Notebook UNTreeCiclo

Contents

1 C++ 2
1.1 C++ plantilla 2
1.2 Librerias 3
1.3 Bitmask 3
1.4 Cosas de strings 3
1.5 Custom Hashing 4
1.6 Random 4

2 Arboles 4
2.1 Centroid Decomposition 4
2.2 Heavy Light Decomposition 4
2.3 LCA 5
2.4 Sack 6
2.5 Virtual Tree 6

3 Estructuras de Datos 6
3.1 Disjoint Set Union 6
3.2 Dynamic Connectivity Offline 7
3.3 Dynamic Segment Tree 8
3.4 Fenwick Tree 8
3.5 Li Chao 8
3.6 Link Cut Tree 9
3.7 Mos Algorithm 11
3.8 Ordered set 11
3.9 Persistent Segment Tree 12
3.10 RMQ 12
3.11 Segment Tree Iterativo 12
3.12 Segment Tree Recursivo 13
3.13 Segment Tree 2D 13
3.14 Segment Tree Beats 14
3.15 Sparse Table 2D 15
3.16 Sqrt Descomposition 15
3.17 Treap 16
3.18 Two Stacks 17
3.19 Wavelet Tree 17

4 Flujos 18
4.1 Blossom 18
4.2 Dinic 19
4.3 Edmonds Karp 20
4.4 Hopcroft Karp 20
4.5 Maximum Bipartite Matching 21
4.6 Minimum Cost Maximum Flow 21
4.7 Weighted Matching 22
4.8 Hungarian 23

5 Geometria 24
5.1 Puntos 24
5.2 Lineas 25
5.3 Poligonos 25
5.4 Circulos 28
5.5 Semiplanos 30
5.6 Segmentos 31
5.7 Convex Hull 32
5.8 Closest Points 32
5.9 Min Circle 32
5.10 3D 33
5.11 KD Tree 34

6 Grafos 34
6.1 Puentes 34
6.2 Puntos de Articulacion 35
6.3 Kosajaru 35
6.4 Tarjan 35
6.5 Dijkstra 36
6.6 Bellman Ford 36
6.7 Floyd Warshall 36
6.8 MST Kruskal 36
6.9 MST Prim 37
6.10 Shortest Path Faster Algorithm 37
6.11 Camino mas corto de longitud fija 37
6.12 2sat 38

7 Matematicas 38
7.1 De Bruijn sequences 38
7.2 Chinese Remainder Theorem 39
7.3 Totient y Divisores 39
7.4 Ecuaciones Diofanticas 39
7.5 Exponenciacion binaria 40
7.6 Exponenciacion matricial 40
7.7 Fibonacci Fast Doubling 41
7.8 Freivalds algorithm 41

7.9 Gauss Jordan 41
7.10 Gauss Jordan mod 2 41
7.11 GCD y LCM 42
7.12 Integral Definida 42
7.13 Inverso modular 42
7.14 Logaritmo Discreto 42
7.15 Miller Rabin 43
7.16 Miller Rabin Probabilistico 43
7.17 Mobius 43
7.18 Pollard Rho 44
7.19 Simplex 44
7.20 Fast Fourier Transform 44
7.21 Number Theoretic Transform 45

8 Programacion dinamica 46
8.1 Bin Packing 46
8.2 CHT 46
8.3 CHT Dynamic 46
8.4 Divide Conquer 47
8.5 Edit Distances 47
8.6 Kadane 2D 48
8.7 Knuth 48
8.8 LIS 48
8.9 SOS 48

9 Strings 49
9.1 Hashing 49
9.2 KMP 49
9.3 KMP Automaton 49
9.4 Manacher 50
9.5 Minimum Expression 50
9.6 Palindromic Tree 50
9.7 Suffix Array 51
9.8 Suffix Automaton 51
9.9 Suffix Tree 52
9.10 Trie 52
9.11 Z Algorithm 53

10 Misc 53
10.1 Counting Sort 53
10.2 Dates 53
10.3 Expression Parsing 53
10.4 Ternary Search 54

11 Teoría y miscelánea 54
11.1 Sumatorias 54
11.2 Teoría de Grafos 55
11.2.1 Teorema de Euler 55
11.2.2 Planaridad de Grafos 55
11.3 Teoría de Números 55
11.3.1 Ecuaciones Diofánticas Lineales 55
11.3.2 Pequeño Teorema de Fermat 55
11.3.3 Teorema de Euler 55
11.4 Geometría 55
11.4.1 Teorema de Pick 55
11.4.2 Fórmula de Herón 55
11.4.3 Relación de Existencia Triangular 55
11.5 Combinatoria 55
11.5.1 Permutaciones 55
11.5.2 Combinaciones 56
11.5.3 Permutaciones con Repetición 56
11.5.4 Combinaciones con Repetición 56
11.5.5 Números de Catalan 56
11.5.6 Estrellas y barras 56
11.6 DP Optimization Theory 57

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

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

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

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

__builtin_clz(x)      -> Cantidad de bits apagados por la
    izquierda
__builtin_ctz(x)      -> Cantidad de bits apagados por la
    derecha. Indice del bit encendido mas a la derecha
__builtin_popcount(x) -> Cantida de bits encendidos

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

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

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

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(nlog(n))
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;
                // add dfs for 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

```
typedef long long T;
T null;
T oper(T a, T b);
// Segment tree
const int maxn=1e5+1; // >= 2e5, remove struct
bool edges=false; // arista padre
struct HLD{
```

```
int par[maxn], root[maxn], dep[maxn];
int sz[maxn], pos[maxn], ti;
vector<int> 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);
}
// O(log^2(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]+edges,pos[y]);
}
void modifyPath(int x, int y, int v){
        processPath(x,y,[this,&v](int l, int r){
                st.upd(l,r,v);
        });
}
T queryPath(int x, int y){
        T res=null;
        processPath(x,y,[this,&res](int l, int r)
                    {
                        res=oper(res, st.get(l,r));
                });
                return res;
        }
        void modifySubtree(int x, int v){st.upd(pos[x]+
            edges,pos[x]+sz[x],v);}
        int querySubtree(int x){return st.get(pos[x]+
            edges,pos[x]+sz[x]);}
        void modify(int x, int v){st.set(pos[x],v);}
        void modifyEdge(int x, int y, int v){
                if(dep[x]<dep[y])swap(x,y);
                modify(x,v);
        }
};
```

2.3 LCA

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

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

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 st[maxn], ft[maxn], ver[2*maxn];
int len[maxn], n, q, pos=0;
vi adj[maxn];

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

// O(nlogn)
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)
                        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], n, pos=0;
bool important[maxn];

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();

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

// vi nodes(k);
// for(int& x:nodes)important[x]=true;
// int root=virtualTree(nodes);
// dp(root) - output answer - reset
```

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 Li Chao

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

struct Line {
  ty m, b;

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

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

struct nLiChao{
        // see coments for min

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

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

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

                if (mid) swap(nline, line);

                if (r == l) return;

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

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

                        else right -> addLine(nline);
                }
        }

        // T(Log(Rango))
        ty get(ty x) {
                ty m = (l + r) >> 1;
                ty op1 = -inf, op2 = -inf; // change to
                    inf
                if(l == r) return line.eval(x);
                else if(x < m){
                        if (left) op1 = left -> get(x);
                        return max(line.eval(x), op1); //
                            change max to min
                }
                else{
                        if (right) op2 = right -> get(x);
                        return max(line.eval(x), op2); //
                            change max to min
                }
        }
};

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

3.6 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;
        }
};
```

3.7 Mos Algorithm

```
// O((n+q)*s), s=n^(1/2)
// O(q*(s+(n/s)^2) => O(q*(n^(2/3))), s=(2*(n^2))^(1/3) -
   s=n^(2/3)
int s,n;
struct upd{int i,old,cur;};
struct query {int l,r,t,idx;};
bool cmp(query& a, query& b){
        int x=a.l/s;
        if(a.l/s!=b.l/s)return a.l/s<b.l/s;
        if(a.r/s!=b.r/s)return (x&1?a.r<b.r:a.r>b.r);
        return a.t<b.t;
}

vector<int> ans;
vector<query> qu;
vector<upd> up;

int act();
void add(int i);
void remove(int i);
void update(int i,int v,int l,int r){
        if(l<=i && i<=r); // add, remove
}

void solve(){
        s=(int)ceil(sqrt(n));
        sort(all(qu), cmp);
        int l=0,r=-1,t=0;
        for(int i=0;i<sz(qu);++i){
                while(t<qu[i].t)update(up[t].i,up[t].cur,
                   l,r),++t;
                while(t>qu[i].t)--t,update(up[t].i,up[t].
                   old,l,r);
                while(r<qu[i].r)add(++r);
                while(l>qu[i].l)add(--l);
                while(r>qu[i].r)remove(r--);
                while(l<qu[i].l)remove(l++);
                ans[qu[i].idx]=act();
        }
}

// tree
int st[maxn],ft[maxn],ver[maxn*2];
bool vis[maxn];

void ask(int v){
        vis[v]=!vis[v];
        if(vis[v])add(v);
        else remove(v);
}

// query[i] = {st[a]+1, st[b], i} + lca
```

3.8 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
```

3.9 Persistent Segment Tree

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

        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));}
};
```

3.10 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=sz(v);
                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]);
        }
};
```

3.11 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);
        }
};
```

3.12 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);}
};
```

3.13 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;
        }
};
```

3.14 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);}
};
```

3.15 Sparse Table 2D

```
const int MAX_N = 100;
const int MAX_M = 100;
const int KN = log2(MAX_N)+1;
const int KM = log2(MAX_M)+1;
int table[KN][MAX_N][KM][MAX_M];
int _log2N[MAX_N+1];
int _log2M[MAX_M+1];

int MAT[MAX_N][MAX_M];
int n, m, ic, ir, jc, jr;

void calc_log2() {
    _log2N[1] = 0;
    _log2M[1] = 0;
    for (int i = 2; i <= MAX_N; i++) _log2N[i] = _log2N[i
        /2] + 1;
    for (int i = 2; i <= MAX_M; i++) _log2M[i] = _log2M[i
        /2] + 1;
}

void build() {
    for (ir = 0; ir < n; ir++) {
        for (ic = 0; ic < m; ic++)
            table[0][ir][0][ic] = MAT[ir][ic];
        for (jc = 1; jc < KM; jc++)
            for (ic = 0; ic + (1 <<(jc-1)) < m; ic++)
                table[0][ir][jc][ic] = min(table[0][ir][
                    jc-1][ic], table[0][ir][jc-1][ic + (1
                    << (jc-1))]);
    }

    for (jr = 1; jr < KN; jr++)
        for (ir = 0; ir < n; ir++)
            for (jc = 0; jc < KM; jc++)
                for (ic = 0; ic < m; ic++)
                    table[jr][ir][jc][ic] = min(table[jr
                        -1][ir][jc][ic], table[jr-1][ir
                        +(1<<(jr-1))][jc][ic]);
}

int rmq(int x1, int y1, int x2, int y2) {
    int lenx = x2-x1+1;
    int kx = _log2N[lenx];
    int leny = y2-y1+1;
    int ky = _log2M[leny];

    int min_R1 = min(table[kx][x1][ky][y1], table[kx][x1
        ][ky][y2 + 1 - (1<<ky)]);
    int min_R2 = min(table[kx][x2+1-(1<<kx)][ky][y1],
        table[kx][x2+1- (1<<kx)][ky][y2 + 1 - (1<<ky)]);
    return min(min_R1, min_R2);
}
```

3.16 Sqrt Descomposition

```
typedef long long T;
struct Sqrt { // O(n/b+b)
        int b; // check b
        vector<T> nums,blocks;
        void build(vector<T>& arr, int n){
                b=(int)ceil(sqrt(n));nums=arr;
                blocks.assign(b, 0);
                for(int i=0;i<n;++i){
                        blocks[i/b]+=nums[i];
                }
        }

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

        T get(int r){
                T res=0;
                for(int i=0;i<r/b;++i){res+=blocks[i];}
                for(int i=(r/b)*b;i<r;++i){res+=nums[i];}
                return res;
        }

        T get(int l, int r){return get(r+1)-get(l);}
};
```

3.17 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);
```

3.18 Two Stacks

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

        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()
                }
        }
};
```

3.19 Wavelet Tree

```
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
        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 Blossom

```
/// Complexity: O(|E||V|^2)
/// Tested: https://tinyurl.com/oe5rnpk
struct network {
  struct struct_edge { int v; struct_edge * n; };
  typedef struct_edge* edge;
  int n;
  struct_edge pool[MAXE]; ///2*n*n;
  edge top;
  vector<edge> adj;
  queue<int> q;
  vector<int> f, base, inq, inb, inp, match;
  vector<vector<int>> ed;
  network(int n) : n(n), match(n, -1), adj(n), top(pool),
      f(n), base(n),
                   inq(n), inb(n), inp(n), ed(n, vector<
                      int>(n)) {}
  void add_edge(int u, int v) {
    if(ed[u][v]) return;
    ed[u][v] = 1;
    top->v = v, top->n = adj[u], adj[u] = top++;
    top->v = u, top->n = adj[v], adj[v] = top++;
  }
  int get_lca(int root, int u, int v) {
    fill(inp.begin(), inp.end(), 0);
    while(1) {
      inp[u = base[u]] = 1;
      if(u == root) break;
      u = f[ match[u] ];
    }
    while(1) {
      if(inp[v = base[v]]) return v;
      else v = f[ match[v] ];
    }
  }
  void mark(int lca, int u) {
    while(base[u] != lca) {
      int v = match[u];
      inb[ base[u ]] = 1;
      inb[ base[v] ] = 1;
      u = f[v];
      if(base[u] != lca) f[u] = v;
    }
  }
  void blossom_contraction(int s, int u, int v) {
    int lca = get_lca(s, u, v);
    fill(inb.begin(), inb.end(), 0);
    mark(lca, u); mark(lca, v);
    if(base[u] != lca) f[u] = v;
    if(base[v] != lca) f[v] = u;
    for(int u = 0; u < n; u++)
      if(inb[base[u]]) {
        base[u] = lca;
        if(!inq[u]) {
            inq[u] = 1;
            q.push(u);
        }
      }
  }
  int bfs(int s) {
    fill(inq.begin(), inq.end(), 0);
    fill(f.begin(), f.end(), -1);
    for(int i = 0; i < n; i++) base[i] = i;
    q = queue<int>();
    q.push(s);
    inq[s] = 1;
   while(q.size()) {
      int u = q.front(); q.pop();
      for(edge e = adj[u]; e; e = e->n) {
        int v = e->v;
        if(base[u] != base[v] && match[u] != v) {
          if((v == s) || (match[v] != -1 && f[match[v]]
              != -1))
            blossom_contraction(s, u, v);
          else if(f[v] == -1) {
            f[v] = u;
            if(match[v] == -1) return v;
            else if(!inq[match[v]]) {
              inq[match[v]] = 1;
              q.push(match[v]);
            }
          }
        }
      }
    }
    return -1;
  }
  int doit(int u) {
    if(u == -1) return 0;
    int v = f[u];
    doit(match[v]);
    match[v] = u; match[u] = v;
    return u != -1;
  }
  /// (i < net.match[i]) => means match
  int maximum_matching() {
    int ans = 0;
    for(int u = 0; u < n; u++)
      ans += (match[u] == -1) && doit(bfs(u));
    return ans;
  }
};
```

4.2 Dinic

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

4.3 Edmonds Karp

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

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

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

    return 0;
}

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

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

    return flow;
}
```

4.4 Hopcroft Karp

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

4.5 Maximum Bipartite Matching

```
// O(|E|*|V|)
struct mbm {
  int l, r;
  vector<vector<int>> g;
  vector<int> match, seen;
  mbm(int l, int r) : l(l), r(r), g(l), match(r), seen(r)
    {}
  void add_edge(int l, int r) { g[l].push_back(r); }
  bool dfs(int u) {
    for(auto v : g[u]) {
      if(seen[v]++) continue;
      if(match[v] == -1 || dfs(match[v])) {
        match[v] = u;
        return true;
      }
    }
    return false;
  }
  int max_matching() {
    int ans = 0;
    fill(match.begin(), match.end(), -1);
    for(int u = 0; u < l; ++u) {
      fill(seen.begin(), seen.end(), 0);
      ans += dfs(u);
    }
    return ans;
  }
  void matchs(){
    for (int i = 0; i<r; i++){
        if (match[i] == -1) continue;
        cout << match[i]+1  << ' ' << i+1 << ln;
    }
  }
};
```

4.6 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};
  }
};
```

4.7 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;
  }
};
```

4.8 Hungarian

```
const int N = 509;
/* Complexity: O(n^3) but optimized
It finds minimum cost maximum matching.
For finding maximum cost maximum matching
add -cost and return -matching()
1-indexed */
struct Hungarian {
  long long c[N][N], fx[N], fy[N], d[N];
  int l[N], r[N], arg[N], trace[N];
  queue<int> q;
  int start, finish, n;
  const long long inf = 1e18;
  Hungarian() {}
  Hungarian(int n1, int n2): n(max(n1, n2)) {
    for (int i = 1; i <= n; ++i) {
      fy[i] = l[i] = r[i] = 0;
      for (int j = 1; j <= n; ++j) c[i][j] = inf; // make
           it 0 for maximum cost matching (not necessarily
           with max count of matching)
    }
  }
  void add_edge(int u, int v, long long cost) {
    c[u][v] = min(c[u][v], cost);
  }
  inline long long getC(int u, int v) {
    return c[u][v] - fx[u] - fy[v];
  }
  void initBFS() {
    while (!q.empty()) q.pop();
    q.push(start);
    for (int i = 0; i <= n; ++i) trace[i] = 0;
    for (int v = 1; v <= n; ++v) {
      d[v] = getC(start, v);
      arg[v] = start;
    }
    finish = 0;
  }
  void findAugPath() {
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int v = 1; v <= n; ++v) if (!trace[v]) {
          long long w = getC(u, v);
          if (!w) {
            trace[v] = u;
            if (!r[v]) {
              finish = v;
              return;
            }
            q.push(r[v]);
          }
          if (d[v] > w) {
            d[v] = w;
            arg[v] = u;
          }
        }
    }
  }
  void subX_addY() {
    long long delta = inf;
    for (int v = 1; v <= n; ++v) if (trace[v] == 0 && d[v
       ] < delta) {
        delta = d[v];
      }
    // Rotate
    fx[start] += delta;
    for (int v = 1; v <= n; ++v) if(trace[v]) {
        int u = r[v];
        fy[v] -= delta;
        fx[u] += delta;
      } else d[v] -= delta;
    for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v])
       {
        trace[v] = arg[v];
        if (!r[v]) {
          finish = v;
          return;
        }
        q.push(r[v]);
      }
  }
  void Enlarge() {
    do {
      int u = trace[finish];
      int nxt = l[u];
      l[u] = finish;
      r[finish] = u;
      finish = nxt;
    } while (finish);
  }
  long long maximum_matching() {
    for (int u = 1; u <= n; ++u) {
      fx[u] = c[u][1];
      for (int v = 1; v <= n; ++v) {
        fx[u] = min(fx[u], c[u][v]);
      }
    }
    for (int v = 1; v <= n; ++v) {
      fy[v] = c[1][v] - fx[1];
      for (int u = 1; u <= n; ++u) {
        fy[v] = min(fy[v], c[u][v] - fx[u]);
      }
    }
    for (int u = 1; u <= n; ++u) {
      start = u;
      initBFS();
      while (!finish) {
        findAugPath();
        if (!finish) subX_addY();
      }
      Enlarge();
    }
    long long ans = 0;
    for (int i = 1; i <= n; ++i) {
      if (c[i][l[i]] != inf) ans += c[i][l[i]];
      else l[i] = 0;
    }
    return ans;
  }
};
```

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

pt normalize(pt p){
    lf norm = hypotl(p.x, p.y);
    if(fabsl(norm) > EPS) return {p.x /= norm, p.y /=
       norm};
    else return p;
}
int cmp(lf a, lf b){ return (a + EPS < b ? -1 :(b + EPS <
     a ? 1 : 0)); } // float comparator

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

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

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

lf arg(pt a){return atan2(a.y, a.x); } // ang(RAD) a x-
    pos
lf dot(pt a, pt b){ return a.x * b.x + a.y * b.y; } // x
    = 90 -> cos = 0
lf cross(pt a, pt b){ return a.x * b.y - a.y * b.x; } //
    x = 180 -> sin = 0
lf orient(pt a, pt b, pt c){ return cross(b - a, c - a);
    } // AB clockwise = -
int sign(lf x){ return (EPS < x) - (x < -EPS); }

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

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

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

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

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

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

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

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

5.2 Lineas

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

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

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

    lf evalx(lf x){
        assert(fabsl(v.x) > EPS);
        return (c + v.y * x) / v.x;
    }
};

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

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

5.3 Poligonos

```
// add Points Lines Segments Circles

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

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

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

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

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

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

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

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

// convex polygons in some order (CCW, CW)
vector<pt> minkowski(vector<pt> P, vector<pt> Q) {
        rotate(P.begin(), min_element(P.begin(), P.end())
           , P.end());
        rotate(Q.begin(), min_element(Q.begin(), Q.end())
           , Q.end());
        P.push_back(P[0]), P.push_back(P[1]);
        Q.push_back(Q[0]), Q.push_back(Q[1]);

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

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

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

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

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

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

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

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

// O(n)
vector<pair<int, int>> antipodal(vector<pt>& p){
    vector<pair<int, int>> ans;
    int n = p.size();
    if (n == 2) ans.push_back({0, 1});
    if (n < 3) return ans;
    auto nxt = [&](int x){ return (x + 1 == n ? 0 : x +
       1); };
    auto area2 = [&](pt a, pt b, pt c){ return cross(b -
       a, c - a); };
    int b0 = 0;
    while (abs(area2(p[n - 1], p[0], p[nxt(b0)])) > abs(
       area2(p[n - 1], p[0], p[b0]))) ++b0;
    for (int b = b0, a = 0; b != 0 && a <= b0; ++a) {
        ans.push_back({a, b});
        while (abs(area2(p[a], p[nxt(a)], p[nxt(b)])) >
           abs(area2(p[a], p[nxt(a)], p[b]))){
            b = nxt(b);
            if (a != b0 || b != 0) ans.push_back({a, b});
            else return ans;
        }
        if (abs(area2(p[a], p[nxt(a)], p[nxt(b)])) == abs
           (area2(p[a], p[nxt(a)], p[b]))){
            if (a != b0 || b != n - 1) ans.push_back({a,
               nxt(b)});
            else ans.push_back({nxt(a), b});
        }
    }
    return ans;
}
// O(n)
// square distance of most distant points, prereq: convex
   , ccw, NO COLLINEAR POINTS
lf callipers(vector<pt>& p){
    int n = p.size();
    lf r = 0;
    for(int i = 0, j = n < 2 ? 0 : 1; i < j; ++i){
        for(;;j = (j + 1) % n){
            r = max(r, norm2(p[i] - p[j]));
            if(cross((p[(i + 1) % n] - p[i]), (p[(j + 1)
               % n] - p[j])) <= EPS) break;
        }
    }

    return r;
}

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

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

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

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

5.4 Circulos

```
using namespace std;
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
const char ln = '\n';

#include "Points.cpp"
#include "Lines.cpp"
// add Lines Points

enum {OUT, IN, ON};

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

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

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

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

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

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

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

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

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

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

    lf td = 0.5L * ( d2 + c1.r * c1.r - c2.r * c2.r );
    lf h2 = c1.r * c1.r - td / d2 * td;

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

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

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

// circle-line inter = 1, inner: 1 = oxo 0 = o=o
```

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

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

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

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

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

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

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

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

  return det1 + det2 + det3 > E0;
}

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

                for(int j = 0; j < c.size(); ++j) if(j !=
                    i){
                        bool b0 = c[i].in(c[j]), b1 = c[j
                            ].in(c[i]);
                        if(b0 && (!b1 || i < j)) ++k;
                        else if(!b0 && !b1){
                                auto v = inter_cc(c[i], c
                                    [j]);
                                if(v.size() == 2){
                    swap(v[0], v[1]);
                                        p.push_back({v
                                            [0], 1});
                    p.push_back({v[1], -1});
                                        if(polar_cmp(v[1]
                                            - O, v[0] - O
                                           )) ++k;
                                }
                        }
                }
                sort(all(p), [&](auto& a, auto& b){
                    return polar_cmp(a.first - O, b.first
                    - O); });
                for(int j = 0; j < p.size(); ++j){
                        pt p0 = p[j ? j - 1 : p.size()
                           -1].first, p1 = p[j].first;
                        lf a = min_angle((p0 - c[i].
                           center), (p1 - c[i].center));
                        r[k] += (p0.x - p1.x) * (p0.y +
                           p1.y) / 2 + c[i].r * c[i].r *
                           (a - sin(a)) / 2;
                        k += p[j].second;
                }
        }
        return r;
}
```

5.5 Semiplanos

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

    bool out(const pt& r){ return cross(pq, r - p) < -EPS
       ;} // checks if p is inside the half plane
    bool operator < (const Halfplane& e) const { return
       angle < e.angle; }

};

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

// O(nlogn) return CCW polygon
vector<pt> hp_intersect(vector<Halfplane>& H) {
    pt box[4] = {pt(INF, INF), pt(-INF, INF), pt(-INF, -
       INF), pt(INF, -INF)};
    for(int i = 0; i < 4; ++i ) {
        Halfplane aux(box[i], box[(i + 1) % 4]);
        H.push_back(aux);
    }

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

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

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

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

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

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

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

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

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

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

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

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

5.6 Segmentos

```
// add Lines Points

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

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

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

// intersection bwn segments
set<pt> inter_ss(pt a, pt b, pt c, pt d) {
    pt out;
    if (proper_inter(a, b, c, d, out)) return {out}; // 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(norm(p - a), norm(p - b)); // otherwise distance to A or B
}

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

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

```
        return result;
    }
```

5.7 Convex Hull

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

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

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

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

        return r;
}
```

5.8 Closest Points

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

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

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

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

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

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

5.9 Min Circle

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

    auto f1 = [&]( int a ){
        Circle ans(v[a], 0.0L);
        for(int i = 0; i < a; ++i)
        if(ans.contains(v[i]) == OUT) ans = f2( i, a );
        return ans;
    };

    Circle ans( v[0], 0.0L );
    for(int i = 1; i < (int) v.size(); ++i)
        if(ans.contains(v[i]) == OUT ) ans = f1(i);

    return ans;
}
```

5.10 3D

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

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

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

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

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

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

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

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

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

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

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

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

5.11 KD Tree

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

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

        Node(vector<pt>&& vp) : pp(vp[0]){
                for(pt p : vp){
                        x0 = min(x0, p.x);
            x1 = max(x1, p.x);
                        y0 = min(y0, p.y);
            y1 = max(y1, p.y);
                }
                if(vp.size() > 1){
                        sort(all(vp), x1 - x0 >= y1 - y0
                            ? onx : ony);
                        int m = vp.size() / 2;
                        first = new Node({vp.begin(), vp.
                            begin() + m});
                        second = new Node({vp.begin() + m
                            , vp.end()});
                }
        }
};
struct KDTree {
        Node* root;
        KDTree(const vector<pt>& vp): root(new Node({all(
            vp)})){}
        pair<ll, pt> search(pt p, Node *node){
                if(!node->first){
                        // avoid query point as answer
                        // if(p.x == node->pp.x && p.y ==
                             node->pp.y) return {inf, pt()
                            };
                        return {norm2(p-node->pp), node->
                            pp};
                }
                Node *f = node->first, *s = node->second;
                ll bf = f->distance(p), bs = s ->
                    distance(p);
                if(bf > bs) swap(bf, bs), swap(f, s);
                auto best = search(p, f);
                if(bs < best.ff) best = min(best, search(
                    p, s));
                return best;
        }
        pair<ll, pt> nearest(pt p){ return search(p, root
            ); }
};
```

6 Grafos

6.1 Puentes

```
// O(n+m)
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

```
// O(n+m)
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.
const int maxn = 1e5+5;
vi adj_rev[maxn],adj[maxn];
bool used[maxn];
vi order,comp;

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

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

void init(int n){
        for(int i=0;i<n;++i)if(!used[i])dfs1(i);
        for(int i=0;i<n;++i)used[i]=false;
        reverse(order.begin(), order.end());
        for(int v:order){
                if(!used[v]){
                        dfs2(v);
                        // comp
                        comp.clear();
                }
        }
}

adj[a].push_back(b);
adj_rev[b].push_back(a);
```

6.4 Tarjan

```
// O(n+m) (?)
```

```
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

```
// 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

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

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

    return dist;
}
```

6.7 Floyd Warshall

```
// O(n^3)
vector<vi> adjMat(n+1, vi(n+1));
//Condicion previa: adjMat[i][j] contiene peso de la
   arista (i, j)
//o INF si no existe esa arista
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

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

6.9 MST Prim

```
// O(E * log V)
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& [to, len] : adj[v]) {
            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false;//ciclo negativo
                }
            }
        }
    }
    return true;
}
```

6.11 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)
// 1=(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 De Bruijn sequences

```
// Given alphabet [0, k) constructs a cyclic string
// of length k^n that contains every length n string as
   substr.
vi deBruijnSeq(int k, int n, int lim){
        if (k == 1) return {0};
        vi seq, aux(n + 1);
        int cont = 0;
        function<void(int,int)> gen = [&](int t, int p) {
                if (t > n){
                        if (n % p == 0) for(int i = 1; i
                            < p + 1; i++){
                                if (cont >= lim) return;
                                seq.pb(aux[i]);
                                cont++;
                        }
                } else {
                        aux[t] = aux[t - p];
                        gen(t + 1, p);
                        while (++aux[t] < k){
                                if (cont >= lim) return;
                                gen(t + 1, t);
                        }
                }
        };
        gen(1, 1);
        return seq;
}
```

7.2 Chinese Remainder Theorem

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

7.3 Totient y Divisores

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

7.4 Ecuaciones Diofanticas

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

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

7.5 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

```
// O(log n) muy rapido
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)
// O(iter*n^2)
int Freivalds(matrix &a, matrix &b, matrix &c) {
  int n = a.r, iteration = 20;
  matrix zero(n, 1), r(n, 1);
  while (iteration--) {
        for(int i = 0; i < n; i++) r.m[i][0] = rnd() % 2;
        matrix ans = (a * (b * r)) - (c * r);
        if(ans.m != zero.m) return 0;
  }
  return 1;
}
```

7.9 Gauss Jordan

```
// O(min(n, m)*n*m)
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

```
const int steps = 1e6; // %2==0
double f(double x);
double simpson(double a, double b){
        double h=(b-a)/steps;
        double s=f(a)+f(b);
        for(int i=1;i<=steps-1;i++){
                double x=a+h*i;
                s+=f(x)*((i&1)?4:2);
        }
        s*=h/3;
        return s;
}
```

7.13 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);
}

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

7.14 Logaritmo Discreto

```
// O(sqrt(m))
// 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), O(c*b) - variables c, restricciones b
struct Simplex{
    vector<vector<double>> A;
    vector<double> B,C;
    vector<int> X,Y;
    double z;
    int n,m;

    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.;
        X=vector<int>(m);Y=vector<int>(n);
        for(int i=0;i<m;++i)X[i]=i;
        for(int i=0;i<n;++i)Y[i]=i+m;
    }

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

    pair<double, vector<double>> maximize(){
        while(1){
            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;}
            // y<0, no solution to Ax<=B
            pivot(x,y);
        }
        while(1){
            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;
            // 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

```
// O(N log N)
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 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;
}
```

8.2 CHT

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

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

    struct Line {
        int slope, yIntercept;

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

    deque<pair<Line, int>> dq;

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

    int query(int x) { // cuando las consultas son
       crecientes
        while (dq.size() > 1) {
            if (dq[1].second <= x) dq.pop_front();
            else break;
        }
        return dq[0].first.val(x);
    }

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

        return qry.first.val(x);
    }
};
```

8.3 CHT Dynamic

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

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

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

8.4 Divide Conquer

```
// C[a][c] + C[b][d] <= C[a][d] + C[b][c] where a < b < c
     < d.
int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

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

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

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

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

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

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

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

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

8.5 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 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);
            }
        }
    }
}
```

8.7 Knuth

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

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

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

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

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

8.8 LIS

```
// O(nlogn)
int lis(vi& a){
        int n=sz(a),last=0;
        vi dp(n+1,INT_MAX),cnt(n,0);
        dp[0]=INT_MIN;
        for(int i=0;i<n;++i){
                int j=lower_bound(all(dp), a[i])-dp.begin
                   (); // upper_bound
                if(dp[j-1]<a[i] && a[i]<dp[j]){ // dp[j
                   -1]<=a[i]
                        dp[j]=a[i];
                        last=max(last,j);
                }
                cnt[i]=j;
        }
        int ans=0;
        for(int i=0;i<=n;i++){
                if(dp[i]<INT_MAX)ans=i;
        }
        vi LIS(ans);
        int act=ans;
        for(int i=n-1;i>=0;--i){
                if(cnt[i]==act){
                        LIS[act-1]=a[i];
                        act--;
                }
        }
        return ans;
}
```

8.9 SOS

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

// O(n*2^n)
```

```
void SOS(){
        for(int i = 0; i < (1 << bits); ++i) dp[i] = A[i
            ];
        // top - down
        for(int i = 0; i < bits; ++i){
                for(int s = 0; s < (1 << bits); ++s){
                        if(s & (1 << i)){
                                dp[s] += dp[s ^ (1 << i)
                                    ];
                        }
                }
        }
        // bottom - up
        for(int i = 0; i < bits; ++i){
                for(int s = (1 << bits) - 1; s >= 0; --s)
                    {
                        if(s & (1 << i)){
                                dp[s ^ (1 << i)] += dp[s
                                    ];
                        }
                }
        }
}
```

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 m){return a+b>=m?a+b-m:a
    +b;}
inline int sbt(int a, int b, int m){return a-b<0?a-b+m:a-
    b;}
inline int mul(int a, int b, int m){return ll(a)*b%m;}
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;
}

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

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 to[maxn][alpha];
```

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

9.4 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& s2, 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=s2;

                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;

        SuffixArray(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 Dates

```
int dateToInt(int y, int m, int d){
        return 1461*(y+4800+(m-14)/12)/4+367*(m-2-(m-14)
            /12*12)/12-
                3*((y+4900+(m-14)/12)/100)/4+d-32075;
}
void intToDate(int jd, int& y, int& m, int& d){
        int x,n,i,j;x=jd+68569;
        n=4*x/146097;x-=(146097*n+3)/4;
        i=(4000*(x+1))/1461001;x-=1461*i/4-31;
        j=80*x/2447;d=x-2447*j/80;
        x=j/11;m=j+2-12*x;y=100*(n-49)+i+x;
}
int DayOfWeek(int d, int m, int y){      //starting on
    Sunday
        static int ttt[]={0, 3, 2, 5, 0, 3, 5, 1, 4, 6,
            2, 4};
        y-=m<3;
        return (y+y/4-y/100+y/400+ttt[m-1]+d)%7;
}
```

10.3 Expression Parsing

```
// O(n) - 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();
}
```

10.4 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.
- $\text{Cat}(n)$ 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