Estufa en Piloto

Contents

1 General

1.1 Plantilla

```cpp
/*   AUTHOR: Estufa en Piloto   */
#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
#define DBG(x) cerr << #x << " = " << (x) << endl
#define RAYA cerr << "==============================" << endl
#else
#define DBG(x)
#define RAYA
#endif

typedef long long ll;
typedef vector<ll> vi; typedef pair<ll,ll> ii;
typedef vector<ii> vii; typedef vector<bool> vb;
#define FIN ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define forr(i, a, b) for(ll i = (a); i < (ll) (b); i++)
#define forn(i, n) forr(i, 0, n)
#define SZ(x) int((x).size())
#define pb push_back
#define mp make_pair
#define all(c) (c).begin(),(c).end()
#define esta(x,c) ((c).find(x) != (c).end())
const int INF = 1<<30; // const ll INF = (1LL<<60);
const int MOD = 1e9+7;  // const int MOD = 998244353;
const int MAXN  = 2e5+5;

int main(){
    FIN;



    return 0;
}
```

1.2 Compilar

```
# Llenar lo siguiente en el build de geany

```

```
# Compile
g++ -std=c++17 -DLOCAL -g -O2 -Wconversion -Wshadow -Wall -Wextra -
    D_GLIBCXX_DEBUG -c %f

# Build
g++ -std=c++17 -DLOCAL -g -O2 -Wconversion -Wshadow -Wall -Wextra -
    D_GLIBCXX_DEBUG -o %e %f
```

1.3 Informacion

```

                DATA STRUCTURE - C++


---------------------- VECTOR -----------------------

vector <ll> v; //DECLARACION DEL VECTOR, TAMANO 0.

vector <ll> v(n); //DECLARACION DEL VECTOR DE TAMANO N

v.push_back(n); //PUSHEA N AL FINAL DEL VECTOR, O(1)

sort(v.begin(),v.end()); //ORDENA EL VECTOR, O(n.logn)

v.pop_back(); //ELIMINA EL ULTIMO VALOR DEL VECTOR, O(1);

v.size(); //RETORNA EL TAMANO DEL VECTOR

v.clear(); //BORRA TODO EL VECTOR

v.resize(n); //ESTABLECEMOS EL TAMANO DEL VECTOR A TAMANO N

vector<pair<ll,ll>> v; //VECTOR DE PAIR

v.push_back(make_pair(a,b)); //PUSHEA EL PAR (A,B).

v[i].first; //RETORNA EL PRIMER VALOR DEL PAIR

v[i].second; //RETORNA EL SEGUNDO VALOR DEL PAIR

for(auto u : v) //RECORRE TODOS LOS VALORES DEL VECTOR

BUENO PARA STACKEAR COSAS, PARA USARLO COMO LISTA DE ADYACENCIAS
EN LOS PROBLEMAS DE GRAFOS, PARA USARLOS COMO ARREGLO.
SI SE ORDENA UN VECTOR DE PAIR, LOS ORDENA DE MENOR A MAYOR SEGUN
EL PRIMER VALOR, Y LUEGO DE MENOR A MAYOR SEGUN EL SEGUNDO.

-----------------------------------------------------


---------------------- SET ------------------------

set <int> s; //DECLARACION DEL SET, EL SET GUARDA LOS ELEMENTOS UNA SOLA
       //VEZ, Y MIENTRAS LOS VA GUARDANDO, LOS VA ORDENANDO.

multiset <int> s; //DECLARACION DEL MULTISET. EL MULTISET PUEDE GUARDAR
          //VARIAS COPIAS DEL MISMO NUMERO, Y MIENTRAS LOS VA
          //GUARDANDO, LOS VA ORDENANDO.

s.insert(n); //INSERTA EL VALOR N EN EL SET, SI YA ESTABA INSERTADO, NO
       //NO SE MODIFICA NADA. O(logn)

s.erase(n); //ELIMINA EL VALOR N DEL SET, O(log n)

s.size(); //TAMANO DEL SET

s.clear(); //BORRA TODOS LOS ELEMENTOS DEL SET O(s.size())

auto it=s.begin(); //DECLARACION DEL ITERATOR IT PARA EL SET

it=s.find(n); //HALLA LA POSICION DEL VALOR N DENTRO DEL SET, SI NO
        //CREO QUE RETORNA s.end(); (FINAL DEL SET)

it.lower_bound(n); //HERRAMIENTA FUERTISIMA DEL SET. CON BUSQUENA
        //BINARIA OBTIENE EL PRIMER VALOR DEL SET QUE ES MAYOR O IGUAL
        //A N, O(logn)

it=upper_bound(n); //HERRAMIENTA TAMBIEN MUY FUERTE, PERO NO LA SUELO
          //USAR MUCHO, SUELE PODER SER REEMPLAZADA POR EL LOWER_BOUND
          //EN O(logn) DEVUELVE EL PRIMER VALOR MAYOR ESTRICTO QUE
          //N

for(auto it : s) //RECORRE TODOS LOS VALORES DEL SET

// Custom comparators
struct Edge {
```

```
  int a, b, w;
};
struct cmp { // Funcion comparadora para el set
  bool operator()(const Edge &x, const Edge &y) const { return x.w < y.w
      ; }
};

int main() {
  int M = 4;
  set<Edge, cmp> v;
  ...

// FACIL DE USAR. PUEDE REEMPLAZAR EL PRIORITY QUEUE PERO ES UN POQUITO
    MAS
// LENTO. BUENO PARA PROBLEMA QUE CONVIENE IR ORDENANDO EN TIEMPO REAL,
// EN VEZ DE LEER TODO Y LUEGO ORDENARLO. SE USA MUCHO EN ALGORITMOS
    PARA
// GRAFOS COMO DIJKSTRA Y PRIM.


-----------------------------------------------------------------

----------------------- QUEUE -----------------------------

queue <int> q; //DECLARACION DEL QUEUE

q.push(n); //PUSHEA AL FINAL DEL QUEUE EL NUMERO N

q.front(); //OBTIENE EL NUMERO QUE SE ENCUENTRA PRIMERO EN LA QUEUE

q.pop() //BORRA EL NUMERO QUE SE ENCUENTRA PRIMERO EN EL QUEUE

-----------------------------------------------------------------

--------------------- PRIORITY QUEUE -------------------------

prority_queue <int> pq; //DECLARACION DEL PRIORITY_QUEUE. A MEDIDA QUE
            //VAMOS INSERTANDO VALORES, LOS VA ORDENANDO EN
            //ORDEN CRECIENTE.

priority_queue<int, vector<int>, greater<int>> pq; // PRIORITY_QUEUE DE
    MINIMO

pq.push(n); //INSERTA EL VALOR N EN EL LUGAR QUE CORRESPONDE. O(log n)

pq.top(); //OBTIENE EL VALOR QUE SE ENCUENTRA PRIMERO EN LA PQ.

pq.pop(); //ELIMINA EL ELEMENTO QUE SE ENCUENTRA PRIMERO EN LA PQ.
```

1.4 Medir Tiempo

```
unsigned t0, t1;
t0=clock();

// Aca Va el codigo

t1 = clock();
double time = (double(t1-t0)/CLOCKS_PER_SEC);
cout << "Execution␣Time:␣" << time << endl;
```

1.5 Plantilla Old

```
#include <bits/stdc++.h>
using namespace std;
//freopen("input.txt", "r", stdin);
//freopen("output.txt", "w", stdout);

// neal Debugger
template<typename A, typename B> ostream& operator<<(ostream &os, const
    pair<A, B> &p) { return os << '(' << p.first << ",␣" << p.second <<
    ')'; }
template<typename T_container, typename T = typename enable_if<!is_same<
    T_container, string>::value, typename T_container::value_type>::type
    > ostream& operator<<(ostream &os, const T_container &v) { os << '{'
    ; string sep; for (const T &x : v) os << sep << x, sep = ",␣";
    return os << '}'; }

void dbg_out() { cerr << endl; }
template<typename Head, typename... Tail> void dbg_out(Head H, Tail... T
    ) { cerr << '␣' << H; dbg_out(T...); }

#ifdef LOCAL
#define dbg(...) cerr << "(" << #__VA_ARGS__ << "):", dbg_out(
    __VA_ARGS__)
#else
#define dbg(...)
#endif
```

```
18
19 typedef long long ll;
20 #define FIN ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
21 #define forr(i, a, b) for(ll i = (a); i < (ll) (b); i++)
22 #define forn(i, n) forr(i, 0, n)
23 #define pb push_back
24 #define mp make_pair
25 #define all(c) (c).begin(),(c).end()
26 #define DBG(x) cerr << #x << "␣=␣" << (x) << endl
27 #define DBGV(v,n) forn(i,n) cout << v[i] << "␣"; cout << endl
28 #define esta(x,c) ((c).find(x) != (c).end())
29 #define RAYA cerr << "==============================" << endl
30 const ll MOD = (ll)(1e9+7); // 998244353
31 const ll INF = (ll)(1<<30); // (1LL<<60)
32 const int MAXN = (int)(2e5+5);
33
34
35 int main(){
36   FIN;
37
38
39   return 0;
40 }
```

2 Data Structures

2.1 Compresion De Coordenadas

```
1  // Compresion de coordenadas
2  // Complejidad: O(n log n) para construir, O(log n) para obtener
3  struct compresion {
4      vector<int> todos;
5      compresion(vector<int> v) {
6          todos = v; sort(all(todos));
7          todos.erase(unique(all(todos)),todos.end());
8      }
9      int obtener(int x) { // Esto se podria hacer tambien con un
           unordered_map en O(1)
10         return (int)(lower_bound(all(todos),x)-todos.begin());
11     }
12 };
```

2.2 Dsu

```
1  // DSU struct con path compression y union por size
2  // Complejidad: O(ack(n)) por operacion, donde ack(n) es la funcion
       inversa de Ackermann, casi O(1)
3  struct DSU {
4      vi link, sz;
5
6      DSU(int tam) {
7          link.resize(tam+5), sz.resize(tam+5);
8          forn(i, tam+5) link[i] = i, sz[i] = 1;
9      }
10
11     ll find(ll x){ return link[x] = (link[x] == x ? x : find(link[x]));
           }
12     bool same(ll a, ll b) { return find(a) == find(b); }
13
14     void join(ll a, ll b) {
15         a = find(a), b = find(b);
16         if(a == b) return;
17         if(sz[a] < sz[b]) swap(a,b);
18         sz[a] += sz[b];
19         link[b] = a;
20     }
21 };
```

2.3 Fenwick

```cpp
// Description: Fenwick Tree (BIT) for range queries and point updates
// Time: O(log n) for both queries and updates
// Usage: BIT bit(v); bit.query(l,r); bit.update(pos,val);
struct BIT {
    vector <ll> prefix, a;
    BIT(vector <ll> &v) {
        int n = v.size(); prefix.resize(n+1); a = v;
        vector <ll> aux(n+1,0);
        forn(i,n) aux[i+1] = aux[i] + v[i];
        forr(i,1,n+1) prefix[i] = aux[i] - aux[i - (i&(-i))];
    }
    ll query(int l, int r) { //[a,b] 0-indexed
        ll ans = 0; r++;
        while(r) ans += prefix[r], r -= r&(-r);
        while(l) ans -= prefix[l], l -= l&(-l);
        return ans;
    }
    void update(int pos, ll val) {
        int i = pos + 1; ll upd = val - a[pos];
        while(i < prefix.size()) prefix[i] += upd, i += i&(-i);
        a[pos] = val;
    }
};
```

2.4 Implicit Treap

```cpp
//~ Puede realizar todas las siguientes operaciones en O(log N)

//~ Dividir el arreglo en dos
//~ Mergear dos arreglos en uno
//~ Insertar elemento en cualquier posicion
//~   Si queremos intertar 'x' en la posicion i-esima, spliteamos en
//~   dos arreglos L = [0,i-1] : R = [i,n]; y mergeamos T = (L,x) y
//~   T = (T,R)
//~ Eliminar elemento en cualquier posicion
//~   Si queremos eliminar el i-esimo elemento, spliteamos en tres
//~   arreglos L = [0,i-1], M = [i,i], R = [i+1,n] y mergeamos T = (L,R)
//~ Aplicar cualquier funcion de Segment Tree o Lazy
//~   Para realizar la query [l,r], tenemos que primero splitear y
//~   obtener el intervalo [l,r]. El valor de su raiz sera la respuesta.
//~   Luego mergeamos para volver a la normalidad.

typedef struct item *pitem;

struct item {
    ll key, prior, cont, mini, sum;
    bool rev; // (parameters for lazy prop)
    pitem l, r;
    item(ll key):
        key(key),prior(rand()),cont(1),l(NULL),r(NULL),rev(0) {}
};

void push(pitem it) { //Lazy for reverse array
    if(it && it->rev) {
        swap(it->l,it->r);
        if(it->l)it->l->rev ^= true;
        if(it->r)it->r->rev ^= true;
        it->rev = false;
    }
}

int cont(pitem it) {return it ? it->cont : 0;}
ll mini(pitem it) {return it ? it->mini : 1<<30;}
ll sum(pitem it) {return it ? it->sum : 0LL;}

void upd_cont(pitem it){
    if(it) {
        it->cont = cont(it->l) + cont(it->r) + 1;
        it->mini = min(it->key,min(mini(it->l),mini(it->r)));
        it->sum = it->key + sum(it->l) + sum(it->r);
    }
}

void merge(pitem &t, pitem l, pitem r) { //Merge l and r, new root t
    push(l); push(r);
    if(!l || !r) t = l ? l : r;
    else if(l->prior > r->prior) merge(l->r,l->r,r), t=l;
    else merge(r->l,l,r->l), t = r;
    upd_cont(t);
}

void split(pitem t, pitem& l, pitem& r, int sz){ //sz:desired size of l
    if(!t) { l = r = 0; return;}
    push(t);
```

```
    if(sz <= cont(t->l)) split(t->l,l,t->l,sz), r = t;
    else split(t->r,t->r,r,sz-1-cont(t->l)), l = t;
    upd_cont(t);
}

int find_min(pitem t) { //Devuelve la posicion del minimo
    push(t);
    if(t->mini == t->key) return cont(t->l);
    else if(t->l && t->l->mini == t->mini) return find_min(t->l);
    else return cont(t->l)+1+find_min(t->r);
}

void set_treap(pitem &root, int p, ll val) {
    pitem aux1 = NULL, aux2 = NULL;
    split(root,root,aux2,p+1); split(root,root,aux1,p);
    aux1->key = val;
    merge(root,root,aux1); merge(root,root,aux2);
}

ll get_sum(pitem &root, int l, int r) {
    pitem aux1 = NULL, aux2 = NULL;
    split(root,root,aux2,r+1); split(root,root,aux1,l);
    ll ans = aux1->mini; // aux1->mini for minimum
    merge(root,root,aux1); merge(root,root,aux2);
    return ans;
}

void DFS(pitem t) { //Good for debug
    DBG(t->key);
    if(t->l != NULL) DBG(t->l->key); if(t->r != NULL) DBG(t->r->key);
    RAYA;
    if(t->l != NULL) DFS(t->l); if(t->r != NULL) DFS(t->r);
}
```

2.5 Indexed Set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
typedef long long ll;

using namespace __gnu_pbds;
using namespace std;

typedef tree<ll,null_type,less<ll>,rb_tree_tag,
    tree_order_statistics_node_update> indexed_set;

int main() {
    indexed_set s;
    s.insert(2); s.insert(3); s.insert(5);
    auto x=s.find_by_order(2); // retorna el valor en la posicion 2 (5)
        (se puede desreferenciar con *)
    int pos=int(s.order_of_key(3)); // retorna la posicion que se
        encuentra el 3 (1)
    return 0;
}
```

2.6 Indexed Set Casero

```
// Indexed Set (Treap) - O(log n) for both queries and updates
typedef struct item *pitem;

struct item {
    ll key, prior, cont, mini, sum;
    bool rev; // (parameters for lazy prop)
    pitem l, r;
    item(ll key):
        key(key),prior(rand()),cont(1),l(NULL),r(NULL),rev(0) {}
};

void push(pitem it) { //Lazy for reverse array
    if(it) {
        if(it->rev) {
            swap(it->l,it->r);
            if(it->l)it->l->rev ^= true;
            if(it->r)it->r->rev ^= true;
            it->rev = false;
        }
    }
}

int cont(pitem it) {return it ? it->cont : 0;}
ll mini(pitem it) {return it ? it->mini : 1<<30;}
ll sum(pitem it) {return it ? it->sum : 0LL;}

void upd_cont(pitem it){
    if(it) {
```

```
        it->cont = cont(it->l) + cont(it->r) + 1;
        it->mini = min(it->key,min(mini(it->l),mini(it->r)));
        it->sum = it->key + sum(it->l) + sum(it->r);
    }
}

void merge(pitem &t, pitem l, pitem r) { //Merge l and r, new root t
    push(l); push(r);
    if(!l || !r) t = l ? l : r;
    else if(l->prior > r->prior) merge(l->r,l->r,r), t=l;
    else merge(r->l,l,r->l), t = r;
    upd_cont(t);
}

void split(pitem t, pitem& l, pitem& r, int sz){ //sz:desired size of l
    if(!t) { l = r = 0; return;}
    push(t);
    if(sz <= cont(t->l)) split(t->l,l,t->l,sz), r = t;
    else split(t->r,t->r,r,sz-1-cont(t->l)), l = t;
    upd_cont(t);
}

int treap_lower_bound(pitem &t, ll val) { // retorna cantidad de
    elementos menores estrictos
    if(t == NULL) return 0;
    if(t->key >= val) return treap_lower_bound(t->l,val);
    return cont(t->l) + 1 + treap_lower_bound(t->r,val);
}

int find_by_order(pitem &t, int k) { // retorna el k-esimo elemento
    if(t == NULL) return -1;
    if(cont(t->l) + 1 == k) return t->key;
    if(cont(t->l) + 1 < k) return find_by_order(t->r,k-cont(t->l)-1);
    else return find_by_order(t->l,k);
}

void insert(pitem &t, ll val) {
    int tam = treap_lower_bound(t,val);
    pitem r = NULL;
    split(t,t,r,tam);
    merge(t,t,new item(val)); merge(t,t,r);
}

void erase(pitem &t, ll val) {
    int tam = treap_lower_bound(t,val);
    pitem r = NULL, aux = NULL;
    split(t,t,r,tam+1); split(t,t,aux,tam);
    merge(t,t,r);
}

void DFS(pitem t) { //Good for debug
    if(t == NULL) return;
    DFS(t->l); cerr << t->key << "␣"; DFS(t->r);
}
```

2.7 Linked List

```
// Linked list based on arrays
struct linked_list {
    ll n;
    vi l, r;
    linked_list(int n): n(_n) {
        l.resize(n); r.resize(n);
        forn(i, n){
            l[i] = (i-1+n)%n;
            r[i] = (i+1)%n;
        }
    }
    void erase(int pos){
        r[l[pos]] = r[pos];
        l[r[pos]] = l[pos];
    }
};
```

2.8 Mo Dquery

```
// Mo's Algorithm for range queries
// Time: O(n*sqrt(n))
// Usage: mo mo; mo.read_queries(m); mo.get_mo(v);

struct query{
    int l, r, ind;
    bool operator <(query q) const {
        return r < q.r;
    }
};
```

```cpp
struct mo {
    const int block = 448; // sqrt(MAXN)
    vector<vector<query>> q;
    vector<ll> ans, cont;

    mo(){
        ans.resize(MAXN), cont.resize(MAXN);
        q.resize(MAXN/block+5);
    }

    void read_queries(int m) { // m --> number of queries
        ans.resize(m,0);
        forn(i, m) {
            int l, r; cin >> l >> r;
            l--; r--; // For 0-indexed
            q[l / block].pb({l,r,(int)i});
        }
    }

    void add(ll &sum, ll val) {
        if(cont[val] == 0) sum++;
        cont[val]++;
    }

    void erase(ll &sum, ll val) {
        if(cont[val] == 1) sum--;
        cont[val]--;
    }

    void get_mo(vector<ll> &v) {
        ll sum = 0;
        forn(i,q.size()) {
            sort(all(q[i]));
            int l, r; l = r = block * i;
            for(query u : q[i]) { // Solve for [l,r]
                while(r <= u.r) add(sum,v[r]), r++;
                while(l <= u.l) erase(sum,v[l]), l++;
                while(l > u.l) l--, add(sum,v[l]);
                ans[u.ind] = sum;
            }
            while(l < r) erase(sum,v[l]), l++;
        }
    }
};
```

2.9 Segtree 2D

```cpp
struct st2d{ // 0 indexado, [l, r]
  // PARA HACER BUILD HAY QUE HACER LOS N*N updates
  int n;
  vector<vi> st;
  ll NEUT=0; // NEUTRO DE LA OPERACION
  ll op(ll a, ll b){return a+b;} // OPERACION

  st2d(int _n): n(_n) { st.resize(2*n+5, vi(2*n+5, 0)); }

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

  ll query(int x0, int x1, int y0, int y1){
    ll r=NEUT; x1++, y1++;
    for(int i0=x0+n,i1=x1+n;i0<i1;i0>>=1,i1>>=1){
      int t[4],q=0;
      if(i0&1)t[q++]=i0++;
      if(i1&1)t[q++]=--i1;
      forn(k,q) for(int j0=y0+n,j1=y1+n;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;
  }
};
```

2.10 Segtree Point Query

```cpp
typedef ll tipo;
struct segtree {
  vector<tipo> t; int tam;
  tipo NEUT = 0;
  tipo op(tipo a, tipo b){ return a + b; }

  void build(vector<tipo> v, int n) { // build the tree
```

```
      // root en 1, ojas en el intervalo [tam, 2*tam-1]
      tam = sizeof(int) * 8 - __builtin_clz(n);
      tam = 1 << tam; //Primera potencia de 2 mayor a n
      t.resize(2 * tam, 0); forn(i, n) t[tam+i] = v[i];
      for(int i = tam - 1; i > 0; i--)
          t[i] = op(t[i << 1], t[i << 1 | 1]); //Poner la operacion del
              seg
  }

  void modify(int l, int r, tipo value) {
    for (l += tam, r += tam; l <= r; l >>= 1, r >>= 1) {
      if (l&1) t[l] = op(t[l], value); //Poner la operacion del seg
      if (!(r&1)) t[r] = op(t[r], value); //Poner la operacion del seg
      l++, r--;
    }
  }

  tipo query(int p) {
    tipo res = NEUT;
    for (p += tam; p > 0; p >>= 1) res = op(res, t[p]); //Operacion del
        seg
    return res;
  }
};
```

2.11 Segtree Range Query

```
typedef long long tipo;
struct segtree {
  vector <tipo> t; int tam;
  tipo NEUT = 0; // Neutral element of operation
  tipo op(tipo a, tipo b){ return a+b; } // Operation to make

  void build(vector<tipo>v, int n) {  // build the tree
    // root en 1, ojas en el intervalo [tam, 2*tam-1]
    tam = sizeof(int) * 8 - __builtin_clz(n); tam = 1<<tam;
    t.resize(2*tam,NEUT); forn(i, n) t[tam+i] = v[i];
    for(int i = tam - 1; i > 0; i--) t[i] = op(t[i<<1], t[i<<1|1]);
  }

  void update(int p, tipo value) {  // set value at position p
    for (t[p += tam] = value; p > 1; p >>= 1) t[p>>1] = op(t[p], t[p^1])
        ;
    //Tener cuidado aca, podria ser que en cosas no conmutativas, para p
        impar: p > p^1.
  }

  tipo query(int l, int r) {  // op on interval [l, r]
    tipo res = NEUT;
    for (l += tam, r += tam; l <= r; l >>= 1, r >>= 1) {
      if (l&1) res = op(res, t[l++]);
      if (!(r&1)) res = op(res, t[r--]);
    }
    return res;
  }
};
```

2.12 Find Kth Min

```
struct Vertex {
    Vertex *l, *r;
    ll sum;

    Vertex(ll val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};

Vertex* build(vector <int> &a, ll tl, ll tr) {
    if (tl == tr) return new Vertex(a[tl]);
    ll tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}

ll get_sum(Vertex* v, ll tl, ll tr, ll l, ll r) {
    if (l > r) return 0;
    if (l == tl && tr == r) return v->sum;
    ll tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}

Vertex* update(Vertex* v, ll tl, ll tr, ll pos, ll new_val) {
    if(tl == tr) return new Vertex(new_val);
    ll tm = (tl + tr) / 2;
    if(pos <= tm) return new Vertex(update(v->l, tl, tm, pos, new_val),
        v->r);
    else return new Vertex(v->l, update(v->r, tm+1, tr, pos, new_val));
}

void compresion(vector <int> ValoresQueAparecen, vector <int> &todos) {
  for(int x : ValoresQueAparecen) todos.pb(x);
  sort(todos.begin(),todos.end());
  todos.erase(unique(todos.begin(),todos.end()),todos.end());
}

// Retorna el valor de f(x)

int obtener(vector <int> &todos, int x) {
  return (int)(lower_bound(todos.begin(),todos.end(),x)-todos.begin());
}

struct kth_min {
  int n;
  vector <int> v, todos, v_comp, real_num;
  vector <Vertex*> estado;
  void read(int z) {
    n = z;
    v.resize(n); v_comp.resize(n); real_num.resize(n);
    forn(i,n) cin >> v[i];
  }
  void prepare() {
    compresion(v,todos);
    forn(i,n) v_comp[i] = obtener(todos,v[i]);
    forn(i,n) real_num[v_comp[i]] = v[i];
    vector <int> cont(n,0);
    estado.pb(build(cont,0,n-1));
    for(int u : v_comp) {
      cont[u]++;
      estado.pb(update(estado.back(),0,n-1,u,cont[u]));
    }
  }
  int find_kth(int l, int r, int k) {
    //tl and tr --> 0 indexed, k --> 1 indexed
    Vertex* posl = estado[l];
    Vertex* posr = estado[r+1];

    int tl = 0, tr = n-1;
    while(tl!=tr) {
      int tm = (tr + tl)/2;
      if(posr->l->sum - posl->l->sum >= k) {
        posr = posr->l;
        posl = posl->l;
        tr = tm;
      }
      else {
        k -= posr->l->sum - posl->l->sum;
        posr = posr->r;
        posl = posl->r;
        tl = tm+1;
      }
```

```
    }
    return real_num[tl];
  }
};
```

2.13 Lazy Aritmetic Sum

```
typedef long long tipo;

struct node {
  tipo ans, l, r, a=0, d=0;
  bool upd;
  node() {ans = a = d = 0;}
  node(tipo val, int pos) {ans = val; l = r = pos;}
  void set_lazy(tipo _a, tipo _d, tipo start) {
    upd=true, a += (l-start)*_d + _a; d += _d;
  }
};

struct lazy {

  #define l(x) int(x<<1)
  #define r(x) int(x<<1|1)

  vector <node> t; int tam;

  node op(node a, node b) { //Operacion de query
        node aux; aux.ans = a.ans + b.ans;
    aux.l = a.l; aux.r = b.r;
        return aux;
    }

  void push(tipo p) {
    if(t[p].upd == true) {
      tipo d = t[p].r - t[p].l + 1;
      t[p].ans += d * t[p].a;
      t[p].ans += t[p].d * (d-1)*(d)/2;
      if(t[p].l < t[p].r) {
        t[l(p)].set_lazy(t[p].a,t[p].d,t[p].l);
        t[r(p)].set_lazy(t[p].a,t[p].d,t[p].l);
      }
      t[p].a = 0; t[p].d = 0; t[p].upd = false;
    }
  }

  node query(tipo l, tipo r, int p = 1) {
    push(p);
    if(l > t[p].r || r < t[p].l) return node();
    if(l <= t[p].l && t[p].r <= r) return t[p];
    return op(query(l,r,l(p)),query(l,r,r(p)));
  }

  void update(tipo l, tipo r, tipo a, tipo d, int p = 1) {
    push(p); node &cur = t[p];
    if(l > cur.r || r < cur.l) return;
    if(l <= cur.l && cur.r <= r) {
      cur.set_lazy(a,d,l); push(p); return;
    }
    update(l, r, a, d, l(p)); update(l, r, a, d, r(p));
    cur = op(t[l(p)], t[r(p)]);
  }

  void build(vector<tipo> v, int n) { // iterative build
        tam = sizeof(int) * 8 - __builtin_clz(n); tam = 1<<tam;
        t.resize(2*tam); v.resize(tam);
        forn(i,tam) t[tam+i] = node(v[i],int(i));
        for(int i = tam - 1; i > 0; i--) t[i] = op(t[l(i)],t[r(i)]);
    }
};
```

2.14 Lazy Extreme

```
typedef long long tipo;
const int NEUT = 0; // REMINDER !!!

struct node {
  tipo ans, l, r, lazy = 0;
  bool upd = false;
  node() {ans = lazy = 0; upd = false; l = r = -1;} // REMINDER !!! SET
      NEUT
  node(tipo val, int pos) : ans(val), l(pos), r(pos) {} // Set node
  void set_lazy(tipo x) {lazy += x; upd = true;}
};

struct segtree_lazy {
  #define l(x) int(x<<1)
```

```
#define r(x) int(x<<1|1)

vector <node> t; int tam;

node op(node a, node b) {
  node aux; aux.ans = a.ans + b.ans; //Operacion de query
  aux.l = a.l; aux.r = b.r;
  return aux;
}

void push(int p) {
  node &cur = t[p];
  if(cur.upd == true) {
    cur.ans += cur.lazy * (cur.r-cur.l+1); //Operacion update
    if(cur.l < cur.r) {
      t[l(p)].lazy += cur.lazy; t[l(p)].upd = true;
      t[r(p)].lazy += cur.lazy; t[r(p)].upd = true;
    }
    cur.lazy = 0; cur.upd = false; //Poner el neutro del update
  }
}

node query(int l, int r, int p = 1) {
  push(p); node &cur = t[p];
  if(l > cur.r || r < cur.l) return node(); // Return NEUT
  if(l <= cur.l && cur.r <= r) return cur;
  return op(query(l,r,l(p)),query(l,r,r(p)));
}

void update(int l, int r, tipo val, int p = 1) { // root at p = 1
  push(p); node &cur = t[p];
  if(l > cur.r || r < cur.l) return;
  if(l <= cur.l && cur.r <= r) {
    cur.set_lazy(val); push(p); return;
  }
  update(l, r, val, l(p)); update(l, r, val, r(p));
  cur = op(t[l(p)], t[r(p)]);

}

void build(vector <tipo> v, int n) { // iterative build
  tam = sizeof(int) * 8 - __builtin_clz(n); tam = 1<<tam;
  t.resize(2*tam); v.resize(tam);
  forn(i,tam) t[tam+i] = node(v[i],i);
  for(int i = tam - 1; i > 0; i--) t[i] = op(t[l(i)],t[r(i)]);
 }
};
```

2.15 Merge Sort Tree

```
//~ Puede resolver Kth Min en O(Q * log^3 N)
//~ La funcion query(l,r,k) retorna la cantidad de numeros en el
//~ intervalo que son menores o iguales a k en [L,R] en O(log^2 N)

typedef long long tipo;

struct segtree {
  struct node {
    tipo ans = 0, l, r, inv = 0; // Poner el neutro del update
    vector <tipo> cur, left, right;
    tipo nomatch = 0; // No match en el intervalo de query
    node base(node aux) { aux.ans = aux.inv = 0, aux.cur.clear(); return
        aux;} //Poner el neutro de la query
    void set_node(tipo x, tipo pos) {ans = 0, cur.pb(x); l = r = pos;}
    void combine(node a, node b) {
      left = a.cur, right = b.cur;
      int p1 = 0, p2 = 0; inv = ans = 0;
      while(p1 < a.cur.size() && p2 < b.cur.size()) {
        if(a.cur[p1] <= b.cur[p2]) cur.pb(a.cur[p1++]);
        else cur.pb(b.cur[p2++]), inv += a.cur.size() - p1;
      }
      while(p1 < a.cur.size()) cur.pb(a.cur[p1++]);
      while(p2 < b.cur.size()) cur.pb(b.cur[p2++]);
      inv += a.inv + b.inv;
      l = min(a.l,b.l); r = max(a.r,b.r);
    }
  };
  vector <node> t;

  int BS(int k, vector <tipo> &aux) {
    int a = -1, b = aux.size();
    while(b-a > 1) {
      int med = (a+b)/2;
      if(aux[med] > k) b = med;
      else a = med;
    }
```

```
    return b; //return aux.size()-b for K-Query
  }

  ll ask(int p, tipo l, tipo r, tipo k) {
    if(l > t[p].r || r < t[p].l) return 0LL;
    if(l <= t[p].l && t[p].r <= r) {
      return BS(k,t[p].cur);
    }
    return ask(2*p+1,l,r,k)+ask(2*p+2,l,r,k);
  }

  void update(int p, tipo pos, tipo val) {
    if(t[p].r < pos || t[p].l > pos) return;
    if(t[p].l == t[p].r) { t[p].set_node(val,pos); return; }
    update(2*p+1, pos, val); update(2*p+2, pos, val);
    t[p].combine(t[2*p+1], t[2*p+2]);
  }

  void build(tipo a, tipo b, int p, vector <tipo> &v) {
    if(a==b) {t[p].set_node(v[a],a); return;}
    tipo med=(a+b)/2;
    build(a, med, 2*p+1, v); build(med+1, b, 2*p+2, v);
    t[p].combine(t[2*p+1], t[2*p+2]);
  }
  ll query(tipo l, tipo r, tipo k) {return ask(0,l,r,k);}
  void modificar(tipo pos, tipo val) {update(0,pos,val);}
  void construir(vector <tipo> &v, int n) { t.resize(4*n); build(0,n
      -1,0,v); }

  tipo find_kth(int l, int r, tipo k) {
    tipo minA = (tipo)(-1e9-7), maxA = (tipo)(1e9+7);
    while(maxA - minA > 1) {
      tipo med = (maxA + minA)/2;
      int ans = query(l,r,med);
      if(ans >= k) maxA = med;
      else minA = med;
    }
    return maxA;
  }
};
```

2.16 Persistent

```
struct Vertex {
    Vertex *l, *r;
    ll sum;

    Vertex(ll val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};

Vertex* build(ll a[], ll tl, ll tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    ll tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}

ll get_sum(Vertex* v, ll tl, ll tr, ll l, ll r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    ll tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}

Vertex* update(Vertex* v, ll tl, ll tr, ll pos, ll new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    ll tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, new_val), v->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos, new_val));
}

vector <Vertex*> estado;
```

2.17 Segtree

```
typedef long long tipo;
```

```cpp
const tipo NEUT = 0; // REMINDER !!!

struct node {
    int l, r; tipo ans;
    node() {ans = NEUT, l = r = -1;} // REMINDER !!! Set NEUT
    node(tipo val, int pos): l(pos), r(pos),ans(val) {} // Set node
    void update(tipo val) {ans = val;} // Define update function
};

struct segtree { // Segtree for Sum Range Query
    #define l(x) int(x<<1)
    #define r(x) int(x<<1|1)

    vector<node> t; int tam;

    node op(node a, node b) { //Operacion de query
        node aux; aux.ans = a.ans + b.ans;
        aux.l = a.l; aux.r = b.r;
        return aux;
    }

  node query(int l, int r, int p = 1) {
    node &cur = t[p];
    if(l > cur.r || r < cur.l) return node(); // Return NEUT
    if(l <= cur.l && cur.r <= r) return cur;
    return op(query(l,r,l(p)),query(l,r,r(p)));
    }

  void update(int pos, tipo val, int p = 1) { // root at p = 1
        node &cur = t[p];
    if(cur.r < pos || cur.l > pos) return;
    if(cur.l == cur.r) { cur.update(val); return; }
    update(pos, val, l(p)); update(pos, val, r(p));
    cur = op(t[l(p)], t[r(p)]);
  }

    void build(vector<tipo> v, int n) { // iterative build
        tam = sizeof(int) * 8 - __builtin_clz(n); tam = 1<<tam;
        t.resize(2*tam); v.resize(tam);
        forn(i,tam) t[tam+i] = node(v[i],int(i));
        for(int i = tam - 1; i > 0; i--) t[i] = op(t[l(i)],t[r(i)]);
    }
};
```

```cpp

//~ Max Sum Query
struct node {
  tipo ans, pref, suff, sum, l, r;
  node() {ans = pref = suff = sum = NEUT, l = r = -1;}
  node(tipo val, int pos) : ans(val), pref(val), suff(val), sum(val), l
      (pos), r(pos) {}
    void update(tipo val) {ans = pref = suff = sum = val;}
};

node op(node a, node b) {
  node aux;
  aux.pref = max(a.pref, a.sum + b.pref);
  aux.suff = max(b.suff, b.sum + a.suff);
  aux.sum = a.sum + b.sum;
  aux.ans = max(max(a.ans,b.ans),a.suff + b.pref);
  aux.l = a.l; aux.r = b.r;
  return aux;
}
```

2.18 Sqrt Decomposition

```
// This code solves the same problem as lazy arithmetic sum

struct block {
  ll l, r, a, b, ans;
  block(ll _l, ll _r) {
    l = _l; r = _r; ans = a = b = 0;
  }
  block() {}
};

const int BSIZE = 350; // Size of each block

struct segment_block {
  int n;
  vi v;
  vector <block> t;

  void refresh_block(block &u) { //clean block and refresh vector
    block new_b = block(u.l,u.r);
    forr(i,u.l,u.r+1) {
      v[i] += u.a + u.b * (i - u.l);
      new_b.ans += v[i];
    }
    u = new_b;
  }

  ll query_block(block &u, ll l, ll r) { // The answer of one blck
    if(l <= u.l && u.r <= r) { // full intersection
      ll last = u.a + u.b * (u.r - u.l);
      return u.ans + (u.a + last) * (u.r-u.l+1) / 2;
    }
    // partial intersection
    refresh_block(u);
    ll ans = 0;
    forr(i,max(l,u.l),min(u.r+1,r+1)) {
      ans += v[i];
    }
    return ans;
  }

  void update_block(block &u, ll l, ll r, ll a, ll b) {// upd one blck
    if(l <= u.l && u.r <= r) { // full intersection
      u.a += (u.l - l) * b + a;
      u.b += b;
      return;
    }
    // partial intersection
    refresh_block(u);
    forr(i,max(l,u.l),min(u.r+1,r+1)) {
      v[i] += (i - l) * b + a;
      u.ans += (i - l) * b + a;
    }
  }

  ll query(ll l, ll r) {
    ll ans = 0;
    for(block &u : t) {
      if(r < u.l || l > u.r) continue;
      ans += query_block(u,l,r);
    }
    return ans;
  }

  void update(ll l, ll r, ll a, ll b) {
    for(block &u : t) {
      if(r < u.l || l > u.r) continue;
      update_block(u,l,r,a,b);
    }
  }

  void build(vi _v, int _n) {
    v = _v; n = _n;
    int tam = 0;
    while(tam <= n-1) {
      t.pb(block(tam,min(n-1,tam+BSIZE-1)));
      tam += BSIZE;
      refresh_block(t.back());
    }
  }

  void debug() { // good for check if updates are correct
    for(block u : t) {
      refresh_block(u);
      forr(i,u.l,u.r+1) cerr << v[i] << "␣";
```

```
    }
    cerr << "\n"; RAYA;
  }
};
```

2.19 Sum Min K Multiset

```
typedef int tipo;

struct MultisetWithSum {
    multiset<tipo> ms;
    tipo sum = 0;
    void insert(tipo x) {ms.insert(x); sum += x;}
    void erase(tipo x) {ms.erase(ms.find(x)); sum -= x;}
    int size() const { return int(ms.size()); }
    tipo highest() const { return *ms.rbegin(); }
    tipo lowest() const { return *ms.begin(); }
};

struct SumMinKMultiset {
    MultisetWithSum low, high;
    void insert(tipo x) {low.insert(x);}
    void erase(tipo x) { (low.ms.find(x) != low.ms.end() ? low : high).
        erase(x);}
    tipo sumMinK(int k) { adjust(k); return low.sum; }
    tipo kth_element(int k) { adjust(k); return low.highest();}
    void adjust(int k) { // Low tenga k elementos
        assert(low.size() + high.size() >= k);
        while (low.size() < k) {
            tipo toMove = high.lowest();
            low.insert(toMove);
            high.erase(toMove);
        }
        while(high.size() > 0 && high.lowest() < low.highest()) {
            tipo toMove = high.lowest();
            high.erase(toMove); low.insert(toMove);
            toMove = low.highest();
            low.erase(toMove); high.insert(toMove);
        }
        while (low.size() > k) {
            tipo toMove = low.highest();
            low.erase(toMove);
            high.insert(toMove);
        }
    }
};
```

2.20 Treap

```
// Treap, es levemente mas poderoso que un set, pero mas lento
// Complejidad: O(log n) por operacion
typedef struct item *pitem;
struct item {
    int pr,key,cnt;
    pitem l,r;
    item(int key):key(key),pr(rand()),cnt(1),l(0),r(0) {}
};
int cnt(pitem t){return t?t->cnt:0;}
void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
void split(pitem t, int key, pitem& l, pitem& r){ // l: < key, r: >= key
    if(!t)l=r=0;
    else if(key<=t->key)split(t->l,key,l,t->l),r=t;
    else split(t->r,key,t->r,r),l=t;
    upd_cnt(t);
}
void insert(pitem& t, pitem it){
    if(!t)t=it;
    else if(it->pr>t->pr)split(t,it->key,it->l,it->r),t=it;
    else insert(it->key<t->key?t->l:t->r,it);
    upd_cnt(t);
}
void merge(pitem& t, pitem l, pitem r){
    if(!l||!r)t=l?l:r;
    else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
    else merge(r->l,l,r->l),t=r;
    upd_cnt(t);
}
void erase(pitem& t, int key){
    if(t->key==key)merge(t,t->l,t->r);
    else erase(key<t->key?t->l:t->r,key);
    upd_cnt(t);
}
void unite(pitem &t, pitem l, pitem r){
    if(!l||!r){t=l?l:r;return;}
    if(l->pr<r->pr)swap(l,r);
    pitem p1,p2;split(r,l->key,p1,p2);
```

```
    unite(l->l,l->l,p1);unite(l->r,l->r,p2);
    t=l;upd_cnt(t);
}
pitem kth(pitem t, int k){
    if(!t)return 0;
    if(k==cnt(t->l))return t;
    return k<cnt(t->l)?kth(t->l,k):kth(t->r,k-cnt(t->l)-1);
}
pair<int,int> lb(pitem t, int key){ // position and value of lower_bound
    if(!t)return {0,1<<30}; // (special value)
    if(key>t->key){
        auto w=lb(t->r,key);w.first+=cnt(t->l)+1;return w;
    }
    auto w=lb(t->l,key);
    if(w.first==cnt(t->l))w.second=t->key;
    return w;
}
```

3 Dp

3.1 Cht Dynamic

```
// Convex Hull trick dinamico.
// O(log n) para agregar e consultar.
// Para minimo, cambiar el signo de m y b.
typedef ll tc;
const tc is_query=-(1LL<<62); // special value for query
struct Line {
    tc m,b;
    mutable multiset<Line>::iterator it,end;
    const Line* succ(multiset<Line>::iterator it) const {
        return (++it==end? NULL : &*it);}
    bool operator<(const Line& rhs) const {
        if(rhs.b!=is_query)return m<rhs.m;
        const Line *s=succ(it);
        if(!s)return 0;
        return b-s->b<(s->m-m)*rhs.m;
    }
};
struct HullDynamic : public multiset<Line> { // for maximum
    bool bad(iterator y){
        iterator z=next(y);
        if(y==begin()){
            if(z==end())return false;
            return y->m==z->m&&y->b<=z->b;
        }
        iterator x=prev(y);
        if(z==end())return y->m==x->m&&y->b<=x->b;
        return (x->b-y->b)*(z->m-y->m)>=(y->b-z->b)*(y->m-x->m);
    }
    iterator next(iterator y){return ++y;}
    iterator prev(iterator y){return --y;}
    void add(tc m, tc b){
        // m *= -1; b *= -1; --> For minumum
        iterator y=insert((Line){m,b});
        y->it=y;y->end=end();
        if(bad(y)){erase(y);return;}
        while(next(y)!=end()&&bad(next(y)))erase(next(y));
        while(y!=begin()&&bad(prev(y)))erase(prev(y));
    }
    tc eval(tc x){
```

```
        Line l=*lower_bound((Line){x,is_query});
        return l.m*x+l.b; // -1*(l.m*x + l.b) for minimum
    }
};
```

3.2 Cht Li Chao Tree

```
//~ Li Chao Tree
//~ Permite resolver CHT dinamico
//~ Desventaja: si evaluamos x > 10^6, no aguanta el segment tree

const int N = int(1e6 + 1);
const ll INF = ll(1e18+10);

struct punto { ll x, y; };

vector <punto> tree(4*N, {0, INF});

ll f(punto line, ll x) { return line.x * x + line.y; }


void insert(punto line, ll lo = 1, ll hi = N, int i = 1){
    ll m = (lo + hi) / 2;
    bool left = f(line, lo) < f(tree[i], lo);
    bool mid = f(line, m) < f(tree[i], m);

    if(mid) swap(tree[i], line);

    if(hi - lo == 1) return;
    else if(left != mid) insert(line, lo, m, 2*i);
    else insert(line, m, hi, 2*i+1);
}

ll query(ll x, ll lo = 1, ll hi = N, int i = 1){
    int m = (lo+hi)/2;
    ll curr = f(tree[i], x);
    if(hi-lo==1) return curr;
    if(x<m) return min(curr, query(x, lo, m, 2*i));
    else return min(curr, query(x, m, hi, 2*i+1));
}
```

3.3 Chull Trick

```
// Convex Hull Trick
// O(log n) para agregar linea y consultar
// Las linease agregan en orden creciente de pendiente
typedef long long tipo;
typedef __int128 ull;

struct punto {
    tipo x, y;
    punto operator -(const punto &p) const {return {x-p.x,y-p.y};}
    punto operator +(const punto &p) const {return {x+p.x,y+p.y};}
    tipo operator *(const punto &p) const {return x*p.x + y*p.y;}
    ull operator ^(const punto &p) const { // Producto Cruz
        return (ull)x * p.y - (ull)y * p.x; } // (ull) --> __int128
};

struct chull { // Agregar siempre pendientes en orden creciente
    ll op = 1; // 1 para minimo, -1 para maximo
    vector <punto> hull;

    chull(bool maxi) { // true para maximo, false para minimo
        if(maxi) op *= -1;
        // add_line(0,0); // Push base case {0,0} if necessary
    }
    ll get(tipo x) {
        punto query = {op * x,op * 1LL};
        int a=0, b=SZ(hull);
        while(b-a > 1) {
            int med = (a+b)/2;
            if(query * hull[med-1] >= query * hull[med]) a = med;
            else b = med;
        }
        return op * (query * hull[a]);
    }

    bool check(punto aux, int last) {
        return op * ((hull[last]-hull[last-1])^(aux-hull[last])) <= 0;
    }

    void add_line(ll x, ll y) {
        if(SZ(hull)) assert(x >= hull.back().x);  // Chequeo de
            pendientes crecientes

        punto aux = {x,y}; int last = SZ(hull)-1;
        while(last > 0 && check(aux,last)) {
```

```
            hull.pop_back(); last--;
        }
        hull.pb(aux);
    }
};
```

3.4 Count Cycles

```
// Count the number of cycles in a graph
// Time: O(n*2^n)
// Usage: count_cycles(n); // n --> number of nodes
const int MAXN = 20;
int g[MAXN][MAXN];

bool bit(int mask, int i) {
    if((mask & 1<<i) != 0) return true;
    return false;
}

int first(int mask, int n) {
    for(int i = 0; i < n; i++) {
        if(bit(mask,i)) {return i;}
    }
    return -1;
}

int count(int mask, int n) {
    int ans = 0;
    for(int i = 0; i < n; i++) {
        if(bit(mask,i)) ans++;
    }
    return ans;
}

ll count_cycles(int n) {
    ll dp[1<<MAXN][MAXN];
    memset(dp,0,sizeof(dp));
    ll ans = 0;
    for(int mask = 1; mask < (1<<n); mask++) {
        if(count(mask,n) == 1) {
            dp[mask][first(mask,n)] = 1;
            continue;
        }
        int f = first(mask,n);
        for(int last = 0; last < n; last++) {
            if(!bit(mask,last) || last == f) continue;
            for(int next = 0; next < n; next++) {
                if(g[last][next]) {
                    dp[mask][last] += dp[mask ^ (1<<last)][next];
                }
            }
            if(count(mask,n) >= 3 && g[first(mask,n)][last]) ans += dp[
                mask][last];
        }
    }
    return ans / 2;
}
```

3.5 Divide And Conquer Dp Opt

```
// Divide and conquer optimization for DP
// Complexity: O(nklogn)
// Usage: solve_dac(n,k); // n --> number of elements, k --> number of
    groups

ll costo[MAXN][MAXN];

vector <ll> last(MAXN), dp(MAXN);

void calc_costo(int n, vector <ll> &v) {
    // Dependiendo el problema, aca dentro realizamos la funcion para
    // precalcular el costo de cada intervalo y guardarlo en 'costo'
}

void compute(int l, int r, int optl, int optr) {
    if(l > r) return;
    int med = (l+r)/2;
    pair<ll,ll> best = {INF,-1};
    for(int p = optl; p <= min(med,optr); p++) {
        best = min(best,{last[p] + costo[p+1][med],p});
    }
    dp[med] = best.first;
    int opt = best.second;
    compute(l,med-1,optl,opt);
    compute(med+1,r,opt,optr);
}
```

```cpp

ll solve_dac(int n, int k) { // divide and conquer optimization
    for(int i = 0; i < n; i++) last[i] = costo[0][i];
    for(int i = 2; i <= k; i++) {
        fill(all(dp),INF);
        compute(0,n-1,0,n-1);
        last = dp;
    }
    return dp[n-1];
}
```

3.6 Elevator Problem

```cpp
/*
 * Elevator Problem
 *
 * Dado los pesos maximos de las personas y el peso maximo del
 * ascensor, cual es el minimo numeros de viajes necesarios para subir
 * a todas las personas: O(N * 2^N)
 *
 */

ll elevator_problem(int n, ll x, vector <ll> &peso) {
    vector <pair<ll,ll>> best(1<<n); best[0] = {1,0};
    forr(s,1,(1<<n)) {
        best[s] = {n+1,0};
        forn(p,n) {
            if(s&(1<<p)) {
                pair<ll,ll> option = best[s^(1<<p)];
                if(option.second + peso[p] <= x) {
                    option.second += peso[p];
                }
                else{
                    option.first++; option.second = peso[p];
                }
                best[s] = min(best[s],option);
            }
        }
    }
    return best[(1<<n)-1].first;
}
```

3.7 Knapsack

```cpp
//~ Problema de la mochila - O(n*w)

struct item { int w, v; };

ll knapsack(vector <item> &v, int n, int w) {
    vector <ll> dp(w+1,0), aux(w+1,0);
    forn(i,n) {
        forn(j,w+1) {
            aux[j]=dp[j];
            if(j-v[i].w < 0) continue;
            aux[j]=max(aux[j],dp[j-v[i].w]+v[i].v);
        }
        forn(j,w+1) dp[j]=aux[j];
    }
    ll ans = 0;
    forn(i,w+1) ans=max(ans,dp[i]);
    return ans;
}
```

3.8 Knapsack Big W

```cpp
// Description: Given a set of items, each with a weight and a value,
    determine the number of each item to include in a collection so that
     the total weight is less than or equal to a given limit and the
    total value is as large as possible.
// Time: O(n * MAXV)
// Usage: knapsack(v,n,w); // v --> vector of items, n --> number of
    items, w --> maximum weight

struct item { int w, v; };

ll knapsack(vector <item> &v, int n, int w) {
    int MAXV = 100005;
    vector <ll> dp(MAXV+1,INF), aux(MAXV+1,INF);
    vector <bool> visto(MAXV,false);
    visto[0]=true; dp[0]=0;
    forn(i,n) {
        forn(j,MAXV) {
            aux[j]=dp[j];
            if(j-v[i].v < 0) continue;
            if(visto[j-v[i].v] == true) {
                aux[j]=min(aux[j],dp[j-v[i].v]+v[i].w);
                visto[j]=true;
```

```
            }
        }
        forn(j,MAXV) dp[j]=aux[j];
    }
    ll ans = 0;
    forn(i,MAXV+1) if(dp[i] <= w && visto[i]==true) ans=i;
    return ans;
}
```

3.9 Knuth Division Dp Opt

```
const ll INF = (ll)(1e18+10);
const int MAXN = 5005;
vector <ll> prefix;

// Sea Pos(i,j) la posicion del corte que optimiza la dp, intervalo [i,j]
// Se puede aplicar optimizacion de Knuth si se cumple que:
// Pos(i,j-1) <= pos(i,j) <= pos(i+1,j)
// Objetivo: llevar un array a elementos individuales mediante cortes.

ll suma(int i, int j) {
    return prefix[j+1] - prefix[i]; // Suma de los elementos del inter
}

ll solve_Knuth(int n) {
    vector <ll> pos(n+1);
    vector <vector <ll> > dp(MAXN,vector<ll>(MAXN,INF));
    for(int i = 0; i < n; i++) dp[i][i] = 0, pos[i] = i;

    for(int len = 1; len < n; len++) {
        vector <ll> aux(n+1,0);
        for(int i = 0; i < n - len; i++) {
            for(int k = pos[i]; k <= pos[i+1]; k++) {
                ll cost = suma(i,i+len) + dp[i][k] + dp[k+1][i+len];
                if(cost < dp[i][i+len]) {
                    dp[i][i+len] = cost; aux[i] = k;
                }
            }
        }
        pos = aux;
    }
    return dp[0][n-1];
}
```

3.10 Lcs

```
// Longest Common Subsequence
// O(n*m) time, O(n*m) space
// Usage: LCS(s1,s2)

string LCS(string s1, string s2) { // O(n*m)
    const int n = SZ(s1), SZ(s2);
    int dp[n+1][m+1]; pair<int,int> last[n+1][m+1];
    forn(i,n+1) {
        forn(j,m+1) {
            if(i==0 || j==0) dp[i][j] = 0;
            else if(s1[i-1] == s2[j-1])
                dp[i][j] = dp[i-1][j-1]+1, last[i][j] = {i-1,j-1};
            else if(dp[i-1][j] > dp[i][j-1])
                dp[i][j] = dp[i-1][j], last[i][j] = {i-1,j};
            else dp[i][j] = dp[i][j-1], last[i][j] = {i,j-1};
        }
    }
    string ans; pair<int,int> cur = {n,m};
    while(cur.first != 0 && cur.second != 0) {
        int x = cur.first, y = cur.second;
        if(x-last[x][y].first==1 && y-last[x][y].second==1) ans += s1[x-1];
        cur = last[x][y];
    }
    reverse(all(ans));
    return ans;
}
```

3.11 Lis

```
// Longest Increasing Subsequence
// O(n*log(n)) time, O(n) space
// Usage: lis(a,strict)

// strict = 1: estrictamente creciente
ll lis(vi &a, int strict = 0){
  vi temp; temp.pb(a[0]);
  forr(i, 1, SZ(a)){
    ll x = a[i];
    if(x >= temp.back()+strict) temp.pb(x);
```

```
    else {
      auto it = upper_bound(all(temp), x-strict);
      *it = x;
    }
  }
  return SZ(temp);
}
```

3.12 Merge Sort

```
// Merge sort con inversiones
// Complejidad: O(n log n)
// Usage: merge_sort(0,n-1,v)

ll inv = 0; // Inversiones

vector <int> merge_sort(int li, int ri, vector <int> &v) {
    if(li == ri) return {v[li]};
    vector <int> ans;
    int med = (li+ri)/2;
    vector <int> l = merge_sort(li,med,v);
    vector <int> r = merge_sort(med+1,ri,v);
    int a = 0, b = 0;
    while(a < l.size() && b < r.size()) {
        if(l[a] <= r[b]) ans.pb(l[a++]);
        else { inv += (ll)(l.size() - a); ans.pb(r[b++]);}
    }
    while(a < l.size()) ans.pb(l[a]), a++;
    while(b < r.size()) ans.pb(r[b]), b++;
    return ans;
}
```

4 Geometry

4.1 Angular Sort

```
// Angular sort
struct frac {
    ll num, den;
    frac() {}
    frac(ll x, ll y) {
        ll m = __gcd(abs(x),abs(y));
        num = x / m; den = y / m;
    }
    int cuad(ll n, ll d) {
        if(n >= 0 && d >= 0) return 1;
        if(n >= 0 && d < 0) return 2;
        if(n < 0 && d < 0) return 3;
        if(n < 0 && d >= 0) return 4;
    }
    bool operator <(frac &p) {
        if(cuad(num,den) != cuad(p.num,p.den)) {
            return cuad(num,den) < cuad(p.num,p.den); }
        return num * p.den < p.num * den;
    }
    bool operator ==(frac &p) {
        return (num * p.den == den * p.num) && (cuad(num,den) == cuad(p.
            num,p.den));
    }
};

struct punto {
    ll x, y;
    frac pend;
    ll val;
    punto() {}
    punto(ld a, ld b, ll z) {
        x = a, y = b, val = z, pend = frac(b,a);}
    bool operator <(punto p) {
        return pend < p.pend;
    }
};
```

4.2 Calipers

```
// Devuelve la distancia entre los puntos mas lejanos en O(N)
// Cuando se usa con nuestra convex_hull, hacer reverse porque vienen CW
ld callipers(vector<punto> p, int n){
    ld r=0;     // prereq: Convex, ccw, NO COLLINEAR POINTS
    for(int i=0,j=n<2?0:1;i<j;++i){
        for(;;j=(j+1)%n){
            r=max(r,(p[i]-p[j]).mod());
            if((p[(i+1)%n]-p[i])^(p[(j+1)%n]-p[j])<=EPS)break;
        }
    }
    return r;
}
```

4.3 Closest Points

```
typedef long double tipo;

struct punto {
    tipo x, y; int ind;
    punto operator -(punto p) const {return {x-p.x,y-p.y};}
    bool operator <(punto p) const {return x != p.x ? x < p.x : y < p.y
        ;}
    tipo mod() {return sqrtl(x*x + y*y);}
    tipo mod2() {return x*x + y*y;}
};

tuple<tipo,int,int> closest(vector<punto> &p) { //closest and indices
    int n = p.size();
    set<punto> s;
    tipo best = (tipo)(1e18); int ansi, ansj;
    int j = 0;
    forn(i,n) {
        tipo d = ceil(sqrt(best));
        while(p[i].x - p[j].x >= best) s.erase({p[j].y, p[j].x, j}), j
            ++;
        auto it1 = s.lower_bound({p[i].y - d, p[i].x});
        auto it2 = s.upper_bound({p[i].y + d, p[i].x});
        for(auto it = it1; it != it2; ++it) {
            tipo dx = p[i].x - it->y;
            tipo dy = p[i].y - it->x;
            if(dx * dx + dy * dy < best) {
                best = dx * dx + dy * dy;
                ansi = i, ansj = it->ind;
            }
        }
        s.insert({p[i].y, p[i].x, (int)i});
    }
    return {sqrtl(best),ansi,ansj};
}
```

4.4 Closest Points Monogon

```
const ll INF = (ll)(1e15+10); // (1e18+10)

template<typename T>
using minpq = priority_queue<T, vector<T>, greater<T>>;

struct punto {
    ll x, y;
    punto (ll x = 0, ll y = 0) : x(x), y(y) {}
    punto operator -(const punto &p) const {
        return punto(x - p.x, y - p.y); }
    ll len2() const { return x * x + y * y; }
    bool const operator <(const punto &p) const {
        return (x != p.x) ? x < p.x : y < p.y;
    }
};

istream& operator>>(istream &is, punto &p) {
    return is >> p.x >> p.y;
}

ll closest(vector<punto> &p, int l, int r) { //Init l = 0 and r = p.size
    ()-1
    if(l == r) return INF;
    int m = (l + r) / 2;
    ll mid = p[m].x;
    ll d = min(closest(p, l, m), closest(p, m + 1, r));
    vector<punto> A, B, C;
    forr(i, l, m + 1) {
        ll r = mid - p[i].x;
        if(r * r <= d) A.push_back(p[i]);
    }
    forr(i, m + 1, r + 1) {
        ll r = mid - p[i].x;
        if(r * r <= d) B.push_back(p[i]);
```

```
    }
    auto cmpy = [&](punto a, punto b) { return a.y < b.y; };
    merge(all(A), all(B), back_inserter(C), cmpy);
    forr(i, 0, C.size()) {
        int j = i + 1;
        while(j < C.size() && (C[i] - C[j]).y * (C[i] - C[j]).y <= d) {
            d = min(d, (C[i] - C[j]).len2());
            j++;
        }
    }
    inplace_merge(p.begin() + l, p.begin() + m + 1, p.begin() + r + 1,
        cmpy);
    return d;
}

void sorting(vector <punto> &p) { sort(all(p)); } // Don't Forget
```

4.5 Complex

```
/*
 * Numeros complejos
 *
 * Tener en cuentas que:
 *
 * - No se pueden leer complejos directamente con cin >>;
 * - Una vez definidas las macros, no podemos definir variables con el
 *   mismo nombre
 *
 */

typedef complex<double> complejo;
#define x real()
#define y imag()

complejo a, b, c;
abs(a); // Devuelve sqrt(a.x^2 + a.y^2)
arg(a); // Devuelve el angulo entre ( -pi ; pi ]
conj(a); // Devuelve a.x - i*a.y
a*b; // Devuelve (a.x*b.x-a.y*b.y) + i*(a.x*b.y + a.y*b.x)
polar(1.0,pi); // Devuelve el complejo de modulo 1 y angulo pi
    (conj(a)*b).y // Producto cruz -> a.x*b.y - a.y*b.x
(conj(b-a)*(c-a)).y/abs(b-c) // Devuelve distancia punto a -> recta bc
```

4.6 Convex Hull

```
/*
Convex Hull:
Complejidad O(N).
La hull no tiene puntos colineales
Los devuelve en orden clockwise -> Para rotating calipers hacer reverse
*/

const ld EPS = 1e-10;
void convex_hull(vector<punto> &a) {
    if(SZ(a) == 1) return;
    sort(all(a));
    punto p1 = a[0];
    vector<punto> up, down;
    up.pb(p1); down.pb(p1);
    forr(i,1,SZ(a)) {
        int n = SZ(up), m = SZ(down);
        while(n > 1 && ((up[n-1]-up[n-2])^(a[i]-up[n-1])) >= -EPS) {
            up.pop_back(); n--;
        } up.pb(a[i]);
        while(m > 1 && ((down[m-1]-down[m-2])^(a[i]-down[m-1])) <= EPS)
            {
            down.pop_back(); m--;
        } down.pb(a[i]);
    } // Cambiar EPS a 0 para mejor precision en enteros.
    a.clear();
    for(punto u : up) a.pb(u);
    for(int i = SZ(down)-2; i > 0; i--) a.pb(down[i]);
}
```

4.7 Formulas

```
// Shoelace formula
tipo area(vector <punto> &v) {
    tipo ans = 0.0; int n = v.size();
    forn(i,n) ans += v[i] ^ v[(i+1)%n];
    return fabs(ans/2.0);
}

tipo dist_point_line(punto &p, recta &r) {
    punto p1 = r.p, p2 = r.p + r.v;
    return fabs((p1-p)^(p2-p))/r.v.mod();
```

```
}

punto project(punto a, punto b) { //Proyeccion de b sobre a
    return ((a*b)/a.mod2()) * a;
}

tipo find_alpha(recta r, punto p) {
    return r.v.x != 0 ? (p.x-r.p.x)/r.v.x : (p.y-r.p.y)/r.v.y;
}
```

4.8 Intersection All

```
struct recta { // Puede usarse para segmentos ([p,p+v] o alpha = [0,1])
    punto v, p; // v -> director, p -> punto por donde pasa
    recta(punto p1, punto p2) { v = (p2-p1); p = p1;}
    recta() {}
    recta(tipo A, tipo B, tipo C) { // Transform Ax + By + C = 0
        v = {-B,A}; A != 0 ? p = {-C / A,0} : p = {0,-C / B};
    }
    bool is_in(punto q){return fabs((q.x-p.x)*v.y - (q.y-p.y)*v.x) < EPS
        ;}
    punto eval(double x) {return x * v + p;}
};

bool inter_recta(recta &r1, recta &r2, punto &ans) {
    // Retorna false si son paralelas, sino guarda el punto en ans
    if(abs(r1.v ^ r2.v) < EPS) return false;
    tipo alpha = tipo((r2.p - r1.p)^r2.v) / tipo(r1.v^r2.v);
    ans = r1.p + alpha * r1.v;
    return true;
}

bool inter_seg(recta &r1, recta &r2, punto &ans) {
    if(r1.p == r2.p || r1.p == r2.p+r2.v) {ans = r1.p; return true;}
    if(r1.p+r1.v == r2.p || r1.p+r1.v == r2.p+r2.v) {
        ans = r1.p+r1.v; return true; } //Casos que coincidan extremos
    if(abs(r1.v ^ r2.v) < EPS) return false; // son paralelos
    tipo alpha = tipo((r2.p - r1.p)^r2.v) / tipo(r1.v^r2.v);
    tipo beta = tipo((r1.p - r2.p)^r1.v) / tipo(r2.v^r1.v);
    if(alpha < -EPS || beta < -EPS) return false;
    if(alpha > 1.0+EPS || beta > 1.0+EPS) return false;
    ans = r1.p + alpha * r1.v; return true;
}


struct circ { punto c; tipo r; };

tipo dist_point_line(punto &p, recta &r) {
    punto p1 = r.p, p2 = r.p + r.v;
    return fabs((p1-p)^(p2-p))/r.v.mod();
}

punto project(punto a, punto b) { //Proyeccion de b sobre a
    return ((a*b)/a.mod2()) * a;
}

vector <punto> inter_circ_line(recta r, circ c) {
    vector <punto> ans; tipo dist = dist_point_line(c.c,r);
    if(dist > c.r+EPS) return ans;
    (c.c-r.p) * r.v != 0 ? r.p = r.p : r.p = r.p + r.v;
    punto aux = c.c - r.p, dir = project(r.v,aux);
    if(fabs(dist-c.r) <= EPS) {ans.pb(r.p + dir); return ans;}
    tipo factor = sqrt(c.r*c.r - dist*dist)/dir.mod();
    ans.pb(r.p + dir + factor * dir); ans.pb(r.p + dir - factor * dir);
    return ans;
}

tipo intersection_area(circ a, circ b) {
    punto aux = (b.c - a.c); tipo dist=aux.mod(), dist2=aux.mod2();
    if(a.r + b.r - dist < -EPS) return 0;
    if(fabs(a.r - b.r) - dist > -EPS) {
        return min(a.r, b.r) * min(a.r, b.r) * 2*acosl(0); }
    tipo alpha = acosl((dist2 + a.r*a.r - b.r*b.r) / (2 * dist * a.r));
    tipo beta = acosl((dist2 + b.r*b.r - a.r*a.r) / (2 * dist * b.r));
    tipo ans1 = (alpha - sinl(alpha+alpha)*0.5) * a.r * a.r;
    tipo ans2 = (beta - sinl(beta+beta)*0.5) * b.r * b.r;
    return ans1 + ans2;
}

vector <punto> inter_circ_circ(circ a, circ b) {
    vector <punto> ans;
    if(a.c==b.c) {
        return abs(a.r-b.r) <= EPS ? vector<punto>{a.c,a.c,a.c} : ans; }
    b.c = b.c - a.c; punto aux = a.c; a.c = a.c - a.c;
    recta r(-2*b.c.x , -2*b.c.y , a.r*a.r - b.r*b.r + b.c.x*b.c.x + b.c.
        y*b.c.y);
    ans = inter_circ_line(r,a);
```

```
    forn(i,ans.size()) ans[i] = ans[i] + aux; return ans;
}
```

4.9 Kd Tree

```
struct pt {  // for 3D add z coordinate
  ll x,y;
  pt(ll x, ll y):x(x),y(y){}
  pt(){}
  ll norm2(){return *this**this;}
  pt operator-(pt p){return pt(x-p.x,y-p.y);}
  ll operator*(pt p){return x*p.x+y*p.y;}
  // 2D from now on
  bool operator<(pt p)const{ // for convex hull
    return x<p.x-EPS||(abs(x-p.x)<=EPS&&y<p.y-EPS);}
};

ll manhattan(pt a, pt b){ return abs(a.x-b.x)+abs(a.y-b.y); }

// 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;
  ll 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 manhattan(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(SZ(vp)>1){
      sort(all(vp),x1-x0>=y1-y0?onx:ony);
      int m=SZ(vp)/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==node->pp) {INF,pt()};
      return {manhattan(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.first) best=min(best,search(p,s));
    return best;
  }
  // Return nearest point and its distance to p
  pair<ll,pt> nearest(pt p){return search(p,root);}
};
```

4.10 Point In Poly

```
bool point_in_poly(vector <punto> &v, punto p) { // O(n) for convex
    unsigned i, j, mi, mj, c = 0;
    for(i = 0, j = v.size()-1; i < v.size(); j = i++) {
        if((v[i].y <= p.y && p.y < v[j].y) || (v[j].y <= p.y && p.y < v[
            i].y)) {
            mi = i; mj = j; if(v[mi].y > v[mj].y) swap(mi,mj);
            if(((p-v[mi]) ^ (v[mj]-v[mi])) < 0 ) c ^= 1;
        }
    }
    return c;
}

bool point_in_poly_convex(vector <punto> &v, punto p) { // O(log n)
    bool ans = true;
    if( ((v[1]-v[0]) ^ (v[2]-v[1])) >= 0) reverse(all(v));
    int a = 1, b = v.size()-1;
    while(b-a > 1) {
        int med = (a+b)/2;
        if( ((v[med]-v[0]) ^ (p-v[med])) <= 0 ) a = med;
        else b = med;
```

```
    }
    if( ((v[a]-v[0]) ^ (p - v[a])) > 0) ans = false;
    if( ((v[b]-v[a]) ^ (p - v[b])) > 0) ans = false;
    if( ((v[0]-v[b]) ^ (p - v[0])) > 0) ans = false;
    return ans;
}
```

4.11 Template Punto

```
typedef long double tipo; //Cambiar a long long para operar en enteros
double EPS = (double)(1e-10);

struct punto { // Puede usarse para vectores
    tipo x, y;
    punto const operator -(const punto &p) const {return {x-p.x,y-p.y};}
    punto const operator +(const punto &p) const {return {x+p.x,y+p.y};}
    tipo operator *(const punto &p) const {return x*p.x + y*p.y;}
    tipo operator ^(const punto &p) const {return x*p.y - y*p.x;}
    bool operator == (const punto &p) const {
        return (abs(x-p.x) < EPS && abs(y-p.y) < EPS); // Para double
    }
    bool operator <(punto p) const {return x != p.x ? x < p.x : y < p.y
        ;}
    tipo arg() {return atan2(y,x);}
    tipo mod() {return sqrtl(x*x + y*y);}
    tipo mod2() {return x*x + y*y;}
};

punto operator*(tipo k, const punto &p) {return {k*p.x, k*p.y};}

ostream &operator << (ostream &os, const punto &p) { //Para imprimir
    return os << "(" << p.x << "," << p.y << ")";
}

istream &operator >> (istream &is, punto &p) { //Para leer
    return is >> p.x >> p.y;
}

struct frac { // Por si es necesario trabajar con enteros
    tipo n, d;
    ll mcd(ll a, ll b) {
        a = abs(a); b = abs(b);
        while(a > 0 && b > 0) {
            if(a >= b) a %= b;
            else b %= a;
        }
        return a == 0 ? b : a;
    }
    frac(ll x, ll y) {
        ll g = mcd(x,y);
        n = x/g; d = y/g;
        if(d < 0) d *= -1, n *= -1;
    }
    frac() {}
    bool operator ==(frac &F) const {return n*F.d == d*F.n;}
    bool operator < (frac &F) const {return n*F.d  < d*F.n;}
};
```

5 Grafos

5.1 2Sat

```cpp
// 2sat (2-satisfiability) - O(n + m) (Korasaju)

struct two_sat { // 2*x representa a x, y 2*x+1 a ~x
    int tot;
    vector<vector<int>> g, g_trans;
    vb used, assignment;
    vector<int> order, comp;

    two_sat(int _tot): tot(_tot){
        g.resize(tot); g_trans.resize(tot);
    } // tot = total de nodos (normales + negados), en general n = tot/2

    void dfs1(int v) {
        used[v] = true;
        for(auto u: g[v]) if(!used[u]) dfs1(u);
        order.pb(v);
    }

    void dfs2(int v, int cl) { // Korasaju para encontrar las SCC
        comp[v] = cl;
        for(auto u: g_trans[v]) if(comp[u] == -1) dfs2(u, cl);
    }

    bool solve() {
        order.clear(); used.assign(tot, false);
        comp.assign(tot, -1);
        forn(i, tot) if(!used[i]) dfs1(i);

        int comp_act = 0;
        forn(i, tot){
            auto v = order[tot-i-1];
            if(comp[v] == -1) dfs2(v, comp_act++);
        }

        assignment.assign(tot/2, false);
        forn(i, tot/2){
            if(comp[2*i] == comp[2*i + 1]) return false;
            assignment[i] = comp[2*i] > comp[2*i+1]; // asignacion
                greedy de variables
        }
        return true;
    }

    void add_edge(int from, int to){ // implicancia comun from->to
        g[from].pb(to);
        g_trans[to].pb(from); // agregar en el TRANSPUESTO
    }

    void add_or(int v1, int v2) { // agrega (v1 or v2)
        add_edge(v1 ^ 1, v2); // ~v1 -> v2
        add_edge(v2 ^ 1, v1); // ~v2 -> v1
    }
    // setear variable x en true/false: add_or(x, x)/add_or(~x, ~x)
};
```

5.2 Bellman Ford

```cpp
// Description: Bellman-Ford algorithm for finding the shortest path
    from a source to all other nodes in a graph. It can also detect
    negative cycles.
// Time: O(VE)

typedef long long tipo;
const int MAXN = 3000;
tipo INF = (tipo)(1e18+7);

struct arista {
    int x, y; tipo w; // Edge from x to y, w = weight
};

struct nodo {
    int p; tipo d; //f -> parent, d -> distance
};

vector<nodo> ans(MAXN);
vector<int> ciclo;

bool bFord(vector<arista> &lista, int n, int start) {
    int m = lista.size();
    forn(i,n) ans[i].p = i, ans[i].d = INF;
    ans[start].d = 0; int x;
    forn(i,n) {
```

```
        x = -1;
        for(arista u : lista) {
            if(ans[u.y].d > ans[u.x].d + u.w) {
                ans[u.y].d = ans[u.x].d + u.w;
                ans[u.y].p = u.x;
                x = u.y;
            }
        }
    }
    if(x == -1) return false;
    else {
        forn(i,n) x = ans[x].p;
        for(int v = x ;; v = ans[v].p) {
            ciclo.push_back(v);
            if(v == x && ciclo.size() > 1) break;
        }
        reverse(all(ciclo));
        return true;
    }
}
```

5.3 Bfs

```
const int MAXN = 200005;
typedef long long tipo;
tipo INF = (tipo)(1e18+7);

struct nodo {
    tipo d; bool visto; //d -> distance, visto -> seen
};

vector<nodo> BFS(int start, int n, vector<vector<int>> &g) {
    vector<nodo> ans(n); queue<int> q;
    forn(i, n) ans[i] = {INF,false};
    ans[start] = {0,true}; q.push(start);
    while(!q.empty()) {
        int v = q.front(); q.pop();
        for(int u : g[v]) {
            if(ans[u].visto) continue;
            ans[u] = {ans[v].d+1, true}; q.push(u);
        }
    }
    return ans;
}
```

5.4 Biconnected

```
struct edge{ ll u, v, id; };

struct graph{
    int n;
    vector<vii> adj; // (to, id)
    vector<edge> edges;
    graph(int _n) : n(_n), adj(_n) {}

    void add_edge(ll u, ll v){
    ll id = SZ(edges);
        adj[u].pb({v, id}); adj[v].pb({u, id});
        edges.pb({u, v, id});
    }

    int add_node(){ adj.pb({}); return n++; }
    vii& operator[](ll u) { return adj[u]; }
};

struct BCC{
    int n; graph adj;
    vector<vi> comps;
    vi num, low, art, stk, bridge;
    BCC(graph &_adj): n(_adj.n), adj(_adj){
        num.resize(n), low.resize(n), art.resize(n), bridge.resize(SZ(
            adj.edges));
        for (ll u = 0, t; u < n; ++u) if (!num[u]) dfs(u, -1, t = 0);
    }

    void dfs(ll v, ll p, ll &t){
        num[v] = low[v] = ++t;
        stk.pb(v);

        for(auto [u, id]: adj[v]) if (u != p){
            if (!num[u]){
                dfs(u, v, t);
                low[v] = min(low[v], low[u]);

        if(low[u] > num[v]) bridge[id] = true;
                if(low[u] >= num[v]){
```

```
                    art[v] = (num[v] > 1 || num[u] > 2);

                    comps.pb({v});
                    while (comps.back().back() != u)
                        comps.back().pb(stk.back()), stk.pop_back();
                }
            }
            else low[v] = min(low[v], num[u]);
        }
    }

    // build the block cut tree
    pair<vi, graph> build_tree(){
        graph tree(0); vi id(n);

        forn(v, n) if (art[v]) id[v] = tree.add_node();

        for (auto &comp : comps){
            ll node = tree.add_node();
            for(ll v: comp){
                if (!art[v]) id[v] = node;
                else tree.add_edge(node, id[v]);
            }
        }
        return {id, tree};
    }
};
```

5.5 Componentes Fuertemente Conexas

```
/*
   Algoritmo para hallar las componentes fuertmente conexas.
   Una componente es fuertemente conexa cuando para todo nodo
   perteneciente a la componente, se puede llegar a cualquier
   otro nodo tambien perteneciente a la componente.
*/
struct SCC {
    int n, scc;
    vector<vi> g, gr, ans;
    vb visto;
    vi order, comp_act, component;

    SCC(vector<vi> &_g): n(SZ(_g)) {
        g = _g;
        gr.resize(n), visto.resize(n), component.resize(n);
        forn(v, n) for(auto u: g[v]) gr[u].pb(v); // me lo creo aca el
            grafo traspuesto

        find_scc();
    }

    void DFS1 (int v) {
        visto[v] = true;
        for (int u : g[v]) if(!visto[u]) DFS1(u);
        order.pb(v);
    }

    void DFS2 (int v) {
        visto[v] = true;
        comp_act.pb(v);
        for (int u : gr[v]) if(!visto[u]) DFS2(u);
    }

    void find_scc() {
        fill(all(visto),false);
        forn(i,n) if(!visto[i]) DFS1(i);
        fill(all(visto),false);
        forn(i,n) {
            int v=order[n-i-1];
            if(!visto[v]) {
                DFS2(v);
                ans.pb(comp_act);
                comp_act.clear();
            }
        }
        scc = SZ(ans); // cantidad de scc's
        forn(i, scc) for(auto v: ans[i]) component[v] = i;
    }
};
```

5.6 Dfs

```
const int MAXN = 200005;
vector< vector<int> > g; // graph represented as an adjacency list
vector <bool> visto(MAXN,false);

```

```cpp
void dfs(int v) {
    visto[v] = true;
    for (int u : g[v]) if (!visto[u]) dfs(u);
}
```

5.7 Dijkstra

```cpp
typedef ll tipo;
const int MAXN = 200005;

struct arista {
    int x; tipo w; // x -> next node, w = weight
};

struct nodo {
    tipo d, v, a; // d -> distance, v -> actual node, a = previous node
    bool operator<(const nodo& x) const {return d > x.d;}
};

vector<nodo> Dijkstra(int start, int n, vector<vector<arista>> &g) {
    vector<nodo> ans(n);
    vector<bool> visto(n, false);
    priority_queue<nodo> p; p.push({0,start,-1});
    while(!p.empty()) {
        nodo it=p.top(); p.pop();
        if(visto[it.v]) continue;
        else {
            ans[it.v] = it; visto[it.v] = true;
            for(arista u : g[it.v]) {
                if(!visto[u.x]) p.push({it.d + u.w, u.x, it.v});
            }
        }
    }
    return ans;
}
```

5.8 Dominator Tree

```cpp
// Dominator Tree
// Description: Given a directed graph, find the dominator tree of the graph
// Time: O(nlogn)
// Usage: Dominator_Tree dt(n,g,root); // n --> number of nodes, g --> graph, root --> root of the tree
// If u is an ancestor of v in the dominator tree, then u dominates v


#define rz(n) resize(n)

struct Dominator_Tree {
    vector<vector<int>> g, tree, rg, bucket;
    vector<int> sdom, par, dom, dsu, label;
    vector<int> arr, rev;
    int n, T = 0, root;

    int Find(int u,int x=0) {
        if(u==dsu[u])return x?-1:u;
        int v = Find(dsu[u],x+1);
        if(v<0)return u;
        if(sdom[label[dsu[u]]]<sdom[label[u]]) {
            label[u] = label[dsu[u]];
        }
        dsu[u] = v;
        return x ? v : label[u];
    }

    void Union(int u,int v) { //Add an edge u-->v
        dsu[v] = u;
    }

    void dfs0(int u) {
        T++; arr[u] = T; rev[T] = u;
        label[T] = T; sdom[T] = T; dsu[T] = T;
        for(int i = 0; i < SZ(g[u]); i++) {
            int w = g[u][i];
            if(!arr[w]) {
                dfs0(w);
                par[arr[w]]=arr[u];
            }
            rg[arr[w]].pb(arr[u]);
        }
    }

    Dominator_Tree(int _n, vector<vector<int>> _g, int _root) {
        n = _n; root = _root;
        g = _g; tree.rz(n+5); rg.rz(n+5); bucket.rz(n+5); sdom.rz(n+5);
        par.rz(n+5); dom.rz(n+5); dsu.rz(n+5); label.rz(n+5);
```

```
        arr.rz(n+5); rev.rz(n+5);
        dfs0(root);


        for(int i = n; i >= 1; i--) {
            for(int j = 0; j < SZ(rg[i]); j++) {
                sdom[i] = min(sdom[i],sdom[Find(rg[i][j])]);
            }
            if(i>1)bucket[sdom[i]].pb(i);
            for(int j = 0; j < SZ(bucket[i]); j++) {
                int w = bucket[i][j],v = Find(w);
                if(sdom[v] == sdom[w]) dom[w] = sdom[w];
                else dom[w] = v;
            }
            if(i>1)Union(par[i],i);
        }

        forr(i, 2, n+1){
            if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];
            tree[rev[i]].pb(rev[dom[i]]);
            tree[rev[dom[i]]].pb(rev[i]);
        }

    }
};
```

5.9 Dynamic Connectivity

```
// Imprime la cantidad de componentes conexas
// Resuelve el problema offline, para agregar una query es necesario
// agregar DynCon.query() en el instante que se quiera conocer la
// respuesta.


struct UnionFind {
    int n,comp;
    vector<int> uf,si,c;
    UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1){
        forr(i,0,n)uf[i]=i;}
    int find(int x){return x==uf[x]?x:find(uf[x]);}
    bool join(int x, int y){
        if((x=find(x))==(y=find(y)))return false;
        if(si[x]<si[y])swap(x,y);
        si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
        return true;
    }
    int snap(){return c.size();}
    void rollback(int snap){
        while(c.size()>snap){
            int x=c.back();c.pop_back();
            si[uf[x]]-=si[x];uf[x]=x;comp++;
        }
    }
};
enum {ADD,DEL,QUERY};
struct Query {int type,x,y;};
struct DynCon {
    vector<Query> q;
    UnionFind dsu;
    vector<int> mt;
    map<pair<int,int>,int> last;
    DynCon(int n):dsu(n){}
    void add(int x, int y){
        if(x>y)swap(x,y);
        q.pb((Query){ADD,x,y});mt.pb(-1);last[{x,y}]=q.size()-1;
    }
    void remove(int x, int y){
        if(x>y)swap(x,y);
        q.pb((Query){DEL,x,y});
        int pr=last[{x,y}];mt[pr]=q.size()-1;mt.pb(pr);
    }
    void query(){q.pb((Query){QUERY,-1,-1});mt.pb(-1);}
    void process(){ // answers all queries in order
        if(!q.size())return;
        forr(i,0,q.size())if(q[i].type==ADD&&mt[i]<0)mt[i]=q.size();
        go(0,q.size());
    }
    void go(int s, int e){
        if(s+1==e){
            if(q[s].type==QUERY) // answer query using DSU
                cout << dsu.comp << "␣";
            return;
        }
        int k=dsu.snap(),m=(s+e)/2;
        for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s)dsu.join(q[i].x,q[i
            ].y);
```

```
        go(s,m);dsu.rollback(k);
        for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join(q[i].x,q[i].y);
        go(m,e);dsu.rollback(k);
    }
};
```

5.10 Eulerian Cycle

```
// Todos tienen que cumplir: SZ(path) == tot_edges+1
//
// Existencia directed:
// Path: Que el start tenga out_deg=in_deg+1
//       Que el final tenga in_deg=out_deg+1
//       Que el resto tenga in_deg=out_deg
//
// Cycle: Todos los nodos in_deg=out_deg
//
// Existencia undirected:
// Path: Que el start y final tengan deg impar, el resto todo par
// Cycle: Todos los nodos in_deg=out_deg

// Directed version (uncomment commented code for undirected)
struct edge {
  int y;
//  list<edge>::iterator rev;
  edge(int _y):y(_y){}
};
list<edge> g[MAXN];
void add_edge(int a, int b){
  g[a].push_front(edge(b));//auto ia=g[a].begin();
//  g[b].push_front(edge(a));auto ib=g[b].begin();
//  ia->rev=ib;ib->rev=ia;
}
vector<int> p;
void go(int x){
  while(SZ(g[x])){
    int y=g[x].front().y;
    //g[y].erase(g[x].front().rev);
    g[x].pop_front();
    go(y);
  }
  p.pb(x);
}
```

```
vector<int> get_path(int x){ // get a path that begins in x
// check that a path exists from x before calling to get_path!
  p.clear();go(x);reverse(all(p));
  return p;
}
```

5.11 Blossom

```
/* Encuentra un matching en un grafo no bipartito
 * en O(nm) mediante el algoritmo de Edmond's blossom
 * Los matchings estan en (i, mate[i]), con i < mate[i]
 */
vi Blossom(vector<vi> &g) {
  int n = SZ(g), timer = -1;
  vi mate(n, -1), label(n), parent(n),
     orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.pb(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(all(label), -1);
    iota(all(orig), 0);
    q.clear();
    label[root] = 0; q.pb(root);
    forn(i, SZ(q)){
```

```
      int v = q[i];
      for (auto x : g[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.pb(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal matching.
  forn(i, n) if(mate[i] == -1) bfs(i);
  return mate;
}
```

5.12 Dinic

```
// Dinic: Max Flow en O(V^2 E). Para el grafo bipartito con source
// y sink dummy, funciona en O(sqrt(V) E). Equivalente a Hopcroft-Karp.

// Matching: aristas saturadas (que no incluyan source/sink)
// Min cut: nodos con dist>=0 vs nodos con dist<0
// MVC: Nodos izquierda con dist<0 + nodos derecha con dist>0
// Maximum Independent Set: complemento de MVC (N-MVC)
struct Dinic{
    int nodes,src,dst;
    vector<int> dist,q,work;
    struct edge { int to,rev; ll f,cap; };
    vector<vector<edge>> g;
    Dinic(int x):nodes(x),dist(x),q(x),work(x),g(x){}
    void add_edge(int s, int t, ll cap){
        g[s].pb({t,SZ(g[t]),0,cap});
        g[t].pb({s,SZ(g[s])-1,0,0});
    }
    bool dinic_bfs(){
        fill(all(dist),-1);dist[src]=0;
        int qt=0;q[qt++]=src;
        forn(qh, qt){
            int u=q[qh];
            forn(i,SZ(g[u])){
                edge &e=g[u][i];int v=g[u][i].to;
                if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
            }
        }
        return dist[dst]>=0;
    }
    ll dinic_dfs(int u, ll f){
        if(u==dst)return f;
        for(int &i=work[u];i<SZ(g[u]);i++){
            edge &e=g[u][i];
            if(e.cap<=e.f)continue;
            int v=e.to;
            if(dist[v]==dist[u]+1){
                ll df=dinic_dfs(v,min(f,e.cap-e.f));
                if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
            }
        }
        return 0;
    }
    ll max_flow(int _src, int _dst){
        src=_src;dst=_dst;
        ll result=0;
        while(dinic_bfs()){
            fill(all(work),0);
            while(ll delta=dinic_dfs(src,INF))result+=delta;
        }
        return result;
    }
};
```

5.13 Hopcroft Karp

```
/**
 * Fast bipartite matching algorithm. Graph $g$ should be a list
 * of neighbors of the left partition, and $btoa$ should be a vector
     full of
 * -1's of the same size as the right partition. Returns the size of
 * the matching. $btoa[i]$ will be the match for vertex $i$ on the right
      side,
 * or $-1$ if it's not matched.
 * Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
```

```
 * Time: O(\sqrt{V}E)
 */
bool dfs(int a, int L, vector<vi> &g, vi &btoa, vi &A, vi &B) {
    if(A[a] != L) return 0;
    A[a] = -1;
    for(auto b: g[a]) if(B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L+1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

int hopcroftKarp(vector<vi> &g, vi &btoa) { // bipartite matching rapido
    int res = 0;
    vi A(SZ(g)), B(SZ(btoa)), cur, next;
    for(;;) {
        fill(all(A), 0); fill(all(B), 0);
        /// Find the starting nodes for BFS (i.e. layer 0).
        cur.clear();
        for(auto a : btoa) if(a != -1) A[a] = -1;
        forn(a, SZ(g)) if(A[a] == 0) cur.pb(a);
        /// Find all layers using bfs.
        for(int lay = 1;; lay++) {
            bool islast = 0;
            next.clear();
            for(auto a: cur) for(auto b: g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay; islast = 1;
                } else if (btoa[b] != a && !B[b]) {
                    B[b] = lay; next.pb(btoa[b]);
                }
            }
            if(islast) break;
            if(next.empty()) return res;
            for(auto a : next) A[a] = lay;
            cur.swap(next);
        }
        /// Use DFS to scan for augmenting paths.
        forn(a, SZ(g)) res += dfs(a, 0, g, btoa, A, B);
    }
}
```

5.14 Hungarian

```
/* Algoritmo O(n^3) para assignment problem
 * Inspirando el el notebook vasito, quien se inspiro
 * en http://e-maxx.ru/algo/assignment_hungary
 */
typedef long double td; typedef vector<td> vd;
const td INF=1e100;//for maximum set INF to 0, and negate costs
bool zero(td x){return fabs(x)<1e-9;}//change to x==0, for ints/ll
struct Hungarian{
    int n; vector<vd> cs; vi L, R;
    Hungarian(int N, int M):n(max(N,M)),cs(n,vd(n)),L(n),R(n){
        forn(x,N)forn(y,M)cs[x][y]=INF;
    }
    void set(int x,int y,td c){cs[x][y]=c;}
    td assign() {
        int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
        forn(i,n)u[i]=*min_element(all(cs[i]));
        forn(j,n){v[j]=cs[0][j]-u[0];forr(i,1,n)v[j]=min(v[j],cs[i][j]-u
            [i]);}
        L=R=vi(n, -1);
        forn(i,n)forn(j,n)
            if(R[j]==-1&&zero(cs[i][j]-u[i]-v[j])){L[i]=j;R[j]=i;mat++;
                break;}
        for(;mat<n;mat++){
            int s=0, j=0, i;
            while(L[s] != -1)s++;
            fill(all(dad),-1);fill(all(sn),0);
            forn(k,n)ds[k]=cs[s][k]-u[s]-v[k];
            for(;;){
                j = -1;
                forn(k,n)if(!sn[k]&&(j==-1||ds[k]<ds[j]))j=k;
                sn[j] = 1; i = R[j];
                if(i == -1) break;
                forn(k,n)if(!sn[k]){
                    auto new_ds=ds[j]+cs[i][k]-u[i]-v[k];
                    if(ds[k] > new_ds){ds[k]=new_ds;dad[k]=j;}
                }
            }
            forn(k,n)if(k!=j&&sn[k]){auto w=ds[k]-ds[j];v[k]+=w,u[R[k
                ]]-=w;}
            u[s] += ds[j];
            while(dad[j]>=0){int d = dad[j];R[j]=R[d];L[R[j]]=j;j=d;}
```

```
            R[j]=s;L[s]=j;
        }
        td value=0;forn(i,n)value+=cs[i][L[i]];
        return value;
    }
};
```

5.15 Maximum Bipartite Matching

```
/*
Algoritmo de Kuhn para bipartite matching, se le pasa el grafo
y encuentra el matching maximo en O(VE).
La pareja de cada nodo se guarda en match[i], sino -1
*/

struct bipartite_matching {
  int n;
  vector<vi> g; // 0-indexed
  vb vis; vi match;

  bipartite_matching(int _n, vector<vi> _g): n(_n), g(_g) {
    vis.resize(n), match.resize(n);
  }

  bool dfs(int node){
    if(vis[node])return 0;
    vis[node] = 1;
    for(auto nx : g[node]){
      if(match[nx]==-1 || dfs(match[nx])){
        match[node] = nx; match[nx] = node;
        return 1;
      }
    }
    return 0;
  }

  int solve() { // toma los nodos de 0 a n-1
    fill(all(match),-1);
    while(true) {
      fill(all(vis),false);
      bool cont = 0;
      forn(i, n) if(match[i] == -1) cont |= dfs(i);
      if(cont==0) break;
    }
    int matching = 0;
    forn(i, n) if(match[i] != -1) matching++;
    return matching; // hay que dividir por 2 si quiero la cantidad de
        parejas
  }
};
```

5.16 Max Flow

```
/*
Algoritmo de Edmonds-Karp, halla el max flow en O(VE^2).
Para eso, va eligiendo caminos de aumento con la menor cantidad de
    aristas
en el grafo residual.

El min_cut esta formado por las aristas que unen un nodo alcanzable por
s en el grafo residual final, con un nodo inalcanzable.
*/
typedef long long tipo; // el tipo en que se mide el flow

struct max_flow { // Edmonds-Karp, O(VE^2)
    int n;
    vector<vector<int>> g;
    vector<vector<tipo>> cap;
    // si n es grande (>5000), hay que usar un map de capacidades

    max_flow(int n): n(n){
        g.resize(n), cap.resize(n, vi(n));
    }
    void add_edge(int x, int y, tipo z){
        g[x].pb(y), g[y].pb(x);
        cap[x][y] += z;
    }
    tipo bfs(int s, int t, vector<int> &parent){
        fill(all(parent), -1);
        queue<pair<int, tipo>> q; q.push({s, INF});
        parent[s] = s;

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

```

```
            for(int next: g[cur]){
                if(parent[next] == -1 && cap[cur][next]) {
                    parent[next] = cur;
                    tipo new_flow = min(flow, cap[cur][next]);
                    if(next == t) return new_flow;
                    q.push({next,new_flow});
                }
            }
        }
        return 0; // no encontre aug paths
    }

    tipo get_max_flow(int s, int t){
        tipo flow = 0, new_flow;
        vector<int> parent(n);

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

    void remove_dups(vector<int> &a){
        sort(all(a)); a.erase(unique(all(a)), a.end());
    }

    vector<pair<int, int>> get_min_cut(int s, int t) {
        // cambiar ans a set<pair<int, int>>
        // si no se quiere usar remove dups
        vector<pair<int, int>> ans;
        for(auto &x: g) remove_dups(x);

        get_max_flow(s, t); // si ya se corrio afuera, comentar

        vector<int> parent(n);
        bfs(s, t, parent); // hago bfs en el grafo residual final
        forn(v, n){
            for(auto u: g[v]){
                if(parent[v] != -1 && parent[u] == -1){
                    ans.pb({v, u});
                }
            }
        }
        return ans;
    }
};

// encontrar paths disjuntos (si tienen cap 1)
// ini_cap es lo mismo que cap antes de correr el flow.
vector<int> path;
bool find_paths(int v, max_flow &mf){
    path.pb(v);
    if(v == mf.n-1) return true;
    for(int u : mf.g[v]) {
        if(mf.ini_cap[v][u] && !mf.cap[v][u]){
            mf.cap[v][u] = 1; // marco la arista como usada (
                dessaturandola)
            if(find_paths(u, mf)) return true;
            mf.cap[v][u] = 0;
        }
    }
    path.pop_back();
    return false;
}
```

5.17 Max Flow Min Cost

```
typedef long long tipo;
typedef double tipo_cost;
const int MAXN = 505;
tipo INF = (tipo)(1e9+7);
tipo_cost MAX_COST = (double)(1e16);

struct arista {
  int v; // Next node
  tipo_cost c; // Cost
};

vector < vector<arista> > g(MAXN);
```

```
map < pair<int,int>,tipo > cap;
map < pair<int,int>,tipo_cost > cost;

void add_edge(int x, int y, tipo z, tipo_cost c) {
  g[x].pb({y,c});
  g[y].pb({x,-1.0*c});
  cap[{x,y}] += z;
  cost[{x,y}] += c;
  cost[{y,x}] -= c;
}

void Bellman_Ford(int s, int t, vector <int> &parent, vector <tipo_cost>
     &d) {
  fill(all(parent),-1);
  fill(all(d),MAX_COST);
  vector <bool> inq(MAXN,false);
  queue <int> q; q.push(s); parent[s] = s; inq[s] = true; d[s] = 0;

  while(q.size() > 0) {
    int u = q.front(); q.pop(); inq[u] = false;
    for(arista next : g[u]) {
      if(cap[{u,next.v}]>0 && d[next.v] > d[u] + next.c) {
        d[next.v] = d[u] + next.c;
        parent[next.v] = u;
        if(!inq[next.v]) {
          inq[next.v] = true;
          q.push(next.v);
        }
      }
    }
  }
}


tipo_cost max_flow_min_cost(int s, int t) {
  tipo flow = 0;
  tipo_cost min_cost = 0;
  vector <int> parent(MAXN);
  vector <tipo_cost> d(MAXN);

  while(true) {
    Bellman_Ford(s,t,parent,d);
    if(d[t] == MAX_COST) break;

    tipo new_flow = INF;
    int cur = t;
    while(cur != s) {
      new_flow = min(new_flow,cap[{parent[cur],cur}]);
      cur = parent[cur];
    }

    flow += new_flow;
    min_cost += d[t] * (tipo_cost)(new_flow);
    cur = t;
    while(cur != s) {
      int prev = parent[cur];
      cap[{prev,cur}] -= new_flow;
      cap[{cur,prev}] += new_flow;
      cur = prev;
    }
  }
  return min_cost;
}
```

5.18 Max Flow Min Cost Multiedge

```
typedef long long tipo;
typedef long long tipo_cost;
const int MAXN = 505;
tipo INF = (tipo)(1e9+7);
tipo_cost MAX_COST = (tipo_cost)(1e16);

struct arista {
  int v;
  tipo cap;
  tipo_cost c; // Cost
  int id;
};

struct prev_node {
  int prev, id;
  void setting() {prev = -1, id = -1;}
};

vector < vector<arista> > g(MAXN);
vector <tipo> cap;
```

```
21
22 void add_edge(int x, int y, tipo z, tipo_cost c) {
23   int id = cap.size();
24   g[x].pb({y,z,c,id});
25   g[y].pb({x,0,-1*c,id+1});
26   cap.pb(z); cap.pb(0);
27 }
28
29 void Bellman_Ford(int s, int t, vector <prev_node> &parent, vector <
       tipo_cost> &d) {
30   for(prev_node u : parent) u.setting();
31   fill(all(d),MAX_COST);
32   vector <bool> inq(MAXN,false);
33   queue <int> q; q.push(s); parent[s] = {s,-1}; inq[s] = true; d[s] = 0;
34
35   while(q.size() > 0) {
36     int u = q.front(); q.pop(); inq[u] = false;
37     for(arista next : g[u]) {
38       if(cap[next.id]>0 && d[next.v] > d[u] + next.c) {
39         d[next.v] = d[u] + next.c;
40         parent[next.v] = {u,next.id};
41         if(!inq[next.v]) {
42           inq[next.v] = true;
43           q.push(next.v);
44         }
45       }
46     }
47   }
48 }
49
50
51 tipo_cost max_flow_min_cost(int s, int t, int total) {
52   tipo flow = 0;
53   tipo_cost min_cost = 0;
54   vector <prev_node> parent(MAXN);
55   vector <tipo_cost> d(MAXN);
56
57   while(true) {
58     Bellman_Ford(s,t,parent,d);
59     if(d[t] == MAX_COST) break;
60
61     tipo new_flow = INF;
62     int cur = t;
63     while(cur != s) {
64       new_flow = min(new_flow,cap[parent[cur].id]);
65       cur = parent[cur].prev;
66     }
67
68     flow += new_flow;
69     min_cost += d[t] * (tipo_cost)(new_flow);
70     cur = t;
71     while(cur != s) {
72       int prev = parent[cur].prev;
73       int id = parent[cur].id;
74       cap[id] -= new_flow;
75       cap[(id^1)] += new_flow;
76       cur = prev;
77     }
78   }
79   if(flow < total) return -1;
80   return min_cost;
81 }
```

5.19 Min Cost Flow

```
1  /*
2  Max flow min cost, trabaja con Dijkstra y funciones potenciales
3  en O(n^3 m).
4  Tolera multiedge, pero no ciclos negativos de costo (esto es complicado
       en gral.)
5
6  Si es necesario el min cost flow (para un flow K fijo),
7  agregar una sink mas t', unir t->t' con una arista de cap K y costo 0
8  y correr flow entre s y t'.
9  */
10 typedef ll tf; // tipo que se usa para el flow
11 typedef ll tc; // tipo que se usa para el cost
12 const tf INFFLOW=1e9;
13 const tc INFCOST=1e9;
14 struct MCF {
15   int n;
16   vector<tc> prio, pot; vector<tf> curflow; vector<int> prevedge,
       prevnode;
17   priority_queue<pair<tc, int>, vector<pair<tc, int>>, greater<pair<tc,
       int>>> q;
18   struct edge{int to, rev; tf f, cap; tc cost;};
```

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

5.20 Floyd Warshall

```
//~ Minino camino entre todos los nodos
//~ O(n^3) - Tambien detecta ciclos negativos

void floyd(vector<vector<int>>&matriz, ll n) {
    forn(k,n) forn(i,n) forn(j,n) {
        matriz[i][j]=min(matriz[i][j],matriz[i][k]+matriz[k][j]);
    }
}
```

5.21 Kruskal

```
struct arista {
    int u, v;
    ll w;
    bool operator<(const arista& other) const { return w < other.w; }
};
struct DSU {
    vi link, sz;

    DSU(int tam) {
        link.resize(tam+5), sz.resize(tam+5);
        forn(i, tam+5) link[i] = i, sz[i] = 1;
    }

    ll find(ll x){ return link[x] = (link[x] == x ? x : find(link[x]));
        }
    bool same(ll a, ll b) { return find(a) == find(b); }

    void join(ll a, ll b) {
        a = find(a), b = find(b);
        if(a == b) return;
        if(sz[a] < sz[b]) swap(a,b);
        sz[a] += sz[b];
        link[b] = a;
    }
};

vector<arista> kruskal(int n, vector<arista> &g) {
    sort(all(g)); DSU dsu(n);
    vector<arista> mst;
    for(arista e : g) {
```

```
        if(!dsu.same(e.v, e.u)){
            dsu.join(e.v, e.u); mst.pb(e);
        }
    }
    return mst;
}
```

5.22 Prim

```
//Prim Algorithm

//Dado un grado no direccionado, halla el arbol tal que la suma de
//los pesos de las aristas sea minimo

//Metodologia: insertamos un nodo cualquiera, agregamos en un priority
//queue las aristas, ordenadas desde el menor peso hasta el mayor.
//insertamos la menor arista y agregamos el otro extremo como un nodo
//nuevo. Iteramos estos pasos hasta obtener las n-1 aristas.

//El algoritmo retorna la lista con las arista que generan el minimum
//Spanning Tree

vector <pair<ll,ll>> v[500005];
vector <pair<ll,ll>> a;
vector <bool> visto(500005,false);
map <pair<ll,ll>,bool> mapa;

struct arista
{
    ll n, m; // n=nodo padre, m=nodo nuevo
    ll peso;
};

struct compare
{
    bool operator()(arista a, arista b)
    {
        if(a.peso!=b.peso) return a.peso<b.peso;
        else return true;
    }
};

vector <pair<ll,ll>> Prim(ll n)
{
    vector <pair<ll,ll>> lista;
    visto[n]=true;
    set <arista,compare> s;
    forn(i,v[n].size())
    {
        arista aux;
        aux.n=n; aux.m=v[n][i].F; aux.peso=v[n][i].S;
        s.insert(aux);
    }
    while(s.size()!=0)
    {
        auto it=s.begin();
        if(visto[it->m]==true) {s.erase(it); continue;}
        visto[it->m]=true;
        ll ini=it->n, fin=it->m, p=it->peso;
        lista.pb(mp(ini,fin));
        forn(i,v[fin].size())
        {
            arista aux;
            aux.n=fin; aux.m=v[fin][i].F; aux.peso=v[fin][i].S;
            if(visto[aux.m]==false) s.insert(aux);
        }
    }
    return lista;
}
```

5.23 Toposort

```
// Topological sort
// Complexity: O(n+m)
vi tsort(vector<vi> &g, int n){  // lexicographically smallest
    topological sort
    vi r; priority_queue<ll> q;
    vi d(2*n);
    forn(i, n) for(ll j: g[i]) d[j]++;
    forn(i, n) if(!d[i]) q.push(-i);
    while(!q.empty()){
        ll x=-q.top(); q.pop(); r.pb(x);
        for(ll j: g[x]){
            d[j]--;
            if(!d[j]) q.push(-j);
        }
```

```
    }
    return r;  // if not DAG it will have less than n elements
}
```

6 Math

6.1 Bell

```
// Los numeros de Bell son utiles para calcular de cuantas maneras
// podemos subdivir el conjunto utilizando todos los elementos

// O(n^2)
// Numeros de Bell
// Bell numbers = {1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, ...}
// bell[i][j] = bell[i-1][j-1] + bell[i][j-1]
// bell[i][0] = bell[i-1][i-1]

vector<int> bellNumber(int n) {
    vector<vector<int>> bell(n+1,vector<int>(n+1));
    vector<int> ans(n+1,0); // Numeros de bell desde 0 hasta n
    bell[0][0] = 1;
    forr(i, 1, n+1){
        bell[i][0] = bell[i-1][i-1];
        forr(j, 1, i+1) bell[i][j] = bell[i-1][j-1] + bell[i][j-1];
    }
    forn(i, n+1) ans[i] = bell[i][0];
    return ans;
}
```

6.2 Berlekamp Massey

```
// Berlekamp-Massey algorithm
// Description: Finds the shortest linear recurrence relation that
    generates a given sequence.
// Time: O(N^2)
// Output: A vector of integers, where the first element is the constant
     term and the last element is the coefficient of x^(n-1).

vi BM(vi x){
    vi ls,cur; ll lf,ld;
    forn(i,SZ(x)){
        ll t=0;
        forn(j,SZ(cur)) t=(t+x[i-j-1]*(ll)cur[j])%MOD;
        if((t-x[i])%MOD==0) continue;
        if(!SZ(cur)){
            cur.resize(i+1); lf=i; ld=(t-x[i])%MOD;
            continue;
```

```
        }
        ll k=-(x[i]-t)*be(ld,MOD-2,MOD)%MOD;
        vi c(i-lf-1); c.pb(k);
        forr(j,0,SZ(ls))c.pb(-ls[j]*k%MOD);
        if(SZ(c)<SZ(cur))c.resize(SZ(cur));
        forr(j,0,SZ(cur))c[j]=(c[j]+cur[j])%MOD;
        if(i-lf+SZ(ls)>=SZ(cur)) ls=cur,lf=i,ld=(t-x[i])%MOD;
        cur=c;
    }
    forn(i,SZ(cur)) cur[i]=(cur[i]%MOD+MOD)%MOD;
    return cur;
}
```

6.3 Catalan

```
// Catalan dp

vector<ll> catalan(int n) { //O(N^2)
    vector<ll> c(n+1,0); c[0] = c[1] = 1;
    forr(i,2,n+1) forn(j,i) c[i] += c[j] * c[i-j-1];
    return c;
}

// Catalan formula
// binom(2n,n) / (n+1) --- (2n!) / [(n+1)! * n!]

ll catalan(ll n) {
  return (((fact[2*n] * ifact[n+1])%MOD)*ifact[n])%MOD;
}
```

6.4 Coeficientes Binomiales

```
//~ Solucion DP

vector<vector<ll>> binomial(int n) { // O (N^2)
    vector<vector<ll>> binom(n+1,vector<ll>(n+1));
    binom[0][0] = binom[1][0] = binom[1][1] = 1;
    forr(i,2,n+1){ forn(j,i+1) {
            if(j==0 || j==i) binom[i][j]=1;
            else binom[i][j] = binom[i-1][j-1] + binom[i-1][j];
        }
    }
    return binom;
}
```

```

//~ ================================================================

//~ Solucion O(N + Log N)

vector<ll> fact, ifact;

ll be(ll b, ll e, ll m) {
    ll r=1; b%=m;
    while(e){if(e&1LL)r=r*b%m;b=b*b%m;e/=2;}
    return r;
}
ll inv_mod(ll x, ll m){ return be(x,m-2,m); }

void init_fact(int maxn){
    fact.resize(maxn), ifact.resize(maxn);
    fact[0] = 1; forr(i, 1, maxn) fact[i] = (fact[i-1]*i)%MOD;
    ifact[maxn-1] = inv_mod(fact[maxn-1], MOD);
    for(int i = maxn-2; i >= 0; i--) ifact[i] = (ifact[i+1]*(i+1))%MOD;
}

ll binom(ll n, ll k){
    if(n < 0 or k < 0 or n < k) return 0;
    ll ans = fact[n];
    ans *= ifact[k]; ans %= MOD;
    ans *= ifact[n-k]; ans %= MOD;
    return ans;
}
```

6.5 Fft

```
typedef long long tipo;

struct CD{
    double r,i;
    CD(double r=0,double i=0):r(r),i(i){}
    double real()const{return r;}
    void operator /=(const tipo c){r/=c; i/=c;}
};
CD operator*(const CD& a, const CD& b){return CD(a.r*b.r-a.i*b.i,a.r*b.i
    +a.i*b.r);}
CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,a.i-b.i);}
```

```cpp
const double pi=acos(-1);
const tipo MAXN=1<<21;
CD cp1[MAXN+9],cp2[MAXN+9];
tipo R[MAXN+9];

void dft(CD* a, tipo n, bool inv){
    forn(i, n) if(R[i]<i) swap(a[R[i]],a[i]);
    for(int m=2;m<=n;m*=2){
        double z=2*pi/m*(inv?-1:1);
        CD wi=CD(cos(z),sin(z));
        for(int j=0;j<n;j+=m){
            CD w(1);
            for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
                CD u=a[k]; CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
            }
        }
    }
    if(inv) forn(i,n) a[i]/=n;
}
vector<tipo> multiply(vector<tipo> &p1, vector<tipo> &p2){
    tipo n=SZ(p1)+SZ(p2)+1;
    tipo m=1,cnt=0;
    while(m<=n) m+=m,cnt++;
    forn(i, m){R[i]=0;forr(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
    forn(i, m) cp1[i]=cp2[i]=0;
    forn(i, SZ(p1)) cp1[i]=p1[i];
    forn(i, SZ(p2)) cp2[i]=p2[i];
    dft(cp1,m,0); dft(cp2,m,0);
    forn(i, m) cp1[i]=cp1[i]*cp2[i];
    dft(cp1,m,1);
    vector<tipo> res;
    n -= 2;
    forn(i,n) res.pb((ll)floor(cp1[i].real()+.5));
    return res;
}
```

6.6 Fft Mod

```cpp
/**
Implementacion iterativa de FFT and FFTmod. O(N log N)

La precision de la FFT depende de la respuesta.
x <= 5e14 (double), x <= 1e18(long double)
donde x = max(ans[i]) para la FFT, y x = N*mod for FFTmod
**/

typedef double ld;
typedef complex<ld> cd;

void fft(vector<cd> &a){
    int n=SZ(a), L=31-__builtin_clz(n);
    vi rev(n);
    forn(i, n) rev[i]=(rev[i/2]+((i&1)<<L))/2;
    forn(i, n) if(i<rev[i]) swap(a[i],a[rev[i]]);
    static vector<cd> root(2,1);
    for(static int k=2;k<n;k*=2){
        root.resize(n);
        cd z=polar(1.0,acos(-1)/k);
        forr(i, k, 2*k) root[i]=root[i/2]*((i&1)?z:1);
    }
    for(int k=1;k<n;k*=2){
        for(int i=0;i<n;i+=2*k){
            forn(j, k){
                cd z=root[j+k]*a[i+j+k];
                a[i+j+k]=a[i+j]-z;
                a[i+j]+=z;
            }
        }
    }
}

vi mul(vi &a, vi &b){
    int sa=SZ(a), sb=SZ(b);
    if(sa==0||sb==0) return {};
    int n=1<<(32-__builtin_clz(sa+sb-2));
    vector<cd> f(n),h(n);
    forn(i, n) f[i]=cd(ld(i<sa?a[i]:0),ld(i<sb?b[i]:0));
    fft(f);
    forn(i, n){
        ll j=(-i)&(n-1);
        h[i]=(f[j]*f[j]-conj(f[i]*f[i]))*cd(0,-0.25/n);
    }
    fft(h);
    vi c(sa+sb-1);
    forn(i, sa+sb-1) c[i]=ll(round(h[i].real()));
    return c;
```

```
}

vi mul_mod(vi &a, vi &b, ll mod){
    int sa=SZ(a), sb=SZ(b);
    if(sa==0||sb==0) return {};
    vi a1(sa),a2(sa),b1(sb),b2(sb);
    forn(i, sa) a1[i]=(a[i]&((1<<15)-1)),a2[i]=(a[i]>>15);
    forn(i, sb) b1[i]=(b[i]&((1<<15)-1)),b2[i]=(b[i]>>15);
    vi c1=mul(a1,b1),c2=mul(a1,b2),c3=mul(a2,b1),c4=mul(a2,b2);
    vi c(sa+sb-1);
    forn(i, sa+sb-1) c[i]=(c1[i]+((1<<15)*(c2[i]%mod+c3[i]%mod))
        +((1<<30)*(c4[i]%mod)))%mod;
    return c;
}
```

6.7 Fht

```
ll c1[MAXN+9],c2[MAXN+9];  // MAXN must be power of 2 !!
void fht(ll* p, int n, bool inv){
    for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)forn(j,l){
        ll u=p[i+j],v=p[i+l+j];
        if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
        else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
        //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
        //else p[i+j]=-u+v,p[i+l+j]=u;
        //if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
        //else p[i+j]=v,p[i+l+j]=u-v;
    }
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vi multiply(vi &p1, vi &p2){
    int n=1<<(32-__builtin_clz(max(SZ(p1),SZ(p2))-1));
    forn(i,n)c1[i]=0,c2[i]=0;
    forn(i,SZ(p1))c1[i]=p1[i];
    forn(i,SZ(p2))c2[i]=p2[i];
    fht(c1,n,false);fht(c2,n,false);
    forn(i,n)c1[i]*=c2[i];
    fht(c1,n,true);
    return vi(c1,c1+n);
}
```

6.8 Fracciones

```
struct frac {
    ll n, d;
    frac(ll x, ll y) {
        ll g = __gcd(x,y);
        n = x/g; d = y/g;
        if(d < 0) n *= -1LL, d *= -1LL;
    }
    bool const operator <(const frac &a) const {
        return n * a.d < d * a.n; }
    bool const operator ==(const frac &a) const {
        return n * a.d == d * a.n; }
    frac const operator +(const frac &a) const {
        return frac(n*a.d + a.n*d, d*a.d); }
    frac const operator -(const frac &a) const {
        return frac(n*a.d - a.n*d, d*a.d); }
    frac const operator *(const frac &a) const {
        return frac(n * a.n, d * a.d); }
    frac const operator /(const frac &a) const {
        return frac(n * a.d, d * a.n); }
};
```

6.9 Gauss Sistema Ecuaciones

```
//~ const int INF = 2; // it doesn't actually have to be infinity or a
    big number

/*
Sistema de ecuaciones lineales modulares. Resuelve Ax=B.
En las filas estan las ecuaciones y en las columnas las variables.
En ans queda la solucion en caso de que sea unica. La matriz A es de n x
     m.
Esta matriz tiene n ecuaciones y m-1 variables, la ultima columna es la
    matrix b.
Complejidad O(min(n, m)nm). Si n == m, es O(n^3).
*/

ll mod(ll x) {x%=MOD; if(x<0) x+=MOD; return x;}
ll add(ll a, ll b){return mod(a+b);}
ll sub(ll a, ll b){return mod(a-b);}
ll mul(ll a, ll b){return mod(mod(a)*mod(b));}
ll be(ll a, ll p) {
  ll ans=1; for(; p; p/=2, a = mul(a,a)) if(p&1) ans=mul(ans,a);
  return ans;
```

```
}
ll divi(ll a, ll b){return mul(a,be(b,MOD-2));}

int gauss(vector<vi> &a, vi &ans) {
    int n = SZ(a), m = SZ(a[0])-1;

    vector<int> where(m, -1);
    for(int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        forr(i, row, n){ // if(abs(a[i][col]) > abs(a[sel][col]))
            if(a[i][col] > a[sel][col]) sel = int(i);
        }
        if(a[sel][col] == 0) continue; // if(abs(a[sel][col]) < EPS)
        forr(i, col, m+1) swap(a[sel][i], a[row][i]);
        where[col] = row;

        forn(i, n){
            if(i != row) {
                ll c = divi(a[i][col], a[row][col]); // double c = a[i][
                    col] / a[row][col];
                forr(j, col, m+1) a[i][j] = sub(a[i][j], mul(a[row][j],
                    c));
            }
        }
        ++row;
    }

    ans.assign(m, 0);
    forn(i, m) if(where[i] != -1) ans[i] = divi(a[where[i]][m], a[where[
        i]][i]);
    forn(i, n){
        ll sum = 0; // double sum = 0;
        forn(j, m) sum = add(sum, mul(ans[j], a[i][j]));
        if(sum - a[i][m] != 0) return 0; // if(abs(sum-a[i][m]) > EPS)
        // No hay soluciones
    }
    forn(i, m) if(where[i] == -1) return INF; // Infinitas soluciones
    return 1; // Exactamente una solucion
}

// Encuentra las bases que generan la solucion
vector<vi> find_bases(vector<vi> &A, vi &ans) {
  int n = SZ(A), m = SZ(ans);
  vi var_ind;
  forn(i,SZ(ans)) if(ans[i] == 0) var_ind.pb(i);

  vector <vi> bases(SZ(var_ind),vi(m,0));
  int r = 0;
  for(auto u : var_ind) {
    bases[r][u] = 1;
    forn(i,n) {
      forn(j,m) {
        if(A[i][j] != 0) {
          bases[r][j] = divi(-A[i][u], A[i][j]);
          break;
        }
      }
    }
    r++;
  }
  return bases;
}
```

6.10 Inverse Matrix

```
// modular functions in gauss

pair<ll,vector<vi>> inverse_matrix(vector<vector<ll> > &x) {
    // returns det and the inverse of the matrix,
    // if det == 0, no inverse
    int n = SZ(x); vector <vi> I(n,vi(n,0));
    ll det = 1;
    forn(i,n) I[i][i] = 1;
    forn(i,n) {
    int pos = -1;
        forr(fila,i,n) if(x[fila][i] > 0) pos = fila; // Not null pivot
        if(pos == -1) {return {0,I};} // If pos = -1, singular matrix
        if(pos != i) {
      swap(x[i],x[pos]), swap(I[i],I[pos]);
      det = mul(det,-1);
    }
    ll pivot = x[i][i];
    det = mul(det,pivot);
        forn(j,n) {
      x[i][j] = divi(x[i][j],pivot);
      I[i][j] = divi(I[i][j],pivot);
```

```
    }
        forn(fila,n) {
            if(fila == i) continue;
            pivot = x[fila][i];
            forn(j,n) {
        x[fila][j] = sub(x[fila][j],mul(pivot,x[i][j]));
        I[fila][j] = sub(I[fila][j],mul(pivot,I[i][j]));
      }
        }
    }
    return {det,I};
}
```

6.11 Iterate Submask

```
// 3 ^ n iteration
forn(mask, (1<<n)){
    int sm = 0;
    do {
        // do something with submask
    } while (sm=(sm-mask)&mask);
}
```

6.12 Lagrange

```
ll lagrange(vii a, ll x){
  /* Recibe una muestra a de n puntos (xi,yi)
     y evalua el polinomio de Lagrange de
     en el punto x desconocido */

  ll ans = 0;
  int n = SZ(a);
  forn(i, n){
    ll term = a[i].second;
    forn(j, n){
      if(j != i)
        term = term*(x-a[j].first)/(a[i].first-a[j].first);
    }
    ans += term;
  }
  return ans;
}
```

6.13 Linear Rec

```
const ll LOG = 60;
// Linear Recurrence
// Description: Calculates the kth term of a linear recurrence relation
// init O(n^2log) query(n^2 logk)
// input: terms: first n term; trans: transition function (calcular con
    BM); MOD; LOG=mxlog of k
// output calc(k): kth term mod MOD
// example: {1,1} {2,1} an=2*a_(n-1)+a_(n-2); calc(3)=3 calc(10007)
    =71480733
struct LinearRec {
    ll n; vi terms, trans; vector<vi> bin;
    vi add(vi &a, vi &b){
        vi res(n*2+1);
        forn(i, n+1) forn(j, n+1) res[i+j]=(res[i+j]*1LL+(ll)a[i]*b[j])%
            MOD;
        for(ll i=2*n; i>n; --i){
            forn(j, n) res[i-1-j]=(res[i-1-j]*1LL+(ll)res[i]*trans[j])%
                MOD;
            res[i]=0;
        }
        res.erase(res.begin()+n+1,res.end());
        return res;
    }
    LinearRec(vi &terms, vi &trans):terms(terms),trans(trans){
        n=SZ(trans);vi a(n+1);a[1]=1;
        bin.pb(a);
        forr(i,1,LOG)bin.pb(add(bin[i-1],bin[i-1])); // Precompute
            powers of a
    }
    ll calc(ll k){
        vi a(n+1);a[0]=1;
        forn(i,LOG) if((k>>i)&1)a=add(a,bin[i]);
        ll ret=0;
        forn(i,n) ret=(ret+a[i+1]*terms[i])%MOD;
        return ret;
    }
};
```

6.14 Matrix Fast Pow

```
typedef vector<vector<ll>> Matrix;

Matrix ones(int n) {
```

```
    Matrix r(n,vector<ll>(n));
    forn(i,n)r[i][i]=1;
    return r;
}
Matrix operator*(Matrix &a, Matrix &b) {
    int n=a.size(),m=b[0].size(),z=a[0].size();
    Matrix r(n,vector<ll>(m));
    forn(i,n)forn(j,m)forn(k,z)
        r[i][j]+=a[i][k]*b[k][j],r[i][j]%=MOD;
    return r;
}
Matrix be(Matrix b, ll e) {
    Matrix r=ones(b.size());
    while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
    return r;
}
```

6.15 Matrix Reduce

```
double reduce(vector<vector<double> > &x) { // returns determinant
    int n=x.size(),m=x[0].size();
    int i=0,j=0;double r=1.;
    while(i<n&&j<m){
        int l=i;
        forr(k,i+1,n)if(abs(x[k][j])>abs(x[l][j]))l=k;
        if(abs(x[l][j])<EPS){j++;r=0.;continue;}
        if(l!=i){r=-r;swap(x[i],x[l]);}
        r*=x[i][j];
        for(int k=m-1;k>=j;k--)x[i][k]/=x[i][j];
        forr(k,0,n){
            if(k==i)continue;
            for(int l=m-1;l>=j;l--)x[k][l]-=x[k][j]*x[i][l];
        }
        i++;j++;
    }
    return r;
}
```

6.16 Ntt

```
struct ntt {
    const int mod = 998244353;
    const int root = 15311432;
    const int root_1 = 469870224;
    const int root_pw = 1 << 23;

    void fft(vector<int> &a, bool invert) {
        int n = SZ(a);

        for(int i = 1, j = 0; i < n; i++) {
            int bit = n >> 1;
            for (; j&bit; bit >>= 1) j ^= bit;
            j ^= bit;

            if (i < j) swap(a[i], a[j]);
        }

        for(int len = 2; len <= n; len <<= 1) {
            int wlen = invert ? root_1 : root;
            for (int i = len; i < root_pw; i <<= 1)
                wlen = (int)(1LL * wlen * wlen % mod);

            for(int i = 0; i < n; i += len) {
                int w = 1;
                for (int j = 0; j < len / 2; j++) {
                    int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w %
                        mod);
                    a[i+j] = u + v < mod ? u + v : u + v - mod;
                    a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                    w = (int)(1LL * w * wlen % mod);
                }
            }
        }

        if (invert) {
            int n_1 = (int) inv_mod(n, mod);
            for (int &x : a) x = (int)(1LL * x * n_1 % mod);
        }
    }

    vector<int> multiply(vector<int> const &a, vector<int> const &b){
        vector<int> fa(all(a)), fb(all(b));
        int n = 1;
        while(n < SZ(a)+SZ(b)) n <<= 1;
        fa.resize(n), fb.resize(n);
        fft(fa, false); fft(fb, false); // transformo
        forn(i, n) fa[i] = (int) (1LL*fa[i]*fb[i]%mod);
```

```
        fft(fa, true); // anti-transformo

        vector<int> result(n);
        forn(i, n) result[i] = fa[i];
        return result;
    }
};
```

6.17 Polynomial

```
typedef int tp; // type of polynomial
template<class T=tp>
struct poly {  // poly<> : 1 variable, poly<poly<>>: 2 variables, etc.
    vector<T> c;
    T& operator[](int k){return c[k];}
    poly(vector<T>& coefs):c(coefs){}
    poly(initializer_list<T> coefs):c(coefs){}
    poly(int k):c(k){}
    poly(){}
    poly operator+(poly<T> o){
        int m=c.size(),n=o.c.size();
        poly res(max(m,n));
        forn(i,m)res[i]=res[i]+c[i];
        forn(i,n)res[i]=res[i]+o.c[i];
        return res;
    }
    poly operator*(tp k){
        poly res(c.size());
        forn(i,c.size())res[i]=c[i]*k;
        return res;
    }
    poly operator*(poly o){
        int m=c.size(),n=o.c.size();
        poly res(m+n-1);
        forn(i,m)forn(j,n)res[i+j]=res[i+j]+c[i]*o.c[j];
        return res;
    }
    poly operator-(poly<T> o){return *this+(o*-1);}
    T operator()(tp v){
        T sum(0);
        for(int i=c.size()-1;i>=0;--i)sum=sum*v+c[i];
        return sum;
    }
};

// example: p(x,y)=2*x^2+3*x*y-y+4
// poly<poly<>> p={{4,-1},{0,3},{2}}
// printf("%d\n",p(2)(3)) // 27 (p(2,3))
set<tp> roots(poly<> p){ // only for integer polynomials
    set<tp> r;
    while(!p.c.empty()&&!p.c.back())p.c.pop_back();
    if(!p(0))r.insert(0);
    if(p.c.empty())return r;
    tp a0=0,an=abs(p[p.c.size()-1]);
    for(int k=0;!a0;a0=abs(p[k++]));
    vector<tp> ps,qs;
    forr(i,1,sqrt(a0)+1)if(a0%i==0)ps.pb(i),ps.pb(a0/i);
    forr(i,1,sqrt(an)+1)if(an%i==0)qs.pb(i),qs.pb(an/i);
    for(auto pt:ps)for(auto qt:qs)if(pt%qt==0){
        tp x=pt/qt;
        if(!p(x))r.insert(x);
        if(!p(-x))r.insert(-x);
    }
    return r;
}
pair<poly<>,tp> ruffini(poly<> p, tp r){ // returns pair (result,rem)
    int n=p.c.size()-1;
    vector<tp> b(n);
    b[n-1]=p[n];
    for(int k=n-2;k>=0;--k)b[k]=p[k+1]+r*b[k+1];
    return mp(poly<>(b),p[0]+r*b[0]);
}
// only for double polynomials
pair<poly<>,poly<> > polydiv(poly<> p, poly<> q){ // returns pair (
    result,rem)
    int n=p.c.size()-q.c.size()+1;
    vector<tp> b(n);
    for(int k=n-1;k>=0;--k){
        b[k]=p.c.back()/q.c.back();
        forn(i,q.c.size())p[i+k]-=b[k]*q[i];
        p.c.pop_back();
    }
    while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();
    return mp(poly<>(b),p);
}
// only for double polynomials
```

```
poly<> interpolate(vector<tp> x, vector<tp> y){ //TODO TEST
    poly<> q={1},S={0};
    for(tp a:x)q=poly<>({-a,1})*q;
    forn(i,x.size()){
        poly<> Li=ruffini(q,x[i]).fst;
        Li=Li*(1.0/Li(x[i])); // change for int polynomials
        S=S+Li*y[i];
    }
    return S;
}
```

6.18 Print Int128

```
string toString(__int128 num) {
    string str;
    do {
        int digit = num % 10;
        str = to_string(digit) + str;
        num = (num - digit) / 10;
    } while (num != 0);
    return str;
}

ostream& operator<<(std::ostream& os, __int128 t) {
    string str = toString(t);
    return os << str;
}
```

6.19 Random

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int my_random(int l, int r){ // Random number in [l, r]
    return uniform_int_distribution<int>(l, r)(rng);
}
```

6.20 Shortest Path Matrix Exp

```
#define INF 2e18

typedef vector<vector<ll> > Matrix;

ll funcion(ll x, ll y){
    if(x!=INF && y!=INF) return (x+y);
    else return INF;
}

Matrix operator*(Matrix &a, Matrix &b) {
    int n=a.size(),m=b[0].size(),z=a[0].size();
    Matrix r(n,vector<ll>(m,INF));
    forn(i,n) forn(j,m) forn(k,z)
        r[i][j]=min(r[i][j],funcion(a[i][k],b[k][j]));
    return r;
}

Matrix be(Matrix b, ll e) {
    Matrix r(b.size(),vector<ll>(b[0].size())); e--;
    forn(i,b.size()){ forn(j,b[0].size()) r[i][j]=b[i][j]; }
    while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
    return r;
}
```

6.21 Simplex

```
struct Simplex {
    vector<int> X,Y;
    vector<vector<double>> A;
    vector<double> b,c;
    double z;
    int n,m;
    void add_equation(vector<double> v, double val){ A.pb(v); b.pb(val);
        }
  void set_objective(vector<double> _c){ c = _c; }
    void pivot(int x,int y){
        swap(X[y],Y[x]);
        b[x]/=A[x][y];
        forr(i,0,m)if(i!=y)A[x][i]/=A[x][y];
        A[x][y]=1/A[x][y];
        forr(i,0,n)if(i!=x&&abs(A[i][y])>EPS){
            b[i]-=A[i][y]*b[x];
            forr(j,0,m)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];
        forr(i,0,m)if(i!=y)c[i]-=c[y]*A[x][i];
        c[y]=-c[y]*A[x][y];
    }
    pair<double,vector<double>> simplex() {
```

```
    // maximiza c^T x, dado Ax<=b, x>=0
        // retorna par (maximum value, solution vector)
        n=b.size();m=c.size();z=0.;
        X=vector<int>(m);Y=vector<int>(n);
        forn(i,m)X[i]=i;
        forn(i,n)Y[i]=i+m;
        while(1){
            int x=-1,y=-1;
            double mn=-EPS;
            forn(i,n)if(b[i]<mn)mn=b[i],x=i;
            if(x<0) break;
            forr(i,m)if(A[x][i]<-EPS){y=i;break;}
            assert(y>=0 && "No solution found"); // no solution to Ax<=b
            pivot(x,y);
        }
        while(1){
            double mx=EPS;
            int x=-1,y=-1;
            forn(i,m)if(c[i]>mx)mx=c[i],y=i;
            if(y<0)break;
            double mn=1e200;
            forn(i,n)if(A[i][y]>EPS&&b[i]/A[i][y]<mn)mn=b[i]/A[i][y],x=i
                ;
            assert(x>=0 && "c^T x is unbounded"); // c^T x is unbounded
            pivot(x,y);
        }
        vector<double> r(m);
        forn(i,n)if(Y[i]<m)r[Y[i]]=b[i];
        return mp(z,r);
    }
};
```

6.22 Simpson

```
double integrate(double f(double), double a, double b, int n=10000){
    double r=0,h=(b-a)/n,fa=f(a),fb;
    forr(i,0,n) {fb=f(a+h*(i+1));r+=fa+4*f(a+h*(i+0.5))+fb;fa=fb;}
    return r*h/6.;
}
```

7 Number Theory

7.1 Binexp Invmod

```
ll be(ll x, ll y, ll m) {
    if (y == 0) return 1;
    ll p = be(x, y/2, m) % m;
    p = (p * p) % m;
    return (y%2 == 0)? p : (x * p) % m;
}

ll be_it(ll b, ll e, ll m) {
    ll r=1; b%=m;
    while(e){if(e&1LL)r=r*b%m;b=b*b%m;e/=2;}
    return r;
}

ll inv_mod(ll x, ll m) {return be(x,m-2,m);}
```

7.2 Criba Phi Euler

```
bool is_composite[1000];
int phi[1000];
vector<long long> prime;

void sieve (int n) {
    fill(is_composite, is_composite + n, false);
    phi[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back (i);
            phi[i] = i - 1; //i is prime
        }
        for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];  //prime[j]
                    divides i
                break;
            } else {
                phi[i * prime[j]] = phi[i] * phi[prime[j]]; //prime[j]
                    does not divide i
            }
```

```
        }
    }
}
```

7.3 Criba Primos

```
// Criba lineal, obtiene los primos menores al parametro
vi min_prime; // min_prime[i] contiene el menor primo que divide a i,
    util para factorizar en log(i)

vi criba(ll n) {
    vb prime(n+1,true);
    min_prime.resize(n+1,INF);
    vi primos;
    for(ll p=2; p*p<=n; p++){
        if(!prime[p]) continue;
        for(ll i=p*p; i<=n; i += p) {
            prime[i] = false;
            min_prime[i] = min(min_prime[i],p);
        }
    }
    forr(i, 2, n+1){
        if(prime[i]) primos.pb(i), min_prime[i] = i;
    }
    return primos; // lista de primos hasta n
}
```

7.4 Diofantica

```
// Algoritmo de euclides extendido para encontrar gcd de a y b.
// En x e y se guarda una solucion particular de la ecuacion ax+by=gcd(a
    ,b)
ll gcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    ll x1, y1;
    ll d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// Soluciones a la ecuacion ax+by=c, retorna true si hay solucion
// x0 e y0 son soluciones particulares de la ecuacion.
// g es el gcd(a,b), hay solucion si y solo si divide a c.
bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0, ll &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) return false;

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
// Todas las soluciones de ax+by=c son de la forma: (x, y) = (x0+k*b/g,
    y0-k*a/g), para todo k
```

7.5 Discrete Log

```
/* Algoritmo de baby-step giant-step para
 * hallar x tal que a^x = b (mod m)
 * en O(sqrt(m))
 */
ll discrete_log(ll a, ll b, ll m) {
    a %= m, b %= m;
    ll n = sqrt(m) + 1;

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

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

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

```
    return -1;
}
```

7.6 Discrete Root

```
// Finds the primitive root modulo p
ll generator(ll p) {
    vi fact;
    ll phi = p-1, n = phi;
    for (ll i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        fact.push_back(n);

    forr(res, 2, p+1){
        bool ok = true;
        for (ll factor : fact) {
            if (be(res, phi / factor, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return res;
    }
    return -1;
}

// This program finds all numbers x such that x^k = a (mod n)
vi discrete_root(ll n, ll k, ll a){
    if (a == 0) return {0};

    ll g = generator(n);

    // Baby-step giant-step discrete logarithm algorithm
    ll sq = (ll) sqrt (n + .0) + 1;
    vector<pair<ll, ll>> dec(sq);
    forr(i, 1, sq+1)
        dec[i-1] = {be(g, i * sq * k % (n - 1), n), i};
    sort(all(dec));
    ll any_ans = -1;
    forn(i, sq){
        ll my = be(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(all(dec), mp(my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1) return {};

    // Print all possible answers
    ll delta = (n-1) / __gcd(k, n-1);
    vi ans;
    for (ll cur = any_ans % delta; cur < n-1; cur += delta)
        ans.push_back(be(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```

7.7 Divisors

```
vi find_divisors(ll n, vi &primos) {
  vector <pair<ll,ll>> factor;
  for(ll prime : primos) {
    int cont = 0;
    while(n % prime == 0) {
      cont++;
      n /= prime;
    }
    if(cont > 0) factor.pb({prime,cont});
  }

  if(n > 1) factor.pb({n,1});

  vi divisores = {1};
  for(auto [p,exp] : factor) {
    int tam = SZ(divisores);
    forn(i,exp) {
      forn(j,tam) {
        int pos = SZ(divisores)-tam;
        divisores.pb(divisores[pos] * p);
```

```
      }
    }
  }

  sort(all(divisores));
  return divisores;
}
```

7.8 Fast Factorization Mrabin Prho

```
typedef long double ld;
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
ll myrand(ll a, ll b) { return uniform_int_distribution<ll>(a, b)(rng);
    }

struct Factorization {
    inline ll mul(ll a, ll b, ll c){
        ll s = a * b - c * ll((ld) a / c * b + 0.5);
        return s < 0 ? s + c : s;
    }

    ll be(ll a, ll k, ll mod) {
        ll res = 1;
        for(; k; k >>= 1, a = mul(a, a, mod)) if (k & 1) res = mul(res,
            a, mod);
        return res;
    }

    bool miller(ll n) { // chequea si un numero es primo
        auto test = [&](ll n, ll a) {
            if (n == a) return true;
            if (n % 2 == 0) return false;

            ll d = (n - 1) >> __builtin_ctzll(n - 1);
            ll r = be(a, d, n);
            while (d < n - 1 && r != 1 && r != n - 1) {
                d <<= 1;
                r = mul(r, r, n);
            }
            return r == n - 1 || d & 1;
        };

        if(n == 2) return 1;
        for(auto p: vi{2, 3, 5, 7, 11, 13}) if(!test(n, p)) return 0;
        return 1;
    }

    ll pollard(ll n) { // devuelve un factor no trivial de n
        auto f = [&](ll x) { return ll((__int128(x) * x + 1) % n); };

        ll x = 0, y = 0, t = 30, prd = 2;
        while (t++ % 40 || __gcd(prd, n) == 1) { // speedup: no tomar
            gcd en cada iteracion
            if (x == y) x = myrand(2, n - 1), y = f(x);
            ll tmp = mul(prd, abs(x - y), n);
            if (tmp) prd = tmp;
            x = f(x), y = f(f(y));
        }
        return __gcd(prd, n);
    }

    vi factorize(ll n) {
        vi res;
        auto dfs = [&](auto &dfs, ll x) {
            if (x == 1) return;
            if (miller(x)) res.pb(x);
            else {
                ll d = pollard(x);
                dfs(dfs, d); dfs(dfs, x / d);
            }
        };
        dfs(dfs, n); sort(all(res));
        return res;
    }
};
```

7.9 Modint

```
// Template de benq para enteros que tienen que ser tomados MOD
// se define un tipo mi como "typedef modular<ll> mi"
template<class T> struct modular {
    T val;
    explicit operator T() const { return val; }
    modular() { val = 0; }
    template<class U> modular(const U& v) {
        val = (-MOD <= v && v <= MOD) ? v : v % MOD;
```

```
        if (val < 0) val += MOD;
    }
    friend ostream& operator<<(ostream& os, const modular& a) { return
        os << a.val; }
    friend bool operator==(const modular& a, const modular& b) { return
        a.val == b.val; }
    friend bool operator!=(const modular& a, const modular& b) { return
        !(a == b); }

    modular operator-() const { return modular(-val); }
    modular& operator+=(const modular& m) { if ((val += m.val) >= MOD)
        val -= MOD; return *this; }
    modular& operator-=(const modular& m) { if ((val -= m.val) < 0) val
        += MOD; return *this; }
    modular& operator*=(const modular& m) { val = (ll)val*m.val%MOD;
        return *this; }
    friend modular be(modular a, ll p) {
        modular ans = 1; for (; p; p /= 2, a *= a) if (p&1) ans *= a;
        return ans;
    }
    friend modular inv(const modular& a) { return be(a,MOD-2); }
    // inv is equivalent to return be(b,b.mod-2) if prime
    modular& operator/=(const modular& m) { return (*this) *= inv(m); }

    friend modular operator+(modular a, const modular& b) { return a +=
        b; }
    friend modular operator-(modular a, const modular& b) { return a -=
        b; }
    friend modular operator*(modular a, const modular& b) { return a *=
        b; }

    friend modular operator/(modular a, const modular& b) { return a /=
        b; }
};

typedef modular<ll> mi;
typedef vector<mi> vmi;
```

7.10 Mod Ops

```
ll mod(ll x) {x%=MOD; if(x<0) x+=MOD; return x;}
ll add(ll a, ll b){return mod(a+b);}
ll sub(ll a, ll b){return mod(a-b);}
ll mul(ll a, ll b){return mod(mod(a)*mod(b));}
ll be(ll a, ll p) {
  ll ans=1; for(; p; p/=2, a = mul(a,a)) if(p&1) ans=mul(ans,a);
  return ans;
}
ll divi(ll a, ll b){return mul(a,be(b,MOD-2));}
```

7.11 Moebius

```
vector<int> prime;
bool is_composite[MAXN];
int moebius[MAXN];

void sieve_moebius(int n) {
    fill(is_composite, is_composite + n, false);
    moebius[1] = 1;
    forr(i, 2, n){
        if(!is_composite[i]){
            prime.pb(i); moebius[i] = - 1; // i is prime
        }
        for(int j = 0; j < SZ(prime) && i * prime[j] < n; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                moebius[i * prime[j]] = 0;  // prime[j] divides i
                break;
            } else {
                moebius[i * prime[j]] = moebius[i] * moebius[prime[j]];
                    // prime[j] does not divide i
            }
        }
    }
}
```

7.12 Teo Chino Resto

```
// Teorema chino del resto
// Resuelve el sistema de ecuaciones modulares
// ai*xi = bi (mi)
ii extendedEuclid(ll a, ll b){ //a * x + b * y = gcd(a,b)
    ll x,y;
    if (b==0) return {1, 0};
    auto p = extendedEuclid(b, a%b);
    x = p.second, y = p.first-(a/b)*x;
    if(a*x+b*y == -__gcd(a,b)) x=-x, y=-y;
```

```
    return {x, y};
}
pair<ii, ii> dioph(ll a, ll b, ll r) {
    // a*x+b*y=r donde r es multiplo de gcd(a,b);
    ll d=__gcd(a,b);
    a/=d; b/=d; r/=d;
    auto p = extendedEuclid(a,b);
    p.first*=r; p.second*=r;
    assert(a*p.first+b*p.second==r);
    return {p, {-b,a}}; // solutions: p+t*ans.second
}
ll inv(ll a, ll mod) { // inverso de a modulo mod
    assert(__gcd(a,mod)==1);
    ii sol = extendedEuclid(a,mod);
    return ((sol.first%mod)+mod)%mod;
}

#define mod(a,m) (((a)%m+m)%m)
ii sol(tuple<ll,ll,ll> c){ //requires inv, diophantine
    auto [a, x1, m] = c; ll d = __gcd(a,m);
    if(d==1) return {mod(x1*inv(a,m),m), m};

    return x1%d ? mp(-1LL,-1LL) : sol({a/d, x1/d, m/d});
}

// cond[i] = {ai,bi,mi} ai*xi=bi (mi); assumes lcm fits in ll
// Mucho cuidado con el overflow, usar __int128 si lcm no entra en ll
pair<ll,ll> crt(vector<tuple<ll,ll,ll>> cond) { // returns: (sol, lcm)
    ll x1=0,m1=1,x2,m2;
    for(auto t:cond){
        tie(x2,m2) = sol(t);
        if((x1-x2)%__gcd(m1,m2)) return {-1,-1};
        if(m1==m2) continue;
        ll k=dioph(m2,-m1,x1-x2).first.second, l=m1*(m2/__gcd(m1,m2));
        x1 = mod((__int128)m1*k+x1,l); m1=l;
    }
    return sol({1,x1,m1});
}
```

8 Strings

8.1 Aho Corasick

```
struct vertex {
    map<char,int> next, go;
    int p,link,nextleaf;
    char pch;
    vector<int> leaf;
    vertex(int p=-1, char pch=-1,int nextleaf=-1):
        p(p),pch(pch),link(-1),nextleaf(nextleaf){}
};

vector<vertex> t;
vector <vector <int> > g(MAXN); // Suffix-links tree

void aho_init(){ //do not forget!!
    t.clear(); t.pb(vertex());
}

void add_string(string s, int id) {
    int v=0;
    for(char c:s) {
        if(!t[v].next.count(c)){
            t[v].next[c]=t.size();
            t.pb(vertex(v,c));
        }
        v=t[v].next[c];
    }
    t[v].leaf.pb(id);
}

int go(int v, char c);

int get_link(int v) {
    if(t[v].link < 0) {
        if(v == 0 || t[v].p == 0) t[v].link = 0;
        else t[v].link = go(get_link(t[v].p),t[v].pch);
        g[t[v].link].pb(v);
    }
    return t[v].link;
}

```

```
int go(int v, char c){
    if(!t[v].go.count(c)) {
        if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
        else t[v].go[c] = v == 0 ? 0 : go(get_link(v),c);
    }
    return t[v].go[c];
}

int init_next_leaf(int v) {
    if(v == 0) t[v].nextleaf=0;
    if(t[get_link(v)].leaf.size()) return t[v].nextleaf=get_link(v);

    else return t[v].nextleaf = t[get_link(v)].nextleaf != -1 ?
        t[get_link(v)].nextleaf :
            init_next_leaf(get_link(v));
}

void construct_links() { forn(i,t.size()) get_link(i); }
```

8.2 Dynamic Hash

```
ll be(ll x, ll y, ll m) {
    if (y == 0) return 1;
    ll p = be(x, y/2, m) % m;
    p = (p * p) % m;
    return (y%2 == 0)? p : (x * p) % m;
}

ll inv_mod(ll x, ll m) {return be(x,m-2,m);}

struct BIT {
  vector <ll> prefix, a; ll M;
  BIT() {}
  BIT(vector <ll> &v, ll MOD) {
    int n = v.size(); prefix.resize(n+1); a = v; M = MOD;
    vector <ll> aux(n+1,0);
    forn(i,n) aux[i+1] = (aux[i] + v[i]) % M;
    forr(i,1,n+1) prefix[i] = (aux[i] + M - aux[i - (i&(-i))]) % M;
  }
  ll query(int l, int r) { //[a,b] 0-indexed
    ll ans = 0; r++;
    while(r) ans += prefix[r], r -= r&(-r), ans %= M;
    while(l) ans += M - prefix[l], l -= l&(-l), ans %= M;
    return ans;
  }
  void update(int pos, ll val) {
    int i = pos + 1; ll upd = (val + M - a[pos]) % M;
    while(i < prefix.size()) prefix[i] += upd, prefix[i] %= M, i += i&(-
        i);
    a[pos] = val;
  }
};

struct Hashing {
  int n;
  const ll MOD[2] = {999727999, 1070777777};
  vector <ll> prefix[2], rev[2], pot[2], inv_pot[2];
  ll P = 31, invP[2] = {inv_mod(P,MOD[0]), inv_mod(P,MOD[1])};
  BIT s[2], rs[2];
  Hashing() {}
  Hashing(string &pal) {
    n = pal.size();
    forn(k,2) {
      prefix[k].resize(n,0); rev[k].resize(n,0);
      pot[k].resize(n,1); inv_pot[k].resize(n,1);
      forn(i,n) {
        if(i) pot[k][i] = (pot[k][i-1] * P) % MOD[k];
        if(i) inv_pot[k][i] = (inv_pot[k][i-1] * invP[k]) % MOD[k];
        prefix[k][i] = (ll)(pal[i]-'a') * pot[k][i] % MOD[k];
        rev[k][i] = (ll)(pal[n-i-1]-'a') * pot[k][i] % MOD[k];
      }
      s[k] = BIT(prefix[k],MOD[k]);
      rs[k] = BIT(rev[k],MOD[k]);
    }
  }
  pair<ll,ll> get(int l, int r) { //[l,r] 0-indexed
    ll x = (s[0].query(l,r) * inv_pot[0][l]) % MOD[0];
    ll y = (s[1].query(l,r) * inv_pot[1][l]) % MOD[1];
    return {x,y};
  }
  pair<ll,ll> getr(int l, int r) { //[l,r] 0-indexed
    l = n-1-l, r = n-1-r; swap(l,r);
    ll x = (rs[0].query(l,r) * inv_pot[0][l]) % MOD[0];
    ll y = (rs[1].query(l,r) * inv_pot[1][l]) % MOD[1];
    return {x,y};
  }
```

```
  void update(int pos, char a) {
    ll val = (ll)(a-'a');
    forn(k,2) {
      s[k].update(pos,(val * pot[k][pos]) % MOD[k]);
      assert(n-1-pos >= 0);
      rs[k].update(n-1-pos,(val * pot[k][n-1-pos]) % MOD[k]);
    }
  }
};
```

8.3 Find Kth Substr Repetitions

```
typedef string str;
#define RB(x) (x<n?r[x]:0)

void csort(vector<int>& sa, vector<int>& r, int k){
    int n=sa.size();
    vector<int> f(max(255,n),0),t(n);
    forr(i,0,n)f[RB(i+k)]++;
    int sum=0;
    forr(i,0,max(255,n))f[i]=(sum+=f[i])-f[i];
    forr(i,0,n)t[f[RB(sa[i]+k)]++]=sa[i];
    sa=t;
}

vector<int> suffix_array(string& s){ // O(n logn)
                                     //~ s += '$';
    int n=s.size(),rank;
    vector<int> sa(n),r(n),t(n);
    forr(i,0,n)sa[i]=i,r[i]=s[i];
    for(int k=1;k<n;k*=2){
        csort(sa,r,k);csort(sa,r,0);
        t[sa[0]]=rank=0;
        forr(i,1,n){
            if(r[sa[i]]!=r[sa[i-1]]||RB(sa[i]+k)!=RB(sa[i-1]+k))rank++;
            t[sa[i]]=rank;
        }
        r=t;
        if(r[sa[n-1]]==n-1)break;
    }
    return sa;
}

```

```
vector <ll> prefix;

ll suma(int l, int r) {
    return prefix[r+1] - prefix[l];
}

int BS(int l, int r, char &z, vector <int> &sa, str &pal, ll pos, ll k)
    {
    ll n = pal.size();
    ll a = l-1, b = r;
    while(b-a > 1) {
        ll med = (a+b) / 2;
        if((med-l+1)*(n-pos) - suma(l,med) >= k) b = med;
        else a = med;
    }
    z = pal[sa[b]+pos];
    return b;
}

int find_left(int l, int r, char z, vector<int> &sa, str &pal, int pos)
    {
    ll a = l-1, b = r;
    ll n = pal.size();
    while(b-a > 1) {
        ll med = (a+b)/2;
        if(sa[med]+pos >= n || pal[sa[med]+pos] != z) a = med;
        else b = med;
    }
    return b;
}

int find_right(int l, int r, char z, vector<int> &sa, str &pal, int pos)
     {
    ll a = l, b = r+1;
    ll n = pal.size();
    while(b-a > 1) {
        ll med = (a+b)/2;
        if(sa[med]+pos >= n || pal[sa[med]+pos] != z) b = med;
        else a = med;
    }
    return a;
}

```

```
72
73  void solve(str &pal, str &ans, int l, int r, int pos, ll k, vector <int>
        &sa) {
74      if(k <= 0) return;
75      vector <int> cont(30,0);
76      ll acum = 0; ll n = pal.size();
77      char z = '#';
78      int cr = BS(l,r,z,sa,pal,pos,k);
79      ans += z;
80
81      int new_start = find_left(l,cr,z,sa,pal,pos);
82      int new_end = find_right(cr,r,z,sa,pal,pos);
83      ll resta = (new_start - l) * (n -pos) - suma(l,new_start-1);
84      if(sa[new_start] + pos == n) new_start++;
85      solve(pal,ans,new_start,new_end,pos+1,k-resta-(new_end-new_start+1),
            sa);
86  }
87
88  string get_kth(string pal, ll k) {
89      vector <int> sa = suffix_array(pal);
90      prefix.resize(sa.size()+1,0);
91      for(int i = 0; i < sa.size(); i++) {
92          prefix[i+1] = prefix[i] + sa[i];
93      }
94      string ans;
95      solve(pal,ans,0,pal.size()-1,0,k,sa);
96
97      return ans;
98  }
```

8.4 Hashing

```
1   struct Hash {
2       const ll P=1777771;
3       const ll MOD[2] = {999727999, 1070777777};
4       const ll PI[2] = {325255434, 10018302};
5
6       vector<ll> h[2],pi[2];
7       Hash(string& s){
8           forn(k,2)h[k].resize(s.size()+1),pi[k].resize(s.size()+1);
9           forn(k,2){
10              h[k][0]=0;pi[k][0]=1;
11              ll p=1;
12              forr(i,1,s.size()+1){
13                  h[k][i]=(h[k][i-1]+p*s[i-1])%MOD[k];
14                  pi[k][i]=(1LL*pi[k][i-1]*PI[k])%MOD[k];
15                  p=(p*P)%MOD[k];
16              }
17          }
18      }
19      ll get(ll s, ll e){ // [s, e] (s y e van de 0 a n-1)
20          e++;
21          ll h0=(h[0][e]-h[0][s]+MOD[0]);
22          h0=(1LL*h0*pi[0][s])%MOD[0];
23          ll h1=(h[1][e]-h[1][s]+MOD[1]);
24          h1=(1LL*h1*pi[1][s])%MOD[1];
25          return (h0<<32)|h1;
26      }
27  };
```

8.5 Manacher

```
1   vi d1;      //d1[i] = max odd palindrome centered on i
2   vi d2;      //d2[i] = max even palindrome centered on i
3               //s  aabbaacaabbaa
4               //d1 1111117111111
5               //d2 0103010010301
6   void manacher(string& s){
7       ll l=0,r=-1,n=SZ(s);
8       d1.resize(n), d2.resize(n);
9       forn(i, n){
10          ll k = (i>r ? 1 : min(d1[l+r-i],r-i));
11          while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) k++;
12          d1[i] = k--;
13          if(i+k>r) l=i-k,r=i+k;
14      }
15      l=0; r=-1;
16      forn(i, n){
17          ll k = (i>r ? 0 : min(d2[l+r-i+1],r-i+1)); k++;
18          while(i+k<=n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
19          d2[i] = --k;
20          if(i+k-1>r) l=i-k, r=i+k-1;
21      }
22  }
```

8.6 Palindromic Tree

```cpp
struct palindromic_tree{
    static const int SIGMA=26;
    struct Node{
        int len, link, to[SIGMA];
        ll cnt;
        Node(int len, int link=0, ll cnt=1):len(len),link(link),cnt(cnt)
            {
            memset(to,0,sizeof(to));
        }
    };
    vector<Node> ns;
    int last;
    palindromic_tree():last(0){ns.pb(Node(-1));ns.pb(Node(0));}
    void add(int i, string &s){
        int p=last, c=s[i]-'a';
        while(s[i-ns[p].len-1]!=s[i])p=ns[p].link;
        if(ns[p].to[c]){
            last=ns[p].to[c];
            ns[last].cnt++;
        }else{
            int q=ns[p].link;
            while(s[i-ns[q].len-1]!=s[i])q=ns[q].link;
            q=max(1,ns[q].to[c]);
            last=ns[p].to[c]=SZ(ns);
            ns.pb(Node(ns[p].len+2,q,1));
        }
    }
};
```

8.7 Prefix Function

```cpp
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    forr(i,1,n) {
        int j = pi[i-1];
        while(j > 0 && s[i] != s[j]) j = pi[j-1];
        if(s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
```

8.8 Rabin Karp

```cpp
/*
   Dadas dos strings s y t, podemos hallar cuantas veces aparece el
       string s en el string t
   en O(|s|+|t|), mediante el uso de hashing.
   */

vector <ll> rabin_karp(string s, string t) {
    ll p=31, m=1e9+7, n=2147483647;
    ll S=s.size(), T=t.size();
    vector <ll> p_pow;
    p_pow.pb(1);
    forn(i,max(S,T)) p_pow.pb((p_pow.back()*p)%m);
    vector <ll> h; h.pb(0);
    forn(i,T) h.pb((h.back()+(t[i]-'a'+1)*p_pow[i])%m);
    ll h_s=0;
    forn(i,S) h_s=(h_s+(s[i]-'a'+1)*p_pow[i])%m;
    vector <ll> o;
    forn(i,T-S+1) {
        ll cur_h=(h[i+S]+m-h[i])%m;
        if(cur_h==(h_s*p_pow[i])%m) o.pb(i);
    }
    return 0;
}
```

8.9 Suffix Array

```cpp
#define RB(x) (x<n?r[x]:0)
void csort(vector<int>& sa, vector<int>& r, int k){
    int n=sa.size();
    vector<int> f(max(255,n),0),t(n);
    forr(i,0,n)f[RB(i+k)]++;
    int sum=0;
    forr(i,0,max(255,n))f[i]=(sum+=f[i])-f[i];
    forr(i,0,n)t[f[RB(sa[i]+k)]++]=sa[i];
    sa=t;
}

vector<int> suffix_array(string& s){ // O(n logn)
    s += '$';
    int n=s.size(),rank;
    vector<int> sa(n),r(n),t(n);
```

```
    forr(i,0,n)sa[i]=i,r[i]=s[i];
    for(int k=1;k<n;k*=2){
        csort(sa,r,k);csort(sa,r,0);
        t[sa[0]]=rank=0;
        forr(i,1,n){
            if(r[sa[i]]!=r[sa[i-1]]||RB(sa[i]+k)!=RB(sa[i-1]+k))rank++;
            t[sa[i]]=rank;
        }
        r=t;
        if(r[sa[n-1]]==n-1)break;
    }
    return sa;
}

vector<int> lcp_construction(string const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++) rank[p[i]] = i;
    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if(rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
        lcp[rank[i]] = k;
        if(k) k--;
    }
    return lcp;
}

bool substr_search(string &text, string &pal, vector <int> &sa) {
    int n = text.size(), a = -1, b = n-1;
    while(b-a > 1) {
        int med = (a+b)/2, pos = sa[med]; bool d = false;
        int tam = min((int)pal.size(),int(n-sa[med]));
        string check = text.substr(sa[med],tam);
        if(check < pal) a=med;
        else b = med;
    }
    int tam = min(int(pal.size()),int(n-sa[b]));
    string fin = text.substr(sa[b],tam);
    return fin == pal ? true : false;
}

ll count_substring(string &pal) {
    ll n = pal.size();
    vector <int> sa = suffix_array(pal);
    vector <int> lcp = lcp_construction(pal,sa);
    ll ans = n * (n+1) / 2;
    for(int u : lcp) ans -= (ll)u;
    return ans;
}

#define SZ(a) ((int)a.size())
string lcsubstr(string s, string t) {
    string r = s + '#' + t + '$';
    int best = 0; int pos = 0;
    vector <int> sa = suffix_array(r);
    vector <int> lcp = lcp_construction(r, sa);
    forr(i, 1, SZ(sa)) {
        if(isInRange(sa[i - 1], 0, SZ(s)) && isInRange(sa[i], SZ(s) + 1,
             SZ(r) - 1)) {
            if(lcp[i-1] > best) best = lcp[i-1], pos = sa[i];
        }
        if(isInRange(sa[i], 0, SZ(s)) && isInRange(sa[i - 1], SZ(s) + 1,
             SZ(r) - 1)) {
            if(lcp[i-1] > best) best = lcp[i-1], pos = sa[i];
        }
    }
    return r.substr(pos,best);
    // To find de LCSubstr between n string, we can concatenate the n
    // strings and apply min_segtree in lcp ans intervals with at least
    // one suffix of each string.
}
```

8.10 Suffix Automaton

```
struct state {
    int len, link;
    map<char,int> next;
};

const int MAXN = 300010;
```

```
state st[MAXN*2];
int sz, last;

void sa_init(){
    forn(i,2*MAXN) {
        st[i].next.clear();
        st[i].link = 0;
        st[i].len = 0;
    }
    last= st[0].len=0; sz=1;
    st[0].link=-1;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p;
    for(p = last; p != -1 && !st[p].next.count(c); p = st[p].link) {
        st[p].next[c] = cur;
    }
    if (p == -1) st[cur].link = 0;
    else {
        int q = st[p].next[c];
        if(st[p].len + 1 == st[q].len) st[cur].link = q;
        else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

/* ..... */

vector <bool> visto(2*MAXN,false);

void count_substr(int cur) {
    // Find #distinct substrings from vertex cur
    visto[cur] = true;
    st[cur].total = 1;
    for(pair<char,int> u : st[cur].next) {
        if(!visto[u.second]) count_substr(u.second);
        st[cur].total += st[u.second].total;
    }
}

/* ..... */

string ans;

void find_kth(int v, ll k) { // Find kth substring (all different)
    if(k <= 0) return;
    ll acum = 0;
    for(pair<char,int> u : st[v].next) {
        if(acum + st[u.second].total >= k) {
            ans += u.first;
            find_kth(u.second, k-acum-1);
            return;
        }
        acum += st[u.second].total;
    }
}
```

8.11 Trie

```
const int K = 26;

struct Vertex {
    int next[K];
    int leaf;

    Vertex() {
        fill(begin(next), end(next), -1);
        leaf = 0;
    }
};

vector<Vertex> trie(1);
```

```
void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (trie[v].next[c] == -1) {
            trie[v].next[c] = trie.size();
            trie.emplace_back();
        }
        v = trie[v].next[c];
    }
    trie[v].leaf++;
}

bool find(string &pal) {
    int cur = 0;
    for(char u : pal) {
        cur = trie[cur].next[u-'a'];
        if(cur == -1) return false;
    }
    return true;
}
```

8.12 Z Function

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

9 Testing

9.1 A

```
// incorrect solution for finding second smallest element
#include <bits/stdc++.h>
using namespace std;
// #warning TODO: remember about bigger N
const int MAX_N = 1e6 + 5;
int a[MAX_N];
set<int> asd[MAX_N], asdfasd[3*MAX_N];
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    sort(a + 1, a + n);
    printf("%d\n", a[2]);
}
```

9.2 Brute

```
// slow solution for finding second smallest element
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for(int& x : a) {
        cin >> x;
    }
    for(int x : a) {
        int count_smaller = 0;
        for(int y : a) {
            if(y < x) {
                ++count_smaller;
            }
        }
        if(count_smaller == 1) {
            cout << x;
            return 0;
```

```
        }
    }
    assert(false);
}
```

9.3 Gen

```
// generating a random sequence of distinct elements
#include <bits/stdc++.h>
using namespace std;

int rand(int a, int b) {
    return a + rand() % (b - a + 1);
}

int main(int argc, char* argv[]) {
    srand(atoi(argv[1])); // atoi(s) converts an array of chars to int
    int n = rand(2, 10);
    printf("%d\n", n);
    set<int> used;
    for(int i = 0; i < n; ++i) {
        int x;
        do {
            x = rand(1, 10);
        } while(used.count(x));
        printf("%d ", x);
        used.insert(x);
    }
    puts("");
}
```

9.4 Gen Tree

```
// generating a tree in a simple way
#include <bits/stdc++.h>
using namespace std;

int rand(int a, int b) {
    return a + rand() % (b - a + 1);
}

int main(int argc, char* argv[]) {
    srand(atoi(argv[1]));
    int n = rand(2, 20);
    printf("%d\n", n);
    for(int i = 2; i <= n; ++i) {
        printf("%d %d\n", rand(1, i - 1), i);
    }
}
```

9.5 Gen Tree2

```
// generating a tree in a not-so-stupid way
#include <bits/stdc++.h>
using namespace std;

int rand(int a, int b) {
    return a + rand() % (b - a + 1);
}

int main(int argc, char* argv[]) {
    srand(atoi(argv[1]));
    int n = rand(2, 20);
    printf("%d\n", n);
    vector<pair<int,int>> edges;
    for(int i = 2; i <= n; ++i) {
        edges.emplace_back(rand(1, i - 1), i);
    }

    vector<int> perm(n + 1); // re-naming vertices
    for(int i = 1; i <= n; ++i) {
        perm[i] = i;
    }
    random_shuffle(perm.begin() + 1, perm.end());

    random_shuffle(edges.begin(), edges.end()); // random order of edges

    for(pair<int, int> edge : edges) {
        int a = edge.first, b = edge.second;
        if(rand() % 2) {
            swap(a, b); // random order of two vertices
        }
        printf("%d %d\n", perm[a], perm[b]);
    }
}
```

9.6 In

```
4
10 30 20 40
```

9.7 Readme.Md

```
#### How to use the stress tester (Thanks Errichto)
- ‘a.cpp‘ - incorrect solution
- ‘brute.cpp‘ - correct slow solution
- ‘in‘ - sample input
- ‘gen.cpp‘ - test generator
- ‘s.sh‘ - script that tests a.cpp and brute.cpp against each other
- ‘gen_tree.cpp‘ - stupid tree generator
- ‘gen_tree2.cpp‘ - better tree generator

My compilation flags:

1. fast running time
    ‘g++ -O2 -std=c++17 -Wno-unused-result -Wshadow -Wall -o a a.cpp‘

2. check for mistakes
    ‘g++ -std=c++17 -Wshadow -Wall -o a a.cpp
        -fsanitize=address -fsanitize=undefined -D_GLIBCXX_DEBUG -g‘
```

9.8 Stress

```
for((i = 1; ; ++i)); do
    echo $i
    ./gen $i > int
    # ./a < int > out1
    # ./brute < int > out2
    # diff -w out1 out2 || break
    diff -w -y <(./a < int) <(./brute < int) || break
done
```

10 Tree

10.1 Binary Lifting

```
struct binary_lifting {
    vector<vi> jump;
    binary_lifting(vi par){
        int n = SZ(par);
        jump.resize(LOG, vi(n));
        jump[0] = par;

        forr(j, 1, LOG){
            forn(i, n) jump[j][i] = jump[j-1][jump[j-1][i]];
        }
    }

    int lift(int v, int k){
        for(int i = LOG-1; i >= 0; i--)
            if(k & (1<<i)) v = jump[i][v];
        return v;
    }
};
```

10.2 Centroid

```
struct centroid {
  vector<vector<int>> g; int n;
  vector<vector<int>> c_tree;
  bool tk[MAXN];
  int fat[MAXN]; // father in centroid decomposition
  int szt[MAXN]; // size of subtree
  int centro = -1;
  int calcsz(int x, int f){
    szt[x]=1;
    for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=calcsz(y,x);
    return szt[x];
  }
  void cdfs(int x=0, int f=-1, int sz=-1){ // O(nlogn)
    if(sz<0)sz=calcsz(x,-1);
    for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
      szt[x]=0;cdfs(y,f,sz);return;
    }
    tk[x]=true;fat[x]=f;
```

```
    for(auto y:g[x])if(!tk[y])cdfs(y,x);
  }
  centroid(vector <vector<int>> gg, int nn) {
    g = gg; n = nn; memset(tk,false,sizeof(tk));cdfs();
    c_tree.resize(n);
    forn(i,n) {
      if(fat[i] == -1) centro = i;
      else c_tree[fat[i]].pb(i);
    }
  }
};
```

10.3 Centroid Cses

```
vector<int> g[MAXN];int n;
bool tk[MAXN];
int fat[MAXN]; // father in centroid decomposition
int szt[MAXN]; // size of subtree
int calcsz(int x, int f){
    szt[x]=1;
    for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=calcsz(y,x);
    return szt[x];
}
void cdfs(int x=0, int f=-1, int sz=-1){ // O(nlogn)
    if(sz<0)sz=calcsz(x,-1);
    for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
        szt[x]=0;cdfs(y,f,sz);return;
    }
    tk[x]=true;fat[x]=f;
    for(auto y:g[x])if(!tk[y])cdfs(y,x);
}
void centroid(){memset(tk,false,sizeof(tk));cdfs();}
```

10.4 Diameter

```
vector < vector <int> > g(MAXN);
vector <bool> visto(MAXN,false);

void DFS(int v, int &new_start, int &ans, int l) {
    visto[v] = true;
    if(l > ans) ans = l, new_start = v;
    for(int u : g[v]) if(!visto[u]) DFS(u,new_start,ans,l+1);
}

int find_diameter() {
    int ans = 0, new_start;
    DFS(0,new_start,ans,0); fill(all(visto),false);
    DFS(new_start,new_start,ans,0);
    return ans;
}
```

10.5 Dsu On Tree

```
#define rs(n) resize(n)

struct DSU_on_Tree { // count #vertex with color c in subtree v
  vector<map<ll, ll>> h; // histogram [color,count]
  vi color, link, ans;
  vii sum;
  vector<vi> g;

  DSU_on_Tree(int n, vector<vi> _g, vi _c) : g(_g), color(_c) {
    ans.rs(n+5); link.rs(n+5); h.rs(n+5); sum.rs(n+5);
    forn(i,n+5) link[i] = i, sum[i] = {0,0};
    dfs(0);
  }

  int find(int x) {
    return link[x] = (link[x] == x ? x : find(link[x]));
  }

  void add(int v, int c, int cnt) { // add [color,cnt] to root v
    ll &cur = h[v][c];
    cur += cnt;
    if(sum[v].first == cur) sum[v].second += c;
    else if(sum[v].first < cur) sum[v] = {cur,c};
  }

  void merge(int i, int j){
    if(SZ(h[i]) < SZ(h[j])) swap(i,j);
    link[j] = i;
    for(auto x : h[j]){
      int c, cnt; tie(c,cnt) = x;
      add(i,c,cnt);
    }
    h[j].clear();
  }
```

```
  void dfs(ll v, ll p = -1){
    add(v,color[v],1);
    for(auto u : g[v]) {
      if(u == p) continue;
      dfs(u, v);
      merge(find(v), find(u));
    }
    ans[v] = sum[find(v)].second; // here solve for vertex v
  }
};
```

10.6 Dynamic Connectivity

```
struct UnionFind {
    int n,comp;
    vi uf,si,c;
    UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1){
        forn(i,n) uf[i]=(int)i;
    }
    int find(int x){return x==uf[x]?x:find(uf[x]);}
    bool join(int x, int y){
        if((x=find(x))==(y=find(y)))return false;
        if(si[x]<si[y])swap(x,y);
        si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
        return true;
    }
    int snap(){return SZ(c);}
    void rollback(int snap){
        while(SZ(c)>snap){
            int x=c.back();c.pop_back();
            si[uf[x]]-=si[x];uf[x]=x;comp++;
        }
    }
};
enum {ADD,DEL,QUERY};
struct Query {int type,x,y;};
struct DynCon {
    vector<Query> q;
    UnionFind dsu;
    vector<int> mt;
    map<pair<int,int>,int> last;
    DynCon(int n):dsu(n){}
    void add(int x, int y){
        if(x>y)swap(x,y);
        q.pb({ADD,x,y});mt.pb(-1);last[{x,y}]=SZ(q)-1;
    }
    void remove(int x, int y){
        if(x>y)swap(x,y);
        q.pb({DEL,x,y});
        int pr=last[{x,y}];mt[pr]=SZ(q)-1;mt.pb(pr);
    }
    void query(int x, int y){q.pb({QUERY,x,y});mt.pb(-1);} // modificar
        que query se tiene que usar
    void process(){ // answers all queries in order
        if(!SZ(q)) return;
        forn(i,SZ(q))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) // answer query using DSU q[s]
                assert(false); // poner alguna operacion entre q[s].x y
                    q[s].y
            return;
        }
        int k=dsu.snap(),m=(s+e)/2;
        for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s)dsu.join(q[i].x,q[i
            ].y);
        go(s,m);dsu.rollback(k);
        for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join(q[i].x,q[i].y);
        go(m,e);dsu.rollback(k);
    }
};
```

10.7 Find Centroid

```
vector < vector <int> > g(MAXN);
vector <bool> is_centroid(MAXN,true);
vector <int> sz(MAXN,0);

void DFS(int v, int prev, int n) {
    sz[v] = 1;
    for(int u : g[v]) {
        if(u == prev) continue;
        DFS(u,v,n);
```

```
        sz[v] += sz[u];
        if(sz[u] > n/2) is_centroid[v] = false;
    }
    if(n - sz[v] > n/2) is_centroid[v] = false;
}

vector <int> find_centroid(int n) {
    // Centroid: A node so that each subtree has at most floor(n/2)
        nodes
    vector <int> ans;
    DFS(1,-1,n);
    forr(i,1,n+1) if(is_centroid[i]) ans.pb(i);
    return ans;
}
```

10.8 Hld

```
struct hld {
  #define rs(x) resize(x)
  int n;
  vector <vector<int>> g;
  vector <tipo> values, sz, heavy, parent, depth, top, r, pos;
  segtree st; // iterative is faster

  hld(int _n, vector <vector<int>> _g, vector <tipo> _v) {
    n = _n; g = _g; values = _v;
    sz.rs(n); parent.rs(n); depth.rs(n); top.rs(n); pos.rs(n);
    heavy.rs(n);
    forn(i,n) heavy[i] = i;
    dfs(0);
    get_order(0,0);
    st.build(r,r.size());
  }

  ll dfs(int v, int p = -1, int d = 0) {
    depth[v] = d; parent[v] = p;
    for(int &u : g[v]) {
      if(u == p) continue;
      ll tam = dfs(u,v,d+1);
      // move heavy node to the begining
      if(sz[u] > sz[heavy[v]]) heavy[v] = u, swap(u,g[v][0]);
      sz[v] += tam;
    }
    sz[v] += 1;
    return sz[v];
  }

  void get_order(int v, int t, int p = -1) { //first child is heavy
    pos[v] = r.size(); top[v] = t; r.pb(values[v]);
    for(int u : g[v]) {
      if(u == p) continue;
      if(u == heavy[v]) get_order(u,t,v);
      else get_order(u,u,v);
    }
  }

  void op(ll &res, int a, int b) { // Set operation
    res = max(res, st.query(pos[a],pos[b]));
    // ------- for segtree recursive use ----------
    // res = max(res, st.query(pos[a],pos[b]).ans);
  }

  tipo query_hld(int a, int b) {
    tipo res = NEUT;
    for(; top[a] != top[b]; b = parent[top[b]]) {
      if(depth[top[a]] > depth[top[b]]) swap(a, b);
      op(res,top[b],b);
    }
    if(depth[a] > depth[b]) swap(a, b);
    op(res,a,b);
    return res;
  }

  void update_hld(int p, tipo val) { st.update(pos[p],val); }
};
```

10.9 Isomorphism Centroid

```
struct centroid {
    vector < vector <int> > g;
    vector <bool> is_centroid;
    vector <int> sz;

    centroid(vector <vector <int> > &graph, int tam) {
        g = graph;
        is_centroid.resize(tam,true);
```

```
        sz.resize(tam,0);
    }

    void DFS(int v, int prev, int n) {
        sz[v] = 1;
        for(int u : g[v]) {
            if(u == prev) continue;
            DFS(u,v,n);
            sz[v] += sz[u];
            if(sz[u] > n/2) is_centroid[v] = false;
        }
        if(n - sz[v] > n/2) is_centroid[v] = false;
    }

    vector <int> find_centroid(int n) {
        // Centroid: A node so that each subtree has at most floor(n/2)
            nodes
        vector <int> ans;
        DFS(0,-1,n);
        forn(i,n) if(is_centroid[i]) ans.pb(i);
        return ans;
    }
};

struct iso {
    vector <vector <int> > t1, t2;
    map <vector <int>,int> mapa;
    int idx = 0;

    iso(vector <vector <int> > &tree1, vector <vector <int> > &tree2) {
        t1 = tree1; t2 = tree2;
    }

    int dfs(int cur, int p, vector < vector<int> > &g) {
        vector <int> v;
        for(int u : g[cur]) {
            if(u != p) v.push_back(dfs(u,cur,g));
        }
        sort(all(v));
        if(!mapa.count(v)) mapa[v] = idx++;
        return mapa[v];
    }

    bool check_iso(vector <int> &centroid1, vector <int> &centroid2) {
        int s1 = dfs(centroid1[0],-1,t1);
        for(int u : centroid2) {
            int s2 = dfs(u,-1,t2);
            if(s1 == s2) return true;
        }
        return false;
    }
};
```

10.10 Lca

```
/*
Tiempo de build: O(nlogn)
Memoria: O(nlogn)
Tiempo por query: O(logn)
*/
const int MAXN = 200005, LOG = 20;

struct LCA {
    int n, root;
    vector<vector<int>> g;
    int jmp[MAXN][LOG], depth[MAXN]; // jmp[i][j] tiene el 2^j-esimo
        ancestro de i
    void lca_dfs(int x) {
        for(int u : g[x]) {
            if(u == jmp[x][0]) continue;
            jmp[u][0] = x; depth[u] = depth[x]+1;
            lca_dfs(u);
        }
    }

    LCA(int tam, vector<vector<int>> &tree, int r): n(tam),root(r),g(
        tree) {
        depth[root] = 0;
        memset(jmp,-1,sizeof(jmp)); jmp[root][0] = root;
        lca_dfs(root);
        forr(k, 1, LOG){ forn(i, n){
                if(jmp[i][k-1]<0) jmp[i][k] = -1;
                else jmp[i][k] = jmp[jmp[i][k-1]][k-1];
            }
        }
    }
```

```

    int lca(int x, int y){
        if(depth[x] < depth[y]) swap(x,y);
        for(int i = LOG-1; i >= 0; i--) {
            if(depth[x]-(1<<i) >= depth[y]) x = jmp[x][i];
        }
        if(x == y) return x;
        for(int i = LOG-1; i >= 0; i--) {
            if(jmp[x][i] != jmp[y][i]) x = jmp[x][i], y = jmp[y][i];
        }
        return jmp[x][0];
    }

    int dist(int x, int y) {
        return depth[x] + depth[y] - 2 * depth[lca(x,y)];
    }
};
```

10.11 Link Cut Tree

```
const int MAXN = 5e5+5;
int ls[MAXN],rs[MAXN],fa[MAXN],siz[MAXN],st[MAXN],ch[MAXN][2],siz2[MAXN];
bool rev[MAXN];
#define ls(x) ch[x][0]
#define rs(x) ch[x][1]
inline bool notrt(int x){return ls(fa[x])==x||rs(fa[x])==x;}
inline void pushdown(int x){
    if(rev[x]){
        if(ls(x)) ls(ls(x))^=rs(ls(x))^=ls(ls(x))^=rs(ls(x)),rev[ls(x)]^=1;
        if(rs(x)) ls(rs(x))^=rs(rs(x))^=ls(rs(x))^=rs(rs(x)),rev[rs(x)]^=1;
        rev[x]=0;
    }
}
inline void rotate(int x){
    int y=fa[x],z=fa[y];bool l=rs(y)==x,r=!l;
    if(notrt(y)) ch[z][rs(z)==y]=x;if(ch[x][r]) fa[ch[x][r]]=y;
    fa[y]=x,fa[x]=z,ch[y][l]=ch[x][r],ch[x][r]=y;
    siz[y]=siz[ls(y)]+siz[rs(y)]+1+siz2[y],siz[x]=siz[ls(x)]+siz[rs(x)]+1+siz2[x];
}
inline void splay(int x){
    int y=x,z=1;st[1]=y;
    while(notrt(y)) st[++z]=y=fa[y];
    while(z) pushdown(st[z--]);
    while(notrt(x)){
        y=fa[x],z=fa[y];
        if(notrt(y)) if(ls(z)==y^ls(y)==x) rotate(x);else rotate(y);
            rotate(x);
    }
}
inline void access(int x){int y=0;while(x) splay(x),siz2[x]+=siz[rs(x)]-siz[y],rs(x)=y,siz[x]=siz[ls(x)]+siz[rs(x)]+1+siz2[x],x=fa[y=x];}
inline void make(int x){access(x),splay(x),ls(x)^=rs(x)^=ls(x)^=rs(x),rev[x]^=1;}
inline void split(int x,int y){make(x),access(y),splay(y);}
inline void link(int x,int y){make(x),make(y),fa[x]=y,siz2[y]+=siz[x];}
inline void cut(int x,int y){split(x,y),fa[x]=ls(y)=0,siz[y]=siz[ls(y)]+siz[rs(y)]+1+siz2[y];}
```

10.12 Sm To Large

```
// Small to large techique.
struct query { // Queries to answer, v: vertex
    ll v, h, idx;
};

vi g[MAXN];
vector<query> q[MAXN]; // Queries to answer
vi ans(MAXN, -1); // Answer to each query

unordered_map<ll, ll> cnt[MAXN] // The structures to store the information and merge

// Merge operation, merge small to large and return the large one
int merge(int v, int u){
    if(SZ(cnt[v]) < SZ(cnt[u])) swap(u, v); // now v is the large one

    // Merge cnt[u] into cnt[v]
    for(auto [x, y]: cnt[u]){
        // Do something with x, y and cnt[v]
    }
    cnt[u].clear(); // Clear the small one to mantain memory in O(n)

```

```
    return v; // return the large node
}

// Process the queries of v, v_repr is the representative of v (large
    node after merging v and its children)
void process_queries(int v, int v_repr){
    for(auto &[_v, k, i]: q[v]) ans[i] = cnt_geq[v_repr][k];
}

string s;

int dfs(int v, int p){
    int v_repr = v // Initialize the representative of v

    // Initialize the storage structs of only v

    for(auto u: g[v]){
        if(u == p) continue;
        int u_repr = dfs(u, v); // Get the representative of u
        v_repr = merge(v_repr, u_repr); // Merge u_repr into v_repr
    }

    process_queries(v, v_repr); // Offline process the queries of v

    return v_repr;
}


void solve(){
    int n, m; cin >> n >> m;
    color.resize(n); forn(i, n) cin >> color[i];

    forn(i, n-1){
        int u, v; cin >> u >> v; u--, v--;
        g[u].pb(v); g[v].pb(u);
    }

    forn(i, m){
        int v, k; cin >> v >> k; v--;
        q[v].pb({v, k, i}); // Add the query to the list of queries of v
    }
    dfs(0, -1);

    forn(i, m) cout << ans[i] << "\n";
}

int main(){
    FIN;

    solve();

    return 0;
}
```