UNTreeCiclo

Contents

1 C++

1.1 C++ plantilla

```
#include <bits/stdc++.h>
using namespace std;
#define watch(x) cout<<#x<<"="<<x<<'\n'
#define sz(arr) ((int) arr.size())
#define all(v) v.begin(), v.end()
typedef long long ll;
typedef long double ld;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
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";
const char ln = '\n';

int main() {
        ios::sync_with_stdio(false);
        cin.tie(0);
        cout << setprecision(20) << fixed;
    // freopen("file.in", "r", stdin);
    // freopen("file.out", "w", stdout);
        return 0;
}
```

1.2 Librerias

```
// 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>
#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 Bitmask

```
// Todas son O(1)Representacion
int a = 5; // Representacion binaria: 0101
int b = 3; // Representacion binaria: 0011
// Operaciones Principales
int resultado_and = a & b; // 0001 (1 en decimal)
int resultado_or = a | b; // 0111 (7 en decimal)
int resultado_xor = a ^ b; // 0110 (6 en decimal)

int num = 42; // Representacion binaria: 00101010
bitset<8> bits(num); // Crear un objeto bitset a partir
    del numero
cout << "Secuencia de bits: " << bits << "\n";
bits.count(); // Cantidad de bits activados
bits.set(3, true); // Establecer el cuarto bit en 1
bits.reset(6);     // Establecer el septimo bit en 0

ll S,T;
// Operaciones con bits (/*) por 2 (redondea de forma
    automatica)
S=34; // == 100010
S = S<<1; // == S*2 == 68 == 1000100
S = S>>2; // == S/4 == 17 == 10001
S = S>>1; // == S/2 == 8  == 1000

// Encender un bit
S = 34;
S = S|(1<<3); // S = 42 (101010)

// Limpiar o apagar un bit
// ~: Not operacion
S = 42;
S &= ~(1<<1); // S = 40 (101000)

// Comprobar si un bit esta encendido
S = 42;
T = S&(1<<3); // (!= 0): el tercer bit esta encendido

// Invertir el estado de un bit
S = 40;
S ^=(1<<2); // 44 (101100)

// LSB (Primero de la derecha)
S = 40;
T = ((S) & -(S)); // 8 (001000)
__builtin_ctz(T); // nos entrega el indice del LSB

// Encender todos los bits
ll n = 3; // el tamanio del set de bits
S = 0;
S = (1<<n) - 1; // 7 (111)

// n es el tamanio de la mask (Alternativa)
// ll n = 64;
// for (ll subset = 0; subset < (1<<n); ++subset){

// Enumerar todos los posibles subsets de un bitmask
int mask = 18;
for (int subset = mask; subset; subset = (mask & (subset
    -1))){
    cout << subset << "\n";
}

// otras funciones de c++
__builtin_popcount(32); // 100000 (base 2), only 1 bit is
    on
__builtin_popcount(30);// 11110 (base 2), 4 bits are on
__builtin_popcountl((1l<<62)-1l); // 2^62-1 has 62 bits
    on (near limit)
__builtin_ctz(32); // 100000 (base 2), 5 trailing zeroes
__builtin_ctz(30); // 11110 (base 2), 1 trailing zero
__builtin_ctzl(1l<<62); // 2^62 has 62 trailing zeroes
```

1.4 Cosas de strings

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

```
        vector<string> res;
        stringstream ss(s);
        string sub;
        while(getline(ss, sub, c))res.push_back(sub);
        return res;
}

for(char& c:s)c=toupper(c);
for(char& c:s)c=tolower(c);
int n=stoi(s); // de string a entero
int n=stoi(s, nullptr, 2); // base 2
double d=stod(s); // de string a double
string s=to_string(n); // de entero a string
```

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;
gp_hash_table<int, int, custom_hash> table;
```

1.6 Random

```
typedef unsigned long long u64;
mt19937_64 rng (chrono::steady_clock::now().
    time_since_epoch().count());
u64 hash=rng();
mt19937 rng (chrono::steady_clock::now().time_since_epoch
    ().count());
int rand(int a, int b){return uniform_int_distribution<
    int>(a, b)(rng);} // uniform_real_distribution
```

2 Arboles

2.1 Centroid Decomposition

```
// O(nlogn)
struct CentroidDecomposition{
        int dad[maxn],sz[maxn];
        set<int> adj[maxn]; // check, proc

        int operator[](int i){return dad[i];}
        void addEdge(int x,int y){adj[x].insert(y);adj[y
            ].insert(x);}

        void build(int v=0, int p=-1){
                int n=dfsSz(v, p);
                int centroid=dfsCentroid(v, p, n);
                dad[centroid]=p;
                for(int u:adj[centroid]){
                        if(p==u)continue;
                        // count paths
                }
                for(int u:adj[centroid]){
                        adj[u].erase(centroid);
                        build(u,centroid);
                }
                adj[centroid].clear();
        }

        int dfsSz(int v,int p){
                sz[v]=1;
                for(int u:adj[v]){
                        if(u==p)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)continue;
                        if(sz[u]>n/2)return dfsCentroid(u
                            ,v,n);
                }
                return v;
        }
};
// for(int b=a;b!=-1;b=cd[b])
```

2.2 Heavy Light Decomposition

```
ll null=LLONG_MIN;
ll oper(ll a, ll b){return max(a,b);}
// segtree build, set, upd, get

const int maxn=1e5+1;
bool VALS_IN_EDGES=false; // arista padre
struct HLD{
	int par[maxn], root[maxn], dep[maxn];
	int sz[maxn], pos[maxn], ti;
	vi adj[maxn];
	SegTree st;
	void addEdge(int x, int y){adj[x].push_back(y);
	    adj[y].push_back(x);}
	void dfsSz(int x){
		sz[x]=0;
		for(int& y:adj[x]){
			if(y==par[x])continue;
			par[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]=ti++;
		for(int y:adj[x]){
		if(y==par[x])continue;
		root[y]=(y==adj[x][0]?root[x]:y);
		dfsHld(y);
		}
	}
	void build(int n,int v=0){
		root[v]=par[v]=v;
		dep[v]=ti=0;
		dfsSz(v);
		dfsHld(v);
		// vl palst(n);
		// for(int i=0;i<n;++i)palst[pos[i]]=a[i
		    ];
		// st.build(palst, n);
		st.build(n);
	}
	template <class Oper>
	void processPath(int x, int y, Oper op){
		for(;root[x]!=root[y];y=par[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]+VALS_IN_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);
		});
	}
	ll queryPath(int x, int y){
		ll res=null;
		processPath(x,y,[this,&res](int l, int r)
		    {
			res=oper(res, st.get(l,r));
		});
		return res;
	}
	void modifySubtree(int x, int v){st.upd(pos[x]+
	    VALS_IN_EDGES,pos[x]+sz[x],v);}
	int querySubtree(int x){return st.get(pos[x]+
	    VALS_IN_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);
	}
};
```

2.3 LCA

```
const int maxn = 2e5+5, maxlog = 20+5;
int up[maxn][maxlog],dep[maxn]; // memset -1, 0
vi adj[maxn];
int n; // <-

void dfs(int v, int p=-1){
	up[v][0]=p;
	for(int u:adj[v]){
		if(u!=p){
			dep[u]=dep[v]+1;
			dfs(u, v);
		}
	}
}

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

int kth(int node, int k){
	for(int l=maxlog-1;l>=0;--l){
		if(node!=-1 && k&(1<<l)){
			node=up[node][l];
```

```
                }
        }
        return node;
}
int lca(int a, int b){
        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];
}
```

2.4 Sack

```
const int maxn = 1e5+5;
int ver[2*maxn],st[maxn],ft[maxn];
int len[maxn],dep[maxn];
vi adj[maxn];
int n,pos=0;

void prec(int v=0, int p=-1){
        len[v]=1;ver[pos]=v;
        st[v]=pos++;
        for(int u:adj[v]){
                if(u==p)continue;
                dep[u]=dep[v]+1;
                prec(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)// delete node
        else if(!vis[v] && add)// add node
}

void dfs(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)continue;
                dfs(u,v,0);
        }
        if(id!=-1)dfs(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);
        // answer queries
        if(keep)return;
        for(int p=st[v];p<ft[v];++p)ask(ver[p], 0);
}
```

2.5 Virtual Tree

```
const int maxn = 2e5+5;
vi adjVT[maxn],adj[maxn];
int st[maxn],ft[maxn];
bool important[maxn];
int n,q,pos=0;

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];}
bool cmp(int v, int u){return st[v]<st[u];}

// O(klogk)
int virtualTree(vi nodes){
        sort(all(nodes), cmp);
        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), cmp);
        nodes.erase(unique(all(nodes)), nodes.end());
        for(int u:nodes)adjVT[u].clear();

        vector<int> 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]; // root
        }
```

3 Estructuras de Datos

3.1 Disjoint Set Union

```
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--;
                }
        }
};
```

3.2 Dynamic Connectivity Offline

```
struct dsu{
        vi p,rank,h;
        int sets;
        dsu(int n){
                sets=n;
                p.assign(n,0);
                rank.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(rank[a]>rank[b])swap(a,b);
                rank[b]+=rank[a];
                h.push_back(a);
                p[a]=b;sets--;
        }
        void rollback(int x){
                int len=h.size();
                while(len>x){
                        int a=h.back();
                        h.pop_back();
                        rank[p[a]]-=rank[a];
                        p[a]=a;sets++;len--;
                }
        }
};

enum { ADD, DEL, QUERY };
struct Query{int type, u, v;};
struct DynCon{
        vector<Query> q;
        dsu uf;vi mt;
        map<pair<int,int>, int> prv;
        DynCon(int n): uf(n){}
        void add(int i, int j){
                if(i>j)swap(i, j);
                q.push_back({ADD, i, j});
                mt.push_back(-1);
                prv[{i,j}]=sz(q)-1;
        }
        void remove(int i, int j){
                if(i > j) swap(i, j);
                q.push_back({DEL, i, j});
                int pr=prv[{i, j}];
                mt[pr]=sz(q)-1;
                mt.push_back(pr);
        }
        void query(){q.push_back({QUERY, -1, -1});mt.
            push_back(-1);}
        void process(){ // answers all queries in order
                if(!sz(q))return;
                for(int i=0;i<sz(q);++i){
                        if(q[i].type==ADD && mt[i]<0)mt[i
                            ]=sz(q);
                }go(0, sz(q));
```

```
        }
        void go(int s, int e){
                if(s+1==e){
                if(q[s].type == QUERY)cout<<uf.sets<<"\n"
                    ;
                return;
                }int k=sz(uf.h),m=(s+e)/2;
                for(int i=e-1;i>=m;--i){
                if(mt[i]>=0 && mt[i]<s)uf.unite(q[i].u, q
                    [i].v);
                }go(s, m);
                uf.rollback(k);
                for(int i=m-1;i>=s;--i){
                if(mt[i]>=e)uf.unite(q[i].u, q[i].v);
                }go(m, e);
                uf.rollback(k);
        }
};
```

3.3 Dynamic Segment Tree

```
T null=0,nolz=0;
T oper(T a, T b);
struct Node{
        T val,lz;
        int l,r;
        Node *pl,*pr;
        Node(int ll, int rr){
                val=null;lz=nolz;
                pl=pr=nullptr;
                l=ll;r=rr;
        }
};

typedef Node* PNode;
void update(PNode x){
        if(x->r-x->l==1)return;
        x->val=oper(x->pl->val,x->pr->val);
}

void extends(PNode x){
        if(x->r-x->l!=1 && !x->pl){
                int m=(x->r+x->l)/2;
                x->pl=new Node(x->l, m);
                x->pr=new Node(m, x->r);
        }
}

void propagate(PNode x){
        if(x->r-x->l==1)return;
        if(x->lz==nolz)return;
        int m=(x->r+x->l)/2;
        // pl, pr
        x->lz=nolz;
}
```

```
struct SegTree{
        PNode root;
        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){
                        // val, lz
                        return;
                }
                extends(x);
                propagate(x);
                upd(x->pl,l,r,v);
                upd(x->pr,l,r,v);
                update(x);
        }

        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;
                extends(x);
                propagate(x);
                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);}
        void build(int l, int r){root=new Node(l, r+1);}
};
```

3.4 Fenwick Tree

```
typedef long long T;
struct FwTree{
        int n;
        vector<T> bit;
        FwTree(int n): n(n),bit(n+1){}
        T get(int r){
                T sum=0;
                for(++r;r;r-=r&-r)sum+=bit[r];
                return sum;
        }
        T get(int l, int r){return get(r)-(l==0?0:get(l
            -1));}
        void upd(int r, T v){
                for(++r;r<=n;r+=r&-r)bit[r]+=v;
        }
};

struct FwTree2d{
        int n, m;
        vector<vector<T>> bit;
```

```
        FwTree2d(){}
        FwTree2d(int n, int m):n(n),m(m),bit(n+1, vector<
            T>(m+1,0)){}
        T get(int x, int y){
                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){
                for(int i=x+1;i<=n;i+=i&-i)
                for(int j=y+1;j<=m;j+=j&-j)bit[i][j]+=dt;
        }
};
```

3.5 Link Cut Tree

```
typedef long long T;
struct SplayTree{
        struct Node{
                int ch[2]={0, 0},p=0;
                T val=0,path=0,sz=1;     // Path
                T sub=0,vir=0,ssz=0,vsz=0; // Subtree
                bool flip=0;T lz=0;      // 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]);
                            // check with st oper
                        ns[x].flip=0;
                }
                if(ns[x].lz){
                        // ...
                        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{ // 1-indexed
        LinkCut(int n):SplayTree(n){}

        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;
        }

        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;
    }

    void reroot(int x){
            access(x);ns[x].flip^=1;push(x);
    }

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

    void cut(int u){ // cut parent
            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){
            access(u);splay(u);push(u);
            return ns[u].sz;
    }

    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);}
    void upd(int u, int v, T val){
            int r=root(u);
            reroot(u);access(v);splay(v);
                // lazy
                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){
            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, v
        father
            int r=root(u);
            reroot(v);access(u);
            T ans=ns[u].vir+ns[u].val; // por el
                reroot
            return reroot(r),ans;
    }
};
```

3.6 Mos Algorithm

```
// O((n+q)*sqrt(n))
int sqrtn,n;
struct query {int l, r, idx;};
bool cmp (query a, query b) {
        int x = a.l/sqrtn;
        if (x != b.l/sqrtn) return x < b.l/sqrtn;
        return (x&1 ? a.r < b.r : a.r > b.r);
}

vector<int> ans;
vector<query> q;

int act();
void add(int i);
void remove(int i);

void solve(){
        sqrtn=(int)ceil(sqrt(n));
```

```
        sort(all(q), cmp);
        int l=0,r=-1;
        for(int i=0;i<sz(q);++i){
                while(r<q[i].r)add(++r);
                while(l>q[i].l)add(--l);
                while(r>q[i].r)remove(r--);
                while(l<q[i].l)remove(l++);
                ans[q[i].idx]=act();
        }
}
```

3.7 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));
// -------- 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
```

3.8 Persistent Segment Tree

```
typedef long long T;
struct Node{T val;int l,r;};
struct SegTree{
        vector<Node> ns;
        vi roots;
        int act=0;
        int size;

        T null=0;
        T oper(T a, T b);
        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));}
        void build(vector<T>& a, int n){size=n;roots.
           push_back(build(a, 0, size));}
};
```

3.9 RMQ

```
typedef long long T;
T oper(T a, T b); // max, min, gcd ...
struct RMQ {
        vector<vector<T>> table;
        void build(vector<T>& v){
```

```
                int n=v.size();
                table.assign(20, vector<T>(n)); // log2(n)
                for(int i=0;i<n;++i)table[0][i]=v[i];
                for(int j=1;(1<<j)<=n;++j)
                        for(int i=0;i+(1<<(j-1))<n;++i)
                                table[j][i]=oper(table[j-1][i],table[j-1][i+(1<<(j-1))]);
        }
        T get(int l, int r){
                int j=31-__builtin_clz(r-l+1);
                return oper(table[j][l], table[j][r-(1<<j)+1]);
        }
};
```

3.10 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(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);
    }
};
```

3.11 Segment Tree Recursivo

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

        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]=oper(vals[2*x+1], vals[2*x+2]);
        }

        void build(vector<T>& a,int n){
                size=1;
                while(size<n)size*=2;
                vals.resize(2*size);
                lazy.assign(2*size, nolz);
                build(a, 0, 0, size);
        }

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

        void upd(int l, int r, T v,int x, int lx, int rx){
                if(lx>=r || l>=rx)return;
                if(lx>=l && rx<=r){
                        // lazy, vals
                        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]);
        }

        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]=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];
			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 oper(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);}
};
```

3.12 Segment Tree 2D

```
const int N=1000+1;
ll st[2*N][2*N];
struct SegTree{
	int n,m,neutro=0;
	inline ll op(ll a, ll 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<vi>& a): n(sz(a)), m(n ? sz(a[0]) : 0){ build(a); }

	void build(vector<vi>& 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 upd(int x, int y, ll 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]);
	}

	ll get(int x0, int y0, int x1, int y1){
		ll 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;
	}
};
```

3.13 Segment Tree Beats

```
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;
};
struct SegTree{
	vector<Node> vals;int size;
	void oper(int a, int b, int c); // node c, left a, right b;
	Node single(T x){
		Node tmp;
		tmp.sum=tmp.max1=tmp.min1=x;
		tmp.maxc=tmp.minc=1;
		tmp.lazy=noVal;
		tmp.max2=-INF;
		tmp.min2=INF;
		return tmp;
	}

	void build(vector<T>& a,int n);
	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);
	}

	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);
		oper(2*x+1, 2*x+2, x);
	}

	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);
		oper(2*x+1, 2*x+2, x);
	}

	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);}
};
```

3.14 Sqrt Descomposition

```
// O(n/sqrt(n)+sqrt(n)) query
struct Sqrt {
	int block_size;
	vl nums,blocks;

	Sqrt(vi &arr){
		block_size=(int)ceil(sqrt(sz(arr)));
		blocks.assign(block_size, 0);
		nums=arr;
		for(int i=0;i<sz(nums);++i){
			blocks[i/block_size]+=nums[i];
		}
	}

	void update(int x, int v){
		blocks[x/block_size]-=nums[x];
		nums[x]=v;
		blocks[x/block_size]+=nums[x];
	}

	ll query(int r){
		ll res=0;
		for(int i=0;i<r/block_size;++i){res+=
			blocks[i];}
		for(int i=(r/block_size)*block_size;i<r
			;++i){res+=nums[i];}
		return res;
	}

	ll query(int l, int r){return query(r)-query(l-1)
		;}
};
```

3.15 Treap

```
// treap => order asc, implicit treap => order array
typedef long long T;
struct Treap{
	Treap *l,*r,*dad;
	u64 prior;
	T sz,value,sum,lz;
	Treap(T v){
		l=r=nullptr;
		lz=0;sz=1;
		prior=rng();
		value=sum=v;
	}
```

```
		~Treap(){delete l;delete r;}
};

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

void propagate(PTreap x){
	if(x && x->lz){
		if(x->l); // lz, value, sum ...
		if(x->r); // lz, value, sum ...
		x->lz=0;
	}
}

void update(PTreap x){
	propagate(x->l);
	propagate(x->r);
	x->sz=cnt(x->l)+cnt(x->r)+1;
	x->sum=sum(x->l)+sum(x->r)+x->value;
	if(x->l)x->l->dad=x;
	if(x->r)x->r->dad=x;
}

void upd(PTreap x, T v){
	if(!x)return;
	update(x);
	// lz, value, sum ...
}

// pair<PTreap, PTreap> split(PTreap x, T key){ // f <=
   key < s
pair<PTreap, PTreap> split(PTreap x, int left){ // cnt(f)
    == left
	if(!x)return {nullptr, nullptr};
	propagate(x);
	if(cnt(x->l)>=left){ // if(x->value>key){
		auto got=split(x->l, left); // , key);
		x->l=got.second;
		update(x);
		return {got.first, x};
	}else{
		auto got=split(x->r, left-cnt(x->l)-1);
		    // , key);
		x->r=got.first;
		update(x);
		return {x, got.second};
	}
}

PTreap merge(PTreap x, PTreap y){
	if(!x)return y;
	if(!y)return x;
	propagate(x);
	propagate(y);
	if(x->prior<=y->prior){
		x->r=merge(x->r, y);
		update(x);
		return x;
	}else{
		y->l=merge(x, y->l);
		update(y);
		return y;
	}
}

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->value);
	x->r=combine(x->r, z.second);
	x->l=combine(z.first, x->l);
	return x;
}

T kth(PTreap& x, int k){ // indexed 0
	if(!x)return null;
	if(k==cnt(x->l))return x->value;
	if(k<cnt(x->l))return kth(x->l, k);
	return kth(x->r, k-cnt(x->l)-1);
}

pair<int, T> lower_bound(PTreap x, T key){ // index,
    value
	if(!x)return {0, null};
	if(x->value<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->value;
	return y;
}

void dfs(PTreap x){
	if(!x)return;
	propagate(x);
	dfs(x->l);cout<<x->value<<" ";dfs(x->r);
}

// PTreap root=nullptr;
// PTreap act=new Treap(c);
// root=merge(root, act);
```

3.16 Two Stacks

```
typedef long long T;
struct Node{T val,acum;};
struct TwoStacks{
	stack<Node> s1,s2;
```

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

	void remove(){
		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 good(){
		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()
		}
	}
};
```

3.17 Wavelet Tree

```cpp
const int maxn = 1e5+5, maxv = 1e9, minv = -1e9;
struct WaveletTree{ // indexed 1 - O(nlogn)
	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
```

```cpp
	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;
	}
};
// int a[maxn];
// WaveletTree wt;
// for(int i=1;i<=n;++i)cin>>a[i];
// wt.build(a+1, a+n+1, minv, maxv);
```

4 Flujos

4.1 Dinic

```cpp
// O(|E|*|V|^2)
struct edge { ll v, cap, inv, flow; };
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, g[v].size(), 0});
    g[v].push_back({u, 0, g[u].size()-1, c});
  }
  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;
  }
  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;
        }
      }
    }
    set<ii> ans;
    for (int i = 0; i<n; i++){
      for (auto &e : g[i]){
        if (vis[i] && !vis[e.v]){
          ans.insert({i+1, e.v+1});
        }
      }
    }
    for (auto [x, y] : ans) cout << x << ' ' << y << ln;
  }
  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;
  }
};
```

4.2 Edmonds Karp

```cpp
//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()) {
```

```cpp
        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;
}
```

4.3 Hopcroft Karp

```cpp
// Complexity: 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) : l(l), r(r), nil(l+r), g(l+r),
                      d(1+l+r, INF), match(l+r, l+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;
    }
  }
};
```

4.4 Maximum Bipartite Matching

```cpp
// 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), seen(r), match(r), g(l)
      {}
  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;
    }
  }
};
```

4.5 Minimum Cost Maximum Flow

```
/// Complexity: 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};
  }
};
```

4.6 Weighted Matching

```
/// Complexity: 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 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 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 * (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 {
        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);
    } // clockwise = -
int sign(lf x){ return (lf(0) < x) - (x < lf(0)); }

// x 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 angle(pt a, pt b){ return acos(max((lf)-1.0, min((
    lf)1.0, dot(a, b)/norm(a)/norm(b)))); } // min 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)));
    });
}
```

5.2 Lineas

```
struct line {
    pt v; T c; // v:direction c: pos in y axis
    line(pt v, T c) : v(v), c(c) {}
    line(T a, T b, T c) : v({b,-a}), c(c) {} // ax + by =
        c
    line(pt p, pt q) : v(q-p), c(cross(v,p)) {}
    T side(pt p) { return cross(v,p)-c; }
    lf dist(pt p) { return abs(side(p)) / abs(v); }
    lf sq_dist(pt p) { return side(p)*side(p) / (lf)norm(
        v); }
    line perp_through(pt p) { return {p, p + rot90ccw(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
    line translate(pt t) { return {v, c + cross(v,t)}; }
    line shift_left(double d) { return {v, c + d*abs(v)};
        }
    pt proj(pt p) { return p - rot90ccw(v)*side(p)/norm(v
        ); } // pt proyected on the line
    pt refl(pt p) { return p - rot90ccw(v)*2*side(p)/norm
        (v); } // pt reflected on the other side of the
        line
};

bool inter_ll(line l1, line l2, pt &out) {
    T d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v*l1.c - l1.v*l2.c) / d; // floating points
    return true;
}
//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/abs(l2.v) + l1.v/abs(l1.v) * sign,
            l2.c/abs(l2.v) + l1.c/abs(l1.v) * sign};
}
```

5.3 Poligonos

```
enum {IN, OUT, ON};
struct polygon {
```

```
    vector<pt> p;
    polygon(int n) : p(n) {}
    int top = -1, bottom = -1;
    void delete_repetead() {
        vector<pt> aux;
        sort(p.begin(), p.end());
        for(pt &i : p)
            if(aux.empty() || aux.back() != i)
              aux.push_back(i);
        p.swap(aux);
    }
    bool is_convex() {
        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);
    }
    lf area(bool s = false) { // better on clockwise
       order
        lf ans = 0;
        for (int i = 0, n = p.size(); i < n; i++)
            ans += cross(p[i], p[(i+1)%n]);
        ans /= 2;
        return s ? ans : abs(ans);
    }
    lf perimeter() {
        lf per = 0;
        for(int i = 0, n = p.size(); i < n; i++)
           per += abs(p[i] - p[(i+1)%n]);
        return per;
    }
    bool above(pt a, pt p) { return p.y >= a.y; }
    bool crosses_ray(pt a, pt p, pt q) { // pq crosses
       ray from a
        return (above(a,q)-above(a,p))*orient(a,p,q) > 0;
    }
    int in_polygon(pt a) {
        int crosses = 0;
        for(int i = 0, n = p.size(); i < n; i++) {
            if(on_segment(p[i], p[(i+1)%n], a)) return ON
               ;
            crosses += crosses_ray(a, p[i], p[(i+1)%n]);
        }
        return (crosses&1 ? IN : OUT);
    }
    void normalize() { /// polygon is CCW
        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;
        top = max_element(p.begin(), p.end()) - p.begin()
           ;
    }
    int in_convex(pt a) {
        assert(bottom == 0 && top != -1);
        if(a < p[0] || a > p[top]) return OUT;
        T orientation = orient(p[0], p[top], a);
        if(orientation == 0) {
            if(a == p[0] || a == p[top]) return ON;
            return top == 1 || top + 1 == p.size() ? ON :
               IN;
        } else if (orientation < 0) {
            auto it = lower_bound(p.begin()+1, p.begin()+
               top, a);
            T d = orient(*prev(it), a, *it);
            return d < 0 ? IN : (d > 0 ? OUT: ON);
        } else {
            auto it = upper_bound(p.rbegin(), p.rend()-
               top-1, a);
            T d = orient(*it, a, it == p.rbegin() ? p[0]
               : *prev(it));
            return d < 0 ? IN : (d > 0 ? OUT: ON);
        }
    }
    polygon cut(pt a, pt b) { // cuts polygon on line ab
        line l(a, b);
        polygon new_polygon(0);
        for(int i = 0, n = p.size(); i < n; ++i) {
            pt c = p[i], d = p[(i+1)%n];
            lf abc = cross(b-a, c-a), abd = cross(b-a, d-
               a);
            if(abc >= 0) new_polygon.p.push_back(c);
            if(abc*abd < 0) {
              pt out; inter_ll(l, line(c, d), out);
              new_polygon.p.push_back(out);
            }
        }
        return new_polygon;
    }
    void convex_hull() {
        sort(p.begin(), p.end());
        vector<pt> ch;
        ch.reserve(p.size()+1);
        for(int it = 0; it < 2; it++) {
            int start = ch.size();
            for(auto &a : p) {
                /// if colineal are needed, use < and
                   remove repeated points
                while(ch.size() >= start+2 && orient(ch[
                   ch.size()-2], ch.back(), a) <= 0)
                    ch.pop_back();
                ch.push_back(a);
            }
            ch.pop_back();
```

```
            reverse(p.begin(), p.end());
        }
        if(ch.size() == 2 && ch[0] == ch[1]) ch.pop_back();
        /// be careful with CH of size < 3
        p.swap(ch);
    }
    vector<pii> antipodal() {
        vector<pii> 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;
    }
    pt centroid() {
        pt c{0, 0};
        lf scale = 6. * area(true);
        for(int i = 0, n = p.size(); i < n; ++i) {
            int j = (i+1 == n ? 0 : i+1);
            c = c + (p[i] + p[j]) * cross(p[i], p[j]);
        }
        return c / scale;
    }
    ll pick() {
        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 area() + 1 - boundary/2;
    }
    pt& operator[] (int i){ return p[i]; }
};
```

5.4 Circulos

```
struct circle {
    pt c; T r;
};
// (x-xo)^2 + (y-yo)^2 = r^2
//circle that passes through abc
circle center(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 + rot90ccw(b*norm(c) - c*norm(b))/cross(b,c)/2;
    return {cen, abs(a-cen)};
}
//centers of the circles that pass through ab and has radius r
vector<pt> centers(pt a, pt b, T r) {
    if (abs(a-b) > 2*r + eps) return {};
    pt m = (a+b)/2;
    double f = sqrt(r*r/norm(a-m) - 1);
    pt c = rot90ccw(a-m)*f;
    return {m-c, m+c};
}
int inter_cl(circle c, line l, pair<pt, pt> &out) {
    lf h2 = c.r*c.r - l.sq_dist(c.c);
    if(h2 >= 0) { // line touches circle
        pt p = l.proj(c.c);
        pt h = l.v*sqrt(h2)/abs(l.v); // vector of len h parallel to line
        out = {p-h, p+h};
    }
    return 1 + sign(h2); // if 1 -> out.F == out.S
}
int inter_cc(circle c1, circle c2, pair<pt, pt> &out) {
    pt d = c2.c-c1.c;
    double d2 = norm(d);
    if(d2 == 0) { assert(c1.r != c2.r); return 0; } // concentric circles (identical)
    double pd = (d2 + c1.r*c1.r - c2.r*c2.r)/2; // = |O_1P| * d
    double h2 = c1.r*c1.r - pd*pd/d2; // = h^2
    if(h2 >= 0) {
        pt p = c1.c + d*pd/d2, h = rot90ccw(d)*sqrt(h2/d2);
        out = {p-h, p+h};
    }
    return 1 + sign(h2);
}
//circle-line inter = 1
```

```
int tangents(circle c1, circle c2, bool inner, vector<pair<pt,pt>> &out) {
    if(inner) c2.r = -c2.r; // inner tangent
    pt d = c2.c-c1.c;
    double dr = c1.r-c2.r, d2 = norm(d), h2 = d2-dr*dr;
    if(d2 == 0 || h2 < 0) { assert(h2 != 0); return 0; } // (identical)
    for(double s : {-1,1}) {
        pt v = (d*dr + rot90ccw(d)*sqrt(h2)*s)/d2;
        out.push_back({c1.c + v*c1.r, c2.c + v*c2.r});
    }
    return 1 + (h2 > 0); // if 1: circle are tangent
}
//circle targent passing through pt p
int tangent_through_pt(pt p, circle c, pair<pt, pt> &out) {
    double d = abs(p - c.c);
    if(d < c.r) return 0;
    pt base = c.c-p;
    double w = sqrt(norm(base) - c.r*c.r);
    pt a = {w, c.r}, b = {w, -c.r};
    pt s = p + base*a/norm(base)*w;
    pt t = p + base*b/norm(base)*w;
    out = {s, t};
    return 1 + (abs(c.c-p) == c.r);
}
```

5.5 Semiplanos

```
struct halfplane{
    double angle;
    pt p, pq;
    halfplane(){}
    halfplane(pt a, pt b): p(a), pq(b - a) {
        angle = atan2(pq.y,pq.x);
    }
    bool operator < (halfplane b)const{return angle < b.angle;}
    bool out(pt q){return cross(pq, (q-p)) < -eps;} // checks if p is inside the half plane
};

const lf inf = 1e100;
// intersection pt of the lines of 2 halfplanes
pt inter(halfplane& h1, halfplane& h2){
    if(abs(cross(unit(h1.pq), unit(h2.pq))) <= eps)return {inf, inf};
    lf alpha = cross((h2.p - h1.p), h2.pq) / cross(h1.pq, h2.pq);
    return h1.p + (h1.pq * alpha);
}

// intersection of halfplanes
vector<pt> 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;
}
```

5.6 Segmentos

```
bool in_disk(pt a, pt b, pt p) { // pt p inside ab disk
    return dot(a-p, b-p) <= 0;
}
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) {
    T oa = orient(c,d,a),
    ob = orient(c,d,b),
    oc = orient(a,b,c),
    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}; //if cross -> 1
        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; // 0, 2
    }
    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(abs(p-a), abs(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
    }
```

6 Grafos

6.1 Puentes

```
    vector<bool> visited;
    vi tin, low;
    int timer;

    void IS_BRIDGE(int u, int v, vii &puentes){
        puentes.push_back({min(u, v), max(u, v)});
    }

    void dfs(vector<vi> &adj, vii &puentes, int v, int p =
       -1) {
        visited[v] = true;
        tin[v] = low[v] = timer++;
        for (int to : adj[v]) {
            if (to == p) continue;
            if (visited[to]) {
                low[v] = min(low[v], tin[to]);
            } else {
                dfs(adj, puentes, to, v);
                low[v] = min(low[v], low[to]);
                if (low[to] > tin[v])
                    IS_BRIDGE(v, to, puentes);
            }
        }
    }

    void find_bridges(vector<vi> &adj, vii &puentes, int n) {
        timer = 0;
        visited.assign(n, false);
        tin.assign(n, -1);
        low.assign(n, -1);
        for (int i = 0; i < n; ++i) {
            if (!visited[i])
                dfs(adj, puentes, i);
        }
    }
```

6.2 Puntos de Articulacion

```
    int n;
    vector<vector<int>> adj;

    vector<bool> visited;
    vector<int> tin, low;
    int timer;

    void dfs(int v, int p = -1) {
        visited[v] = true;
        tin[v] = low[v] = timer++;
        int children=0;
        for (int to : adj[v]) {
            if (to == p) continue;
            if (visited[to]) {
                low[v] = min(low[v], tin[to]);
            } else {
                dfs(to, v);
                low[v] = min(low[v], low[to]);
                if (low[to] >= tin[v] && p!=-1)
                    IS_CUTPOINT(v);
                ++children;
            }
        }
        if(p == -1 && children > 1)
            IS_CUTPOINT(v);
    }

    void find_cutpoints() {
        timer = 0;
        visited.assign(n, false);
        tin.assign(n, -1);
        low.assign(n, -1);
        for (int i = 0; i < n; ++i) {
            if (!visited[i])
                dfs (i);
        }
```

```
    }
```

6.3 Kosajaru

```
//Encontrar las componentes fuertemente conexas en un
    grafo dirigido
//Componente fuertemente conexa: es un grupo de nodos en
    el que hay
//un camino dirigido desde cualquier nodo hasta cualquier
     otro nodo dentro del grupo.
void Kosaraju(int u, int pass) {
    dfs_num[u] = 1;
    vii &neighbor = (pass == 1) ? AL[u] : AL_T[u];
    for (auto &[v, w] : neighbor)
        if (dfs_num[v] == UNVISITED)
            Kosaraju(v, pass);
    S.push_back(u);
}

int main(){
    S.clear();
    dfs_num.assign(N, UNVISITED);
    for (int u = 0; u < N; ++u)
        if (dfs_num[u] == UNVISITED)
            Kosaraju(u, 1);
    numSCC = 0;
    dfs_num.assign(N, UNVISITED);
    for (int i = N-1; i >= 0; --i)
        if (dfs_num[S[i]] == UNVISITED)
            ++numSCC, Kosaraju(S[i], 2);
    printf("There are %d SCCs\n", numSCC);
}
```

6.4 Tarjan

```
vi low, num, comp, g[nax];
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);
}
```

6.5 Dijkstra

```
//Camino mas cortos
//NO USAR CON PESOS NEGATIVOS, usar Bellman Ford o SPFA(
    mas rapido)
// O ((V+E)*log V)
vi dijkstra(vector<vii> &adj, int s, int V){
    vi dist(V+1, INT_MAX); dist[s] = 0;
    priority_queue<ii, vii, greater<ii> > pq; pq.push(ii
        (0, s));
    while(!pq.empty()){
        ii front = pq.top(); pq.pop();
        int d = front.first, u = front.second;
        if (d > dist[u]) continue;

        for (int j = 0; j < (int)adj[u].size(); j++){
            ii v = adj[u][j];
            if (dist[u] + v.second < dist[v.first]){
                dist[v.first] = dist[u] + v.second;
                pq.push(ii(dist[v.first], v.first));
            }
        }
    }
    return dist;
}
```

6.6 Bellman Ford

```
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;
    }
```

6.7 Floyd Warshall

```
//Camino minimo entre todos los pares de vertices
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n; cin >> n;
    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
    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]);
            }
        }
    }
}
```

6.8 MST Kruskal

```
//Arbol de minima expansion
//O(E*log V)
int main() {
    int n, m;
    cin >> n >> m;
    vector<pair<int, ii>> adj; //Los pares son: {peso, {
        vertice, vecino}}

    for (int i = 0; i<m; i++){
        int x, y, w; cin >> x >> y >> w;
        adj.push_back(make_pair(w, ii(x, y)));
    }

    sort(adj.begin(), adj.end());

    int mst_costo = 0, tomados = 0;
    dsu UF(n);
    for (int i = 0; i<m && tomados < n-1; i++){
        pair<int, ii> front = adj[i];
        if (!UF.is_same_set(front.second.first, front.
            second.second)){
            tomados++;
            mst_costo += front.first;
            UF.unionSet(front.second.first, front.second.
                second);
        }
    }
    cout << mst_costo;
}
```

6.9 MST Prim

```
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;
}
```

6.10 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 edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            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;
}
```

6.11 Camino mas corto de longitud fija

```
/*
Modificar operacion * de matrix de esta forma:
En la exponenciacion binaria inicializar matrix ans = b
*/
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;
}
```

6.12 2sat

```
// O(n+m)
// l=(x1 or y1) and (x2 or y2) and ... and (xn or yn)
struct sat2 {
        int n;
        vector<vector<vi>> g;
        vector<bool> vis, val;
        vi comp;
        stack<int> st;

        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;}
        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);}
        void diff(int u, int v){eq(u, neg(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].PB(v);
                g[1][v].PB(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++);
```

```
                }
        }
        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;
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";
```

7 Matematicas

7.1 Coeficientes binomiales

```
const int MAX_N = 100010;   // MOD > MAX_N
// O (log MOD)
ll inv (ll a){
    return binpow(a, MOD-2, MOD);
}

ll fact[MAX_N];
// O(log MOD)
ll C(int n, int k){
    if (n < k) return 0;
    return (((fact[n] * inv(fact[k])) % MOD) * inv(fact[n
        -k])) % MOD;
}

int main() {
    fact[0] = 1;
    for (int i = 1; i<MAX_N; i++){
        fact[i] = (fact[i-1]*i) % MOD;
    }
    cout << C(100000, 50000) << "\n";
    return 0;
}
```

7.2 Criba Modificada

```
//Criba modificada
/*
Si hay que determinar el numero de factores primos para
    muchos (o un rango) de enteros.
La mejor solucion es el algoritmo de criba modificada O(N
    log log N)
*/
int numDiffPFarr[MAX_N+10] = {0}; // e.g., MAX_N = 10^7
for (int i = 2; i <= MAX_N; ++i)
    if (numDiffPFarr[i] == 0) // i is a prime number
        for (int j = i; j <= MAX_N; j += i)
            ++numDiffPFarr[j]; // j is a multiple of i

//Similar para EulerPhi
int EulerPhi[MAX_N+10];
for (int i = 1; i <= MAX_N; ++i) EulerPhi[i] = i;
for (int i = 2; i <= MAX_N; ++i)
    if (EulerPhi[i] == i) // i is a prime number
        for (int j = i; j <= MAX_N; j += i)
            EulerPhi[j] = (EulerPhi[j]/i) * (i-1);
```

7.3 Ecuaciones Diofanticas

```
// O(log(min(a, b)))
ll extEuclid(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;   //Devuelve gcd(a, b)
}

bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0,
    ll &g) {
    g = extEuclid(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

7.4 Funcion Totient de Euler

```
//EulerPhi(N): contar el numero de enteros positivos < N
    que son primos relativos a N.
//El vector p es el que genera la criba de eratostenes
//Phi(N) = N * productoria(1 - (1/pi))
ll EulerPhi(ll N) {
    ll ans = N; // Empezar con ans = N
    for (int i = 0; (i < (int)p.size()) && (p[i]*p[i] <=
        N); ++i) {
        if (N%p[i] == 0) ans -= ans/p[i]; //contar
            factores
        while (N%p[i] == 0) N /= p[i];    //primos unicos
    }
    if (N != 1) ans -= ans/N; // ultimo factor
    return ans;
}
```

7.5 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;
}
```

7.6 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;
}
```

7.7 Fibonacci Fast Doubling

```
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};
}
```

7.8 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)
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;
}
```

7.9 Gauss Jordan

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be
    infinity or a big number

int gauss (vector < vector<double> > a, vector<double> &
    ans) {
     int n = (int) a.size();
     int m = (int) 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;
          for (int i=col; i<=m; ++i)
               swap (a[sel][i], a[row][i]);
          where[col] = row;

          for (int i=0; i<n; ++i)
               if (i != row) {
                    double c = a[i][col] / a[row][col];
                    for (int j=col; j<=m; ++j)
                         a[i][j] -= a[row][j] * c;
               }
          ++row;
     }

     ans.assign (m, 0);
     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;
}
```

7.10 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;
}
```

7.11 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))
```

7.12 Integral Definida

```
double f(double x){
  return x*x;
}

const int N = 1000 * 1000; // number of steps (already
    multiplied by 2)
double simpson_integration(double a, double b){
```

```
    double h=(b-a)/N;
    double s=f(a)+f(b);
    for (int i=1;i<=N-1;i++){
      double x=a+h*i;
      s+=f(x)*((i & 1)?4:2);
    }
    s*=h/3;
    return s;
}
```

7.13 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);
}
```

7.14 Logaritmo Discreto

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    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;
}
```

7.15 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;
}
```

7.16 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;
    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;
}
```

7.17 Mobius

```
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];
    }
  }
}
```

7.18 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);
}
```

7.19 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) - numero de variables c, numero de
   restricciones b
struct Simplex {
        vector<vector<double>> a;
        vector<double> b, c;
        double z;int n, m;
```

```
vi X, Y;
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];
}
Simplex(vector<vector<double>> &A,vector<double>
    &B,vector<double> &C){
        a=A;b=B;c=C;
        n=b.size();m=c.size();z= 0.0;
        X.resize(m);iota(X.begin(),X.end(),0);
        Y.resize(n);iota(Y.begin(),Y.end(),m);
}
// {z, {x1,x2,x3...}}
pair<double, vector<double>> maximize(){
        while(true){
                int x=-1,y=-1;
                double 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); // no hay solucion
                    para Ax<=b
                pivot(x,y);
        }
        while(true){
                double 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;
                double 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); // unbounded
                pivot(x,y);
        }
        vector<double> r(m);
        for(int i=0;i<n;++i){
                if(Y[i]<m)r[Y[i]]=b[i];
        }
        return {z,r};
    }
};
```

7.20 Fast Fourier Transform

```
const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a);
      }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang))
      , w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w
          = w * wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b
```

```
      /= n;
}
vector<long long> multiply(vector<int> &a, vector<int> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a);
//   while((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
```

7.21 Number Theoretic Transform

```
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;
}
```

8 Programacion dinamica

8.1 LIS

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n;cin >> n;
    vl vals(n);
    for (int i = 0; i < n; i++) cin >> vals[i];

    vl copia(vals);
    sort(copia.begin(),copia.end());

    map <ll,ll> dicc;
    for (int i=0;i<n;i++)if (!dicc.count(copia[i])) dicc[copia[i]]=i;

    vl baseSt(n,0);
    nodeSt st(baseSt, 0, n - 1);
    ll maxi =0;
    for (ll pVal:vals) {
        ll op =st.get(0,dicc[pVal]-1)+1;
        maxi = max(maxi,op);
        st.actl(dicc[pVal],op);
    }
    cout<<maxi<<ln;
```

```
}
```

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 Algoritmo de 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);
            }
        }
    }
}
```

8.4 Knuth Clasico

```
const int N = 1010;
const int INF = (int) 1e9;
int v[N], dp[N][N], sum[N], best[N][N];

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n;
    while(cin >> n) {
        if(n == 0) break;
        for(int i = 0; i < n; i++) cin >> v[i];

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

        for(int i = 0; i < n; i++) best[i][i] = i;

        for(int len = 2; len <= n; ++len) {
            for(int i = 0; i+len-1 < n; ++i) {
                int j = i+len-1;
                int &ref = dp[i][j];
                ref = INF;
                for(int k = best[i][j-1]; k <= best[i+1][
                    j]; ++k) {
                    if(k < j) {
                        int cur = dp[i][k] + dp[k+1][j];
                        if(cur < ref) {
                            best[i][j] = k;
                            ref = cur;
                        }
                    }
                }
```

```
                ref += sum[j+1] - sum[i];
            }
        }
        cout << dp[0][n-1] << '\n';
    }
    return 0;
}
```

8.5 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];
}
```

8.6 Divide Conquer

```
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 Knuth

```
#Condiciones
#C(b,c)<=C(a,d)
#C(a,c)+C(b,d)<=C(a,d)+C(b,c)
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;
```

```
}
```

9 Strings

9.1 Hashing

```
// 1000234999, 1000567999, 1000111997, 1000777121,
    1001265673, 1001864327, 999727999, 1070777777
const int MOD[2] = { 1001864327, 1001265673 };
const ii BASE(257, 367), ZERO(0, 0), ONE(1, 1);
const int MAXN = 1e6;

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

ii p[MAXN+1];
void prepare(){ // Acordate del prepare()!!
        p[0]=ONE;
        for(int i=1;i<=MAXN;i++)p[i]=p[i-1]*BASE;
}

ii combine(ii a, ii b, int lenb){return a*p[lenb]+b;}
template <class type>
struct hashing{
        vector<ii> h;
        hashing(type& t){
                h.resize(sz(t)+1);
                h[0]=ZERO;
                for(int i=1;i<sz(h);++i)
                        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]*p[r-l
            +1];}
};
```

9.2 KMP

```
// O(n)
vi phi(string& s){
        int n=sz(s);
        vi 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;
        vi 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;
}
```

9.3 KMP Automaton

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

// O(n*alpha)
void build(string& s){
        aut[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)aut[i][j]=aut[p][
                    j];
                if(i<n){
                        aut[i][conv(s[i])]=i+1;
                        p=aut[p][conv(s[i])];
                }
        }
}
```

9.4 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;
	}
```

9.5 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; //
		   cambiar por < para max
		else j=j+k+1,k=0;
		if(i==j)j++;
	}
	return min(i, j);
}
```

9.6 Palindromic Tree

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

// tree suf is the longest suffix palindrome
// tree dad is the palindrome add c to the right and left
struct Node{
	int next[alpha];
	int len,suf,dep,cnt,dad;
};

// O(nlogn)
struct PalindromicTree{
	vector<Node> tree;
	string s;
	int len,n;
	int size; // node 1 - root with len -1, node 2 -
	   root with len 0
	int last; // max suffix palindrome

	bool addLetter(int pos){
		int cur=last,curlen=0;
		int let=s[pos]-fc;
		while(true){
			curlen=tree[cur].len;
			if(pos-1-curlen>=0 && s[pos-1-
			   curlen]==s[pos])break;
			cur=tree[cur].suf;
		}
		if(tree[cur].next[let]){
			last=tree[cur].next[let];
			tree[last].cnt++;
			return false;
		}

		size++;
		last=size;
		tree[size].len=tree[cur].len+2;
		tree[cur].next[let]=size;
		tree[size].cnt=1;
		tree[size].dad=cur;

		if(tree[size].len==1){
			tree[size].suf=2;
			tree[size].dep=1;
			return true;
		}

		while(true){
			cur=tree[cur].suf;
			curlen=tree[cur].len;
			if(pos-1-curlen>=0 && s[pos-1-
			   curlen]==s[pos]){
				tree[size].suf=tree[cur].
				   next[let];
				break;
			}
		}

		tree[size].dep=1+tree[tree[size].suf].dep
		   ;
		return true;
	}

	PalindromicTree(string& _s, int n){
		tree.assign(n+4,Node());
		tree[1].len=-1;tree[1].suf=1;
		tree[2].len=0;tree[2].suf=1;
		size=2;last=2;s=_s;

		for(int i=0;i<n;i++)addLetter(i);
		for(int i=size;i>=3;i--)tree[tree[i].suf
		   ].cnt+=tree[i].cnt;
	}
};
```

9.7 Suffix Array

```
// O(nlogn)
struct SuffixArray{
```

```
    const int alpha = 256;
    string s;int n;
    vi sa,rnk,lcp;

    void build(string& _s){
        s=_s;s.push_back('$'); // check
        n=sz(s);
        sa.assign(n, 0);
        rnk.assign(n, 0);
        lcp.assign(n-1, 0);
        buildSA();
    }

    void buildSA() {
        vi cnt(max(alpha, n),0);
        for(int i=0;i<n;++i)cnt[s[i]]++;
        for(int i=1;i<max(alpha,n);++i)cnt[i]+=cnt[i-1];
        for(int i=n-1;i>=0;--i)sa[--cnt[s[i]]]=i;
        for(int i=1;i<n;++i)rnk[sa[i]]=rnk[sa[i-1]]+(s[sa[i]]!=s[sa[i-1]]);

        for(int k=1;k<n;k*=2){
            vi nsa(n),nrnk(n),ncnt(n);
            for(int i=0;i<n;++i)sa[i]=(sa[i]-k+n)%n;
            for(int i=0;i<n;++i)ncnt[rnk[i]]++;
            for(int i=1;i<n;++i)ncnt[i]+=ncnt[i-1];
            for(int i=n-1;i>=0;--i)nsa[--ncnt[rnk[sa[i]]]]=sa[i];
            for(int i=1;i<n;++i){
                ii op1={rnk[nsa[i]], rnk[(nsa[i]+k)%n]};
                ii op2={rnk[nsa[i-1]], rnk[(nsa[i-1]+k)%n]};
                nrnk[nsa[i]]=nrnk[nsa[i-1]]+(op1!=op2);
            }
            swap(sa, nsa);swap(rnk, nrnk);
        }

        for(int i=0,k=0;i<n-1;++i){
            while(s[i+k]==s[sa[rnk[i]-1]+k])k++;
            lcp[rnk[i]-1]=k;
            if(k)k--;
        }
    }
};
```

9.8 Suffix Automaton

```
// O(n*log(alpha))
struct SuffixAutomaton{
    vector<map<char,int>> to;
    vector<bool> end;vi suf,len; // len, longest string
    int last;

    SuffixAutomaton(string& s){
        to.push_back(map<char,int>());
        suf.push_back(-1);
        len.push_back(0);
        last=0;

        for(int i=0;i<sz(s);i++) {
            to.push_back(map<char,int>());
            suf.push_back(0);
            len.push_back(i+1);
            int r=sz(to)-1;

            int p=last;
            while(p>=0 && to[p].find(s[i])==to[p].end()){
                to[p][s[i]]=r;
                p=suf[p];
            }
            if(p!=-1){
                int q=to[p][s[i]];
                if(len[p]+1==len[q]){
                    suf[r]=q;
                }else{
                    to.push_back(to[q]);
                    suf.push_back(suf[q]);
                    len.push_back(len[p]+1);
                    int qq=sz(to)-1;
                    suf[q]=qq;
                    suf[r]=qq;
                    while(p>=0 && to[p][s[i]]==q){
                        to[p][s[i]]=qq;
                        p=suf[p];
                    }
                }
            }
            last=r;
        }

        end.assign(sz(to), false);
        int p=last;
        while(p){
            end[p]=true;
            p=suf[p];
        }
    }
```

```
};
```

9.9 Suffix Tree

```
// O(n)
struct SuffixTree{
        vector<map<char,int>> to;
        vector<int> pos,len,link;
        const int inf = 1e9;
        int size=0;
        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;
                        }
                }
        }

        void build(string& _s){
                make(-1,0);int p=0,lef=0;
                for(char c:_s)add(p,lef,c);
                add(p,lef,'$');
                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]);
        }
};
```

9.10 Trie

```
const int maxn = 2e6+5, alpha = 26, bits = 30;
int to[maxn][alpha],cnt[maxn],act;

void init(){
        for(int i=0;i<=act;++i){
                cnt[i]=0;
                // suf[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;
}

// Aho-Corasick
vector<int> adj[maxn]; // dad or suf
int dad[maxn],suf[maxn];

// O(sum(n)*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);
		}
}
```

9.11 Z Algorithm

```
// O(n)
vi z_function(string& s){
	int n=sz(s),l=0,r=0;vi 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;
}
```

10 Misc

10.1 Counting Sort

```
// O(n+k)
void counting_sort(vi& a){
	int maxi=*max_element(all(a));
	int mini=*min_element(all(a));
	int k=maxi-mini+1,n=sz(a);
	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 Expression Parsing

```
// En python es eval()
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();
}
```

10.3 Ternary Search

```
// O(log((r-l)/eps))
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));
}
```

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).

11.3 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.
- $\text{Cat}(n)$ cuenta el número de expresiones que contienen n pares de paréntesis correctamente emparejados.
- $\text{Cat}(n)$ 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