Cuadernillo

Sparkies

Octubre 2025

Contents

1 Data Structures

1.1 Disjoint Set Union Find (DSU)

```
/*
Autor: Oscar Vargas Pabon

Recordar que existe el 'reachibility tree'
        (dejar las uniones explicitas como nuevos nodos, sacrificando
            compresion de caminos)


TODO: En la superregional se hizo un 'augmenting' para que acumulara en sets.


Lo probe en https://codeforces.com/problemset/problem/1857/G
*/

int dsu_pi[template_limit], dsu_sz[template_limit];

void build( int n ) {
/* Construye el estado inicial en tiempo O(n) */
        for ( int i = 0 ; i < n ; ++i ) dsu_pi[i]=i, dsu_sz[i]=1;
}

int getRepr( int x ) {
/* Halla el representante de x */
        if ( dsu_pi[x] != x ) dsu_pi[x] = getRepr( dsu_pi[x] );
        return dsu_pi[x];
}

void mergeSet( int x, int y ) {
/* Une los sets de 'x' y 'y' */
        x = getRepr( x ); y = getRepr( y );
        if ( x != y ) {
                if ( dsu_sz[x] < dsu_sz[y] ) swap( x, y );
                dsu_pi[y] = x;
                dsu_sz[x] += dsu_sz[y];
        }
}
```

1.2 Segment Tree

1.2.1 Lazy Recursive

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC

tested in https://codeforces.com/problemset/problem/2117/H

l -> left tree bound, r -> right tree bound
ql -> left op bound, qr -> right op bound
*/
class Lazy{
public:
        int vl;
        Lazy(int v=0):vl(v){};

        void combine( const Lazy &l ){
                // unir con otro lazy (l estaba en el padre)
                vl+=l.vl;
        }
        pair<Lazy,Lazy> separe( int l, int r ) {
                // separar en izquierda y derecha
                return {*this,*this};
        }
};
class Data {
        public:
        int sum;
        Data(int s=0) : sum(s) {};
        Data operator + ( const Data &o ) const {
                return Data(sum+o.sum);
        }
        void update( const Lazy &o, int l, int r ){
                // modificar el dato
                sum+=(r-l+1)*o.vl;
        }
};

class SegT{
public:
        vector<Data> tree;
        vector<Lazy> tag;
        int t_size;
        void calc_cons( int ind, int l, int r, int &m, int &rind ) {
                m = (l+r)/2; rind = ind+2*(m-l+1);
        }
void raw_build ( const vector<Data> &arr,int ind=0,int l=0,int r=-1 ) {
        if(r==-1)r=t_size;
    /* funcion recursiva auxiliar que construye el segTree */

        // tag[ind] = Lazy(); // siempre en neutro
        if ( l == r ) tree[ind] = arr[l]; // caso base
        else {
                int m,rind; calc_cons(ind,l,r,m,rind);
                raw_build( arr, ind+1, l, m ); // hijo izquierdo
                raw_build( arr, rind, m+1, r ); // hijo derecho

                tree[ind] = tree[ind+1] + tree[rind];
        }
}
void push( int ind, int l, int r ) {
/* Empuja el vlor del tag a los nodos mas profundos */
        tree[ind].update( tag[ind],l,r );
        if ( l < r ) { // no hay edge-case cuando lt==rt
                int m, rind; calc_cons(ind,l,r,m,rind);
                pair<Lazy,Lazy> sd=tag[ind].separe(l,r);
                tag[ind+1].combine(sd.first);
                tag[rind].combine(sd.second);
        }
        tag[ind] = Lazy(); // para no sobrecontar
}
        SegT()=default;
        SegT( const vector<Data>&arr ){
                t_size=arr.size();
                tag.resize(t_size*2); tree.resize(t_size*2);
                --t_size;
                raw_build( arr );
        }


void update( int ql, int qr, Lazy vl, int ind=0, int l=0, int r=-1 ) {
        if(r==-1)r=t_size;
/* actualizo los vlores en arr[lx],...,arr[rx] por arr[lx]+=vl,...,arr[rx]+=vl
        en el arbol en tiempo O(lg n) */
        push( ind, l, r );
        if ( ql <= l && r <= qr ) {
                tag[ind].combine(vl);
                push( ind, l, r );
        } else if ( !(r<ql||qr<l) ) {
                int m,rind; calc_cons(ind,l,r,m,rind);

                update( ql, qr, vl, ind+1, l, m ); // hijo izquierdo
                update( ql, qr, vl, rind, m+1, r ); // hijo derecho
                tree[ind] = tree[ind+1] + tree[rind]; // actualizo el arbol
        }
}
```

```
Data query( int ql, int qr, int ind=0, int l=0, int r=-1 ) {
        if(r==-1)r=t_size;
/* Hago una query de arr[lx]+...+arr[rx] en tiempo O(lg n) */
        Data res;
        push( ind, l, r );
        if ( ql <= l && r <= qr ) res = tree[ind];
        else if ( !(r<ql||qr<l) ) {
                int m,rind; calc_cons(ind,l,r,m,rind);
                res = query(ql,qr,ind+1,l,m)+query(ql,qr,rind,m+1,r);
        }
        return res;
}
};
```

1.2.2 Iterative

```
/*
Autor : Oscar Vargas Pabon
Material de referencia para ICPC
Lo probe en 12299 - RMQ with Shifts
*/
int tree[template_limit], arr_size;

void build( int *arr, int n ) {
        /* Construye el arbol en tiempo O(n) */
        arr_size = n;
        for ( int i = 0 ; i < arr_size ; ++i ) tree[i+n] = arr[i];
        for ( int i = arr_size-1 ; i > 0 ; --i ) tree[i] = min( tree[i*2],
            tree[i*2+1] );
}
void update( int x, int val ) {
        /* Modifica el valor en arr[x] segun la representacion del arbol en O(
            lg n) */
        x+=arr_size;
        tree[x]=val;
        for ( x >>= 1 ; x > 0 ; x >>= 1 ) tree[x] = min( tree[x*2], tree[x
            *2+1] );
}

int query( int l, int r ) {
        /* Responde a la query arr[l] + ... + arr[r] en tiempo O(lg n) */
        int res = 1e9 ;
        for ( l += arr_size, r += arr_size ; l <= r ; l >>= 1, r >>= 1 ) {
                if ( l&1 ) res = min( res, tree[l++] );
                if ( !(r&1) ) res = min( res, tree[r--] );
        }
        return res;
}
```

1.2.3 Persistent

```
/*
Autor: Oscar Vargas Pabon

No olvidar el guardar las nuevas versiones en rts[rt_n++]
Esta version no tiene LAZY PROPAGATION

Probado en https://www.spoj.com/problems/DQUERY/
           https://codeforces.com/contest/2111/problem/G
*/

class Data {
        public:
        int sum;
        Data(int s=0) : sum(s) {};
        Data operator + ( const Data &o ) const {
                return Data(sum+o.sum);
        }
};

class Node {
        public:
        Node *l,*r;
        Data dat;
        Node(){ l=r=NULL; }
};

int t_sz, rt_n;
Node *rts[template_limit]; // roots;
vector<Node*> to_erase; // guarda todo lo que se debe borrar;
Node* create_node(){Node *nd=new Node();to_erase.push_back(nd);return nd;}
Node* build( const vector<Data> &arr, int l=0,int r=-1 ) {
        if ( r==-1 ) t_sz=r=arr.size()-1;
        Node *nd = create_node();
        if ( l>=r ) nd->dat = arr[l];
        else {
                int m = (l+r)/2;
                nd->l = build(arr,l,m);
                nd->r = build(arr,m+1,r);
                nd->dat = nd->l->dat + nd->r->dat;
        }
        return nd;
}
Node *update( int x, const Data &vl, Node *nd, int l=0, int r=t_sz ){
        Node *neo = create_node();
        if ( l >= r ) neo->dat= nd->dat+vl;
        else {
                int m =(l+r)/2;
                if ( x <= m ) neo->l=update(x,vl,nd->l,l,m),neo->r=nd->r;
                else neo->r=update(x,vl,nd->r,m+1,r),neo->l=nd->l;
                neo->dat = neo->l->dat + neo->r->dat;
        }
        return neo;
}

Node *bulk_update(const vector<pair<int,Data>>&vl,Node*nd ){
        // I assume elements of the form vl[i]={ind,x} where im supposed to
        // do Arr[vl[i].first]=vl[i].second (or whatever update im using
        int ind=0; //I assume vl is sorted by .first (index)
        function<Node*(Node*,int,int)> upd=[&](Node*nd, int l,int r){
                if ( ind==int(vl.size())|| r<vl[ind].first||vl[ind].first<l )
                    return nd;
                // debug(l,r,vl[ind].first,ind);
                Node *neo = create_node(); neo->dat=nd->dat;
                if ( l >= r ) while(ind<int(vl.size())&&vl[ind].first==l) neo
                    ->dat= neo->dat+vl[ind++].second;
                else {
                        int m =(l+r)/2;
                        neo->l=upd(nd->l,l,m); neo->r=upd(nd->r,m+1,r);
                        neo->dat = neo->l->dat + neo->r->dat;
                }
                return neo;
        }; return upd(nd,0,t_sz);
}

Data query( int ql, int qr, Node *nd, int l=0, int r=t_sz ) {
        Data res;
        if ( ql<=l && r <= qr ) res = nd->dat;
        else if ( !(r<ql||qr<l) ) {
                int m = (l+r)/2;
                res = query(ql,qr,nd->l,l,m)+query(ql,qr,nd->r,m+1,r);
        }
        return res;
}

void free_pseg(){
        for ( Node *nd : to_erase) delete nd;
        rt_n = 0; to_erase={};
}
```

```
/* EndOf persistent segtree */
```

1.3 Fenwick Tree

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC
Lo probe en testing\rangeQtest.cpp
*/

int bit[template_limit], bit_size; // BIT -> Binary Indexed Tree

void build( int *arr, int n ) {
	/* Crea el arbol en tiempo O(n) sobre el arreglo 'arr' */
	int nxt; bit_size = n+1; // porque esta indexado en 1
	memset( bit, 0, sizeof(int)*bit_size );
	for ( int i = 0 ; i < bit_size ; ++i ) {
		bit[i] += arr[i]; nxt = i + (i&(-i));
		if ( nxt < bit_size ) bit[nxt] += bit[i];
	}
}
int query( int x ) {
	/* Responde la suma del prefijo de bit[1]+...+bit[x] en tiempo O(lg n)
	   */
	int res = 0;
	for ( ; x > 0 ; x -= x&(-x) ) res += bit[x];
	return res;
}
int query( int l, int r ) {
	/* responde la suma del rango de bit[l]+...+bit[r] en tiempo O(lg n)
	   */
	return query( r ) - query( l-1 );
}
int query2(int l,int r){
	// la misma query de arriba pero funciona con trucos que me saque
	// del servidor de disc GF comentario de Aeren en tips-and-tricks
	int res=0;
	for(;l<r;r-=r&(-r))res+=bit[r];
	for(;r<l;l-=l&(-l))res-=bit[l];
	return res;
}
void update( int x, int val ) {
	/* Modifica el valor de arr[x] en la representacion arborea en O(lg n)
	   */
	for ( ; x < bit_size ; x += x&(-x) ) bit[x] += val;
}

// computa el logaritmo entero de num(la cantidad de bits activos) - esta fun
    es aparte
//int ilog2( int num ) { return sizeof(int)*8 - __builtin_clz( num ) -1; }
// asumo el anterior de mi template
int binlift( int val ) {
	/* Retorna el menor indice 'x' tal que query( x ) >= val en tiempo O(
	   lg n) */
	int x = 0, nxt, sm = 0;
	for ( int exp = ilog2(bit_size) ; exp >= 0 ; --exp ) {
		nxt = x|(1<<exp);
		if ( nxt < bit_size && bit[nxt] < val ) {
			val -= bit[nxt];
			x = nxt;
		}
	}
	return x+1;
}
```

1.4 Treap

```
/*
Autor: Oscar Vargas Pabon
Notar que este treap construye un max-heap en el atributo 'key'.
No tiene anadido ningun tipo de 'lazy propagation'
Las operaciones asumen indexacion en 0

Testeado en https://www.spoj.com/problems/GSS6/en/
*/

// mt19937_64 random_64( chrono::steady_clock::now().time_since_epoch().count
    () );
// asumo el anterior elemento de mi template

class Data{
	public:
	int pref,suf,sum,mx;
	Data(){
		pref=suf=mx=-1e9;
		sum = 0;
	}
	Data( int vl ) { pref=suf=sum=mx= vl; }
	Data operator +( const Data &o ) const {
		Data res; // NO necesariamente es asociativo
		res.sum = sum+o.sum;
		res.pref=max(pref,sum+o.pref);
		res.suf=max(o.suf,o.sum+suf);
		res.mx=max({mx,o.mx,suf+o.pref});
		return res;
	}
};

class Node{
	public:
	// mv->MyValue; sv->SubtreeValue;
	Data mv,sv;
	int key, sz;
	Node *l,*r; // hijos
	Node( const Data &dt=Data() ){
		key = random_64()%(1ll<<30); sz = 1;
		mv=sv=dt;
		l=r=NULL;
	}
	void update(){
		auto subt_dt=[&]( Node *act ){
			return (act)?act->sv:Data();
		};
		sv = subt_dt(l) + mv + subt_dt(r) ;
		auto subt_sz=[&]( Node *act ){
			return (act)?act->sz:0;
		};
		sz = subt_sz(l) + 1 + subt_sz(r);
	}
};
void join( Node *&t, Node *l, Node *r){
	if ( !l || !r ) t=(l)?l:r;
	else if ( l->key < r->key ) join(r->l,l,r->l),t=r;
	else join(l->r,l->r,r),t=l;
	t->update();
}
void split( Node* t, int x, Node *&l, Node *&r ) {
	// l=t[0..x); r=t[x..n)
	int lsz=(t)?((t->l)?t->l->sz+1:1):0;
	if (!t) l=r=NULL;
	else if ( lsz <= x ) split(t->r,x-lsz,t->r,r),l=t;
	else split(t->l,x,l,t->l),r=t;
	if(t)t->update();
}

void t_in(Node *&t, int pos, Data vl){
	// a[0..pos)+vl+a[pos..n)
	Node *l,*r;
	split(t,pos,l,r);
```

```
        Node *neo= new Node(vl);
        join(t,l,neo);
        join(t,t,r);
}

void t_upd(Node *&t, int pos, Data vl){
        Node *l,*m,*r; split(t,pos,l,m); split(m,1,m,r);
        m->mv=vl;
        join(t,l,m); join(t,t,r);
}

void t_out(Node *&t,int pos){
        // a[0..pos)+a(pos..n)
        Node *l,*m,*r;
        split(t,pos,l,m);
        split(m,1,m,r);
        if(m) delete m;
        join(t,l,r);
}

Data t_query(Node *&t,int ql,int qr){
        // a[ql..qr)
        Node *l,*m,*r;
        split(t,qr,m,r);
        split(m,ql,l,m);

        Data res=m->sv;
        join(t,m,r); join(t,l,t);
        return res;
}

Node * t_build(const vector<Data> &arr){
        vector<Node*> ms;//MonotonicStack
        for(const Data &act:arr){
                Node *nd=new Node(act),*prv=NULL;
                while(!ms.empty()&&ms.back()->key<=nd->key){
                        prv=ms.back(); ms.pob();
                        prv->update();
                }
                if(!ms.empty())ms.back()->r=nd;
                nd->l=prv;
                ms.pb(nd);
                nd->update();
        }
        Node *prv=NULL;
        while(!ms.empty()){
                ms.back()->r=prv;
                prv=ms.back(); ms.pob();
                prv->update();
        }
        return prv;
}

void t_debug(Node*t){
        function<void(Node*)> aux=[&](Node*nd){
                if(!nd)return;
                aux(nd->l);

                cout << "(-" << nd->key << "-";
                cout << nd->sv.sum << '-';

                aux(nd->r);
        }; aux(t); cout << endl;
}
```

1.5 Mo Algo

```
/*
Autor: Oscar Vargas Pabon
Recordar que existe la idea de balanceo (usar sqrt-decomp o otra de modo que
        la update se haga en O(1), aunque la query pueda empeorar, pues el
            algoritmo hace
        mas updates que queries)


Asumo que T(REM) y T(ADD) son los tiempos de remover y anadir un indice
        del global que maneja MO.

Idea 1: usar bloque de tamano \sqrt(n). Genera tiempo O((n+q)\sqrt(n))
Idea 2: usar bloque de tamano \frac{n}{\sqrt(q)}. Genera tiempo O(n\sqrt(q))
        source: https://codeforces.com/blog/entry/61203?#comment-451304
Idea 3: Usar orden de hilbert. Genera tiempo O(n\sqrt(q))
        source: https://codeforces.com/blog/entry/61203
        impl:https://codeforces.com/blog/entry/61203?#comment-1064868

Nota: La idea 1 (o tal vez 2 tambien) puede no funcionar muy bien cuando se
    requiere hacer un balanceo
Probado con https://www.spoj.com/problems/DQUERY/
Otro problema interesante https://codeforces.com/problemset/problem/2006/D
*/
uint64_t hilbertorder(uint64_t x, uint64_t y) {
        // https://codeforces.com/blog/entry/61203?#comment-1064868
    const uint64_t logn = __lg(max(x, y) * 2 + 1) | 1;
    const uint64_t maxn = (1ull << logn) - 1;
    uint64_t res = 0;
    for (uint64_t s = 1ull << (logn - 1); s; s >>= 1) {
        bool rx = x & s, ry = y & s;
        res = (res << 2) | (rx ? ry ? 2 : 1 : ry ? 3 : 0);
        if (!rx) {
            if (ry) x ^= maxn, y ^= maxn;
            swap(x, y);
        }
    }
    return res;
}

class Query{
public:
    int ind, l, r, ord;
    Query()=default;
    bool operator < ( const Query &o ) const{ return ord<o.ord; }
};

const int method=0;
vector<int> mo_algo( int n, vector<Query> &query ) {
        int q = query.size();

        // idea 1 -> O(n\sqrt(q))
        if(method==0)for(Query &q:query)q.ord=hilbertorder(q.l,q.r);
        else{
          int block_size;
          if(method==1){
            // idea 2 -> O(n\sqrt(q))
            int sqrt=1; while(sqrt*sqrt<q)++sqrt;
            block_size=n/sqrt;
          } else {
            // idea 3 -> O((n+q)\sqrt(n))
            block_size=1;
            while ( block_size*block_size < n ) ++block_size;
          }
          block_size=max(block_size,1);
          for(Query &q:query){
            int bl_ind=q.l/block_size;
            q.ord=(bl_ind&1)?n-q.r:q.r;
            q.ord+=bl_ind*n;
          }
        }
```

```
        sort(query.begin(),query.end());

        /// Llenar esta parte con lo necesario del problema
        vector<int> res(q);
        vector<int> all(n,0);
        int cnt=0;
        function<void(int)> add = [&]( int x ){
                if ( !all[a[x]] ) ++cnt;
                ++all[a[x]];
        };
        function<void(int)> rem = [&]( int x ) {
                --all[a[x]];
                if ( !all[a[x]] ) --cnt;
        };
        //// termina la parte del llenado

        int l = 0, r = -1;
    for ( const Query &act : query ){
        while ( r > act.r ) rem(r--);
        while ( r < act.r ) add(++r);
        while ( l < act.l ) rem(l++);
                while ( l > act.l ) add(--l);

                res[act.ind] = cnt; // esto depende del ejercicio
        }
        return res;
}
```

1.6 tipsNtricks

Remember doing binary lifting (sparse table) is posible (precalculating all jumps)

Remember doing coordinate compression is possible

2 Trees

2.1 Centroid

```
/*
Autor: Oscar Vargas Pabon
Implementacion de referencia para ICPC

Nomas es una referencia sobre hallar un centroide.
        * Recordar como se puede hacer una 'centroid decomposition' que
            permite
                hallar propiedades sobre los caminos que pasan por cada
                    centroide asegurando
                O(n lg n )

Probado en CSES 2079
*/

int size[template_limit];

int findSize( const vector<list<int>> &tree, int node=0, int parent=-1 ) {
        /* halla el tamano de cada subarbol */
        size[node] = 1;
        for ( const int edge : tree[node] ) {
                if ( edge != parent ) size[node] += findSize( tree, edge, node
                    );
        }
        return size[node];
}

int findCentroid( const vector<list<int>> &tree, int node=0 ) {
        /* halla el centroide */
        int centroid = node;
        for ( const int edge : tree[node] ) {
                if ( size[edge] > size[node]/2 ) {
                        // reroot to the other edge
                        size[node] -= size[edge]; size[edge] += size[node];
                        centroid = findCentroid( tree, edge );
                        break;
                }
        }
        return centroid;
}
```

2.2 Heavy Light Decomposition (HLD)

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC

Probado en CSES 2134 -- 1138

Esta implementacion asume una Range-DS con funciones
        * void build( int *arr, int n ) * int query(int l, int r )
*/

int tin[template_limit], tout[template_limit], sz[template_limit]; // data del
    arbol
int jmp[template_limit], chain[template_limit]; // data de los saltos de HLD
int perm[template_limit]; // data para la Range-DS

bool isParent( int u, int v ) {
        /* Responde si 'u' es padre de 'v' en el arbol */
        bool res = tin[u] < tin[v] && tout[u] >= tout[v];
        return res;
}

void getSz( vector<list<int>> &tree, int node=0, int parent=-1 ) {
        /* Hallo el tamano, pongo las Heavy-edges de primero */
        sz[node] = 1;
        int heavy = node;
        for ( const int it : tree[node] ) {
                if ( it != parent ) {
                        getSz( tree, it, node );
                        if ( sz[heavy] < sz[it] ) heavy = it; // Heavier edge
                        sz[node] += sz[it]; // calculo el tamano
                }
        }
        if ( heavy != node ) { // reorganizo para que las Heavy-edges queden
            de primero
                tree[node].remove( heavy );
                tree[node].push_front( heavy );
        }
}

void getLabeling( const vector<list<int>> &tree, int node=0, int parent=-1,
    int time=0 ) {
        /* Hallo los tin,tout, y los jmp,chain */
        tin[node] = time++; // hallo el tiempo de entrada
        bool first = true;
        for ( const int it : tree[node] ) {
                if ( it != parent ) {
                        if ( first ) { // esta es una Heavy-Edge
                                jmp[it] = jmp[node];
                                chain[it] = chain[node];
                                first = false;
                        } else { // las demas
                                jmp[it] = node;
                                chain[it] = it;
                        }

                        getLabeling( tree, it, node, time );
                        time = tout[it]; // actualizo el tiempo
                }
        }
        tout[node] = time; // encuentro el tout
}
```

```
void build_hld( vector<list<int>> &tree, const vector<int> &value ){
        /* Construye el hld a partir de un arbol y sus valores asociados en
            tiempo
                O(n+B(n)) donde B indica el tiempo de la Range-DS usada en
                    inicializar */
        const int root = 0; // arbitrario
        getSz( tree, root ); // hallo las Heavy-Edges
        jmp[root] = root;
        getLabeling( tree, root ); // hallo tin,tout,jmp y chain

        // construyo la Range-DS
    for ( int i = 0 ; i < n ; ++i ) perm[tin[i]] = value[i];
    build( perm, n );
}

int queryPath( int u, int v ) {
        /* Responde a la query arr[u] + .. + arr[v] donde todos los elementos
            estan en el
                camino de 'u' a 'v' en el arbol. En tiempo O(lg n *T(n) )
                    donde T(n) es el
                tiempo de la DS usada */
        int res = 0;
        while ( chain[u] != chain[v] ) {
                if ( isParent( chain[u], v ) ) swap( u, v ); // el que llegue
                    primero hace swap
                res += query( tin[chain[u]], tin[u] );
                u = jmp[u];
        }

        if ( isParent( u, v ) ) swap( u, v ); // para incluir el chain hasta
            el LCA
        res += query( tin[v], tin[u] );
        return res;
}
```

2.3 Lowest Common Ancestor (LCA)

Recordar que la version por Binary Lifting implica computar $\pi : V \times \mathbb{N} \to V$ donde $\pi(v, k)$ significa hacer 2^k saltos sucesivos a los ancestros, iniciando desde v.

Recordar que dado $A \subseteq V$, entonces $C = \{lca(a, b) | a, b \in A\}$ se puede lograr ordenando los vertices en A por tiempos de entrada del DFS y hallando $LCA(a_i, a_{i+1})$.

2.3.1 Euler Tour version

Recordar que esta version ayuda a 'aplanar' el arbol para trabajar queries sobre subarboles.

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC
Lo probe en mi maquina
Esta implementacion asume una RMQ-DS con funciones
        * void build( int *arr, int n ) * int query(int l, int r )
*/

int euler[template_limit], tin[template_limit], itin[template_limit];

int dfs_time( const vector<list<int>> &tree, int node, int father=-1, int time
    =0 ) {
        tin[node] = time; itin[time] = node; // los renombramientos
        euler[time] = time; // el orden en la RMQ-DS
        ++time;

        for ( const int it : tree[node] ) {
                if ( it != father ) {
                        time = dfs_time( tree, it, node, time );
                        euler[time] = tin[node]; // para que aparezca entre
                            todos sus sub-arboles
                        ++time;
                }
        }
        return time;
}
void build_lca( const vector<list<int>> &tree, int root=0 ) {
/* construyo la LCA en tiempo O(n+B(n)) donde B es el tiempo de construir de
    la RMQ-DS */
        int tmp = dfs_time( tree, root ); // calculo los tiempos y -euler
            circuit-
        build( euler, tmp ); // construyo la RMQ-DS
}
int lca( int u, int v ) {
/* respondo el LCA en tiempo O(T(n)) donde T es el tiempo de query de la RMQ-
    DS */
        u = tin[u]; v = tin[v];
        if ( u > v ) swap( u, v );
        return itin[query(u,v)];
}
```

2.3.2 Binary Lifting

Recordar que esta version ayuda a agregar valores sobre caminos en el arbol (version estatica de lo que se hace en HLD).

```
/*
Autor: Oscar Vargas Pabon

Esto funciona en O(nlgn) de memoria y preprocesamiento, O(lgn) de query

Notar que estoy asumiendo que 0 es la raiz (de lo contrario se puede danar
        la acumulacion de bl "bl[nd][i+1]" en build_lca)

Fue testeado en: https://codeforces.com/problemset/problem/1843/F2
*/
class Data{
        public:
        int sm,mxpr,mnpr,mxsf,mnsf,mxsq,mnsq;
        Data(int xx=0){
                sm=xx;
                mxpr=mxsf=mxsq=max(0,xx);
                mnpr=mnsf=mnsq=min(0,xx);
        }
        Data operator +(const Data &o)const{
                Data neo;
                neo.sm=sm+o.sm;
                neo.mxpr=max(mxpr,sm+o.mxpr);
                neo.mnpr=min(mnpr,sm+o.mnpr);

                neo.mxsf=max(o.mxsf,o.sm+mxsf);
                neo.mnsf=min(o.mnsf,o.sm+mnsf);

                neo.mxsq=max({mxsq,o.mxsq,mxsf+o.mxpr});
                neo.mnsq=min({mnsq,o.mnsq,mnsf+o.mnpr});
                return neo;
        }
        void invert(){ swap(mnpr,mnsf); swap(mxpr,mxsf); }
};

const int lgi=20;
Data bl[template_limit][lgi];
int pi[template_limit][lgi], dpt[template_limit];

void build_lca(const vector<vector<int>> &t, const vector<Data> &arr, int nd
    =0,int p=0, int d=0 ){
        dpt[nd]=d;
        pi[nd][0]=p;
        bl[nd][0]=arr[nd];

        rep(i,0,lgi-1){
                int anc = pi[nd][i];
                pi[nd][i+1]=pi[anc][i];
```

```
                if(anc)bl[nd][i+1]=bl[nd][i] + bl[anc][i];
        }

        for(int e:t[nd]) if(e!=p) build_lca(t,arr,e,nd,d+1);
}
Data query(int u,int v){
        auto jump=[&](Data &dt, int &x,int j){ dt=dt+bl[x][j]; x=pi[x][j]; };
        if(dpt[u]<dpt[v])swap(u,v);
        Data l,r;
        rep(i,lgi-1,-1)if( (dpt[u]-dpt[v])&(1<<i) ) jump(l,u,i);
        if(u==v)return l+bl[u][0];

        rep(i,lgi-1,-1)if( pi[u][i]!=pi[v][i] ){
                jump(l,u,i); jump(r,v,i);
        }
        jump(l,u,0); jump(r,v,0);
        l=l+bl[u][0]; r.invert();
        return l+r;
}
```

3 Graph

3.1 random shit

3.1.1 shortest paths

Remember 0-1 BFS exists. Remember $\delta(u,v) \leq \delta(u,k)+\delta(k,v)$ for any k and that there exists k that makes it equality (unless $(u,v) \in E$). Remember that A^* exists and it requires the over-estimation function to fulfill $\delta(u,v) \geq \delta(u,k)+OE(k,v)$ where OE is the over-estimation. Remember that dijkstra (as other methods) can span an arborescence or a dag (of used edges).

3.1.2 bridges

Remember the algorithm of bridges we have studied mantains the time of arrival of each node and the minimum time it looked on its path, in a way its

$$mini(v) = min\begin{cases} time(v) & \\ mini(c) & \text{Where } c \text{ is } v\text{'s child in the DFS tree} \\ time(e) & \text{Where } (v,e) \text{ represents a back-edge} \end{cases}$$

For each child in the DFS tree there is an articulation point in v iff $mini(c) \geq time(v)$; now (v,c) is a bridge iff $mini(c) > time(v)$.

3.1.3 floyd-warshal

We have the dp function $dst(k+1,u,v) = min(dst(k,u,v), dst(k,u,k)+dst(k,k,v))$. Remember it can be optimized to $O(n^2)$ memory by the relaxation step being safe. For only connectivity it can be further optimized by bitset my love.

3.2 Dinitz

Note that it usually works in $O(V^2E)$. In unit networks (they all have capacity of 1 except for source and sink) it works in $O(E\sqrt{v})$

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC
Lo probe en https://codeforces.com/problemset/problem/2026/E
Notar las variables globales:
    * vector<list<int>> graph;
        * vector<Edge> edges;
        * int source, sink;
        *
        * int level[template_limit];
        * bool blocked[template_limit];
        * list<int>::iterator ptr[template_limit];
Mi implementacion interpreta los valores de 'graph' como indices de 'edges'
*/

class Edge {
public:
        int u, v, c1, c2, f;// c1 es la capacidad u->v y c2 es para v->u
        Edge()=default;
        Edge( int u, int v, int c1, int c2=0 ) : u(u),v(v),c1(c1),c2(c2),f(0)
            {};
        int to( int node ) const { // el opuesto a 'node'
                return ( u == node ) ? v : u;
        }
        void push( int node, int pushed ) { // empujar un flujo 'pushed' desde
            'node'
                if ( node == u ) c1 -= pushed, c2 += pushed, f += pushed;
                else c1 += pushed, c2 -= pushed, f -= pushed;
        }
        int cap( int node ) const { // la capacidad desde 'node'
                return ( this->u == node ) ? c1 : c2;
        }
        int flow( int node ) const { // el flujo desde 'node'
                return ( u == node ) ? f : -f;
        }
};

int level[template_limit]; // para el 'layered network'
bool blocked[template_limit]; // para omitir vertices 'blocked'
list<int>::iterator ptr[template_limit]; // para omitir aristas 'blocked'

vector<list<int>> graph;
vector<Edge> edges;
int source, sink;

void addEdge( int u, int v, int c1, int c2=0 ) {
        // u->v has capacity c1; v->u has capacity c2

        graph[u].push_back( edges.size() );
        graph[v].push_back( edges.size() );
        edges.push_back( Edge( u, v, c1, c2 ) );
}

bool level_bfs( ) {
/* construye el 'layered network' de manera implicita con 'level' en O(V+E) */
        memset( level, -1, sizeof(int)*int(graph.size()) );
        level[source] = 0; // 'level' es la profundidad el el BFS-Tree

        queue<int> q; q.push( source );
        while ( !q.empty() && level[sink] == -1 ) {
                int act = q.front(); q.pop();
                for ( int edge : graph[act] ) {
                        int nxt = edges[edge].to(act); // el siguiente vertice
                        if ( level[nxt] == -1 && edges[edge].cap(act)>0 ) {
                                level[nxt] = level[act] + 1;
                                q.push( nxt );
```

```
                }
            }
        }
        return level[sink]!=-1; // para saber si ya terminamos
}

int push_dfs( int node, int flow=1e9 ) {
/* Empuja el flujo por todas las aristas del 'layered graph' que pueda */
        if ( node == sink || flow==0 ) return flow; // ya llegamos o no
            podemos empujar mas

        int push_flow = 0, edge_flow, edge, nxt;
        while ( ptr[node] != graph[node].end() && flow > push_flow ) {
                edge_flow = 0; edge = *ptr[node]; nxt = edges[edge].to(node);
                // si la arista pertenece al 'layered graph' y el vertice NO
                    esta bloqueado
                if ( level[node] < level[nxt] && !blocked[nxt] )  {
                        edge_flow = push_dfs( nxt, min(edges[edge].cap(node),
                            flow-push_flow) );
                        push_flow += edge_flow;
                        edges[edge].push( node, edge_flow );
                }
                ++ptr[node];
        }
        --ptr[node]; // para corregir el ultimo ++ptr[node]
        blocked[node] = (push_flow == 0); // verifica si se bloqueo el vertice
        return push_flow;
}

int maxFlow( ) {
/* Hace el maximo flujo del grafo (retorna el maxFlow, las asignaciones quedan
     en las aristas)
        Funciona en peor caso O(v^2*E) */
        int flow = 0;
        while ( level_bfs( ) ) {
                memset( blocked, 0, sizeof(bool)*int(graph.size()) ); //
                    reinicializo esto
                for ( int i = 0 ; i < int(graph.size()) ; ++i ) ptr[i] = graph
                    [i].begin();

                flow += push_dfs( source );
        }
        return flow;
}
```

3.3 Blossoms

```
/*
Sacado del comentario de bicsi encontrable en
https://codeforces.com/blog/entry/92339?#comment-810166

Retorna un vector de vertices que tiene -1 si no ha sido considerado en el
    matching,
        de lo contrario el vertice con el que ha sido 'matcheado'.
Toma tiempo O(V*E).
Probado en <11439 Maximizing the ICPC>.
*/
vector<int> Blossom(vector<list<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> 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.push_back(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(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(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.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

3.4 Hungarian

Solves the problem of the max-matching of min-cost given edge-weights on the graph.

```
/*
 * Hungarian Algorithm
 * ----------------------------------------
 *  Entrada:
 *   - Una matriz cuadrada cost_matrix de tamano nXn,
 *     donde cost_matrix[i][j] representa el costo de asignar
 *     el trabajador i al trabajo j.
 *
 *  Salida:
 *   - El costo minimo total de asignacion optima.
 *
 *  Uso:
 *   - Resuelve el "Assignment Problem": asignar n tareas a n agentes
 *     minimizando el costo total (o maximizando utilidad si se invierte el
 *    signo).
 *   - Complejidad: O(n^3)
 */

#include <bits/stdc++.h>
using namespace std;

class Hungarian {
private:
    vector<vector<int>> cost_matrix; // Matriz de costos
    int n;                           // Tamano de la matriz
    vector<int> u, v, p, way;        // Vectores auxiliares para el algoritmo
```

```
public:
    // Constructor: inicializa con la matriz de costos
    Hungarian(const vector<vector<int>> &matrix) : cost_matrix(matrix) {
        n = matrix.size();
        u.resize(n + 1, 0);
        v.resize(n + 1, 0);
        p.resize(n + 1, 0);
        way.resize(n + 1, 0);
    }

    // Funcion principal: devuelve el costo minimo total
    int compute() {
        for (int i = 1; i <= n; ++i) {
            p[0] = i;
            vector<int> minv(n + 1, INT_MAX);
            vector<bool> used(n + 1, false);
            int j0 = 0;
            while (true) {
                used[j0] = true;
                int i0 = p[j0], delta = INT_MAX, j1 = 0;
                for (int j = 1; j <= n; ++j) {
                    if (!used[j]) {
                        int cur = cost_matrix[i0 - 1][j - 1] - u[i0] - v[j];
                        if (cur < minv[j]) {
                            minv[j] = cur;
                            way[j] = j0;
                        }
                        if (minv[j] < delta) {
                            delta = minv[j];
                            j1 = j;
                        }
                    }
                }
                for (int j = 0; j <= n; ++j) {
                    if (used[j]) {
                        u[p[j]] += delta;
                        v[j] -= delta;
                    } else {
                        minv[j] -= delta;
                    }
                }
                j0 = j1;
                if (p[j0] == 0)
                    break;
            }
            while (true) {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
                if (j0 == 0)
                    break;
            }
        }

        int total_cost = 0;
        for (int j = 1; j <= n; ++j)
            total_cost += cost_matrix[p[j] - 1][j - 1];
        return total_cost;
    }
};
```

3.5 Euler Circuit

Nota: Mi impl puede no funcionar bien para no dirigidos. Upd:nomas use un set para ver cuando no puede visitar algo y ya (para el undirected case)

```
/*
Autor: Oscar Vargas Pabon

Esta implementacion destruye el grafo y es solo para dirigidos.

Recordar que solo hay un circuito euleriano si indeg[nd]==outdeg[nd] para
        todos los nodos.
Un camino euleriano se cumple cuando para u,v indeg[u]+1==outdeg[u],
        indeg[v]==outdeg[v]+1 y todos los demas nodos cumplen indeg[nd]==
            outdeg[nd].
        En este caso basta con iniciar la primera solucion de 'euler_circuit'
            en u.
*/
void euc_aux( vector<list<pair<int,int>>> &g, int nd, list<pair<int,int>> &res
     ) {
        if ( !g[nd].empty() ) {
                int neo_nd,val;
                neo_nd = g[nd].front().first; val = g[nd].front().second;

                res.pb( pair<int,int>(nd,val) );
                g[nd].ppf();

                euc_aux( g, neo_nd, res );

        }
}

list<pair<int,int>> euler_circuit( vector<list<pair<int,int>>> &g ) {
        list<pair<int,int>> res;
        euc_aux( g, 0, res );
        for ( list<pair<int,int>>::iterator it = res.begin() ; it != res.end()
             ; ++it ){
                if ( !g[it->first].empty()  ) {
                        list<pair<int,int>> tmp;
                        euc_aux( g, it->first , tmp );

                        for ( const pair<int,int> &act : tmp ) res.insert(it,
                            act);
                        while ( !tmp.empty() ) {
                                tmp.ppf(); --it;
                        }
                }
        }
        return res;
}
```

3.6 2 SAT

Nota: mi impl es una mierda. Tomadlo de guia nada mas

```
/*
Autor: Oscar Vargas Pabon

Usado en https://codeforces.com/contest/2120/problem/F
y en https://codeforces.com/gym/105053/problem/E

Esta version no esta testeada por completo (es mas para dar la idea)
Mi representacion de las clausulas es una mierda
*/

vector<int> topo(const vector<vector<int>> &g){
        int n=g.size();
        vector<int> res; vector<bool> vis(n,0);
        function<void(int)> taux=[&](int nd){
                vis[nd]=1;
                for(int e:g[nd])if(!vis[e])taux(e);
                res.push_back(nd);
        };
        rep(i,0,n)if(!vis[i])taux(i);
        return res;
}
bool check_sat(const vector<pair<pair<int,bool>,pair<int,bool>> &claus, const
    vector<bool> &ass){
        bool res=0;
        for( pair<pair<int,bool>,pair<int,bool>> ac:claus){
```

```
                res=res&&((ass[ac.first.first]^(!ac.first.second))||(ass[ac.
                    second.first]^(!ac.second.second)));
        }
        return res;
}
vector<bool> sat2(const vector<pair<pair<int,bool>,pair<int,bool>>> &claus,int
     n){
        vector<vector<int>> g(n);
        for(pair<pair<int,bool>,pair<int,bool>> ac:claus)rep(i,0,2){
                g[ac.first.first*2+ !ac.first.second].push_back(ac.second.
                    first*2 + ac.second.second);
        }
        vector<int> tord=topo(g);
        vector<int> vl(n,-1);
        rep(i,n*2-1,-1) if ( vl[tord[i]/2]==-1 ) vl[tord[i]/2]=tord[i]&1;
        vector<bool> res(n);rep(i,0,n)res[i]=vl[i];
        return (check_sat(claus,res))?res:{};
}
```

3.7 Kosaraju for strongly connected components

Nota: no lo he testeado.

```
/*
Autor: Oscar Vargas Pabon

Esta untested. Tengo la gran sospecha de que no es necesario el grafo
        transpuesto, pero no tengo ganas de revisarlo/probarlo
*/
vector<int> topo(const vector<vector<int>> &g){
        int n=g.size();
        vector<int> res; vector<bool> vis(n,0);
        function<void(int)> taux=[&](int nd){
                vis[nd]=1;
                for(int e:g[nd])if(!vis[e])taux(e);
                res.push_back(nd);
        };
        rep(i,0,n)if(!vis[i])taux(i);
        return res;
}
void kosaraju(const vector<vector<int>> &g){
        int n=g.size();

        vector<vector<int>> ig(n);// Transpose graph
        for(int i=0;i<n;++i)for(int e:g[i])ig[e].push_back(i);

        vector<int> tord=topo(g);
        vector<int> gr(n,-1);int gind=0;// finding group
        function<void(int)> dfs=[&](int nd){
                gr[nd]=gind;for(int e:g[nd])if(gr[e]==-1)dfs(e);
        };
        for(int nd:tord)if(gr[nd]==-1){
                dfs(nd); ++gind;
        }
        vector<vector<int>> cg(gind);//condensed graph
        for(int i=0;i<n;++i)for(int e:g[nd]){
                cg[gr[i]].push_back(gr[e]);
        }
}
```

3.8 clique

This shit shouldnt exists. But here we are

```
/*
Autor: Oscar Vargas Pabon

Backtracking version tested in https://open.kattis.com/problems/maxclique
        (it seems to do well with V<=50)
Diamond free version tested in https://codeforces.com/gym/105505/problem/D
Add version for chordal graphs
*/

// Note it can be transformed to give the maximal clique by
// keeping the used vertices instead of 'am'
int max_clique(const vector<lint> &g){
        // I'm assuming no self-edges
        int res=0,n=g.size();
        auto ilog2ll=[]( lint num ) { return 8*sizeof(lint) - __builtin_clzll(
            num ) - 1; };
        function<void(int,lint)> backt=[&](int am,lint msk){
                if(am+__builtin_popcountll(msk)<=res)return;

                if( msk==0ll )res=max(res,am);
                else {
                        int i=ilog2ll(msk&(-msk));
                        backt(am+1,msk&g[i]);
                        backt(am,msk&(~(1ll<<i)));
                }
        };backt(0,(1ll<<n)-1);
        return res;
}

// For a diamond free graph, each edge belongs to,
// at most, 1 maximal clique. O(n^2)
vector<vector<int>> chordal_clique(const vector<vector<bool>> &g){
        // I assume adj-matrix representation of g
        int n=g.size();vector<vector<int>> res;
        vector<vector<bool>> vis(n,vector<bool>(n,0));
        rep(i,0,n)rep(j,0,n)if(!vis[i][j] && g[i][j] ){
                vector<int> act={i,j}; //now I search the group
                rep(k,0,n)if(g[i][k]&&g[j][k])act.pb(k);
                for(int u:act)for(int v:act)vis[u][v]=1;
                res.pb(act);
        }
        return res;
}
```

4 Strings

4.1 Z function

Computa $Z : 1..N \to 1..N$ donde $Z(i)$ implica que $str[j] = str[i+j]$ para todo $j \leq Z(i)$ y $str[Z(i)+1] \neq str[i+Z(i)+1]$.

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC
Lo probe en mi carpeta de pruebas
*/

vector<int> zFunction( const string &cad ) {
/* Computa la funcion z en O(n).
Esta es z[i] -> maximo prefijo comun de cad y cad[i...] */
        int n = cad.size();
        vector<int> z( n, 0 );
        int l = -1, r = -1;
        for ( int i = 1 ; i < n ; ++i ) {
                z[i] = max( 0, min( r-i, z[i-l] ) );
                while ( i+z[i] < n && cad[z[i]] == cad[i+z[i]] ) ++z[i];
                if ( i+z[i] > r ) r=i+z[i],l=i;
        }
        return z;
}
```

4.2 Prefix function

Computa $\pi : 1..N \to 1..N$ donde $\pi(i)$ implica que $str[1..\pi(i)] = str[i-\pi(i)..i]$ y este es maximal.

```
/*
```

```
Autor: Oscar Vargas Pabon
Material de referencia para ICPC
Probado en mis biblioteca
*/

vector<int> prefixFunction( const string &cad ) {
/* Computa la prefix function en O(n).
   Esta es pi[i] -> el tamano del mayor prefijo
        de cad que tambien es sujifo de cad[0..i] */
        int n = cad.size();
        vector<int> pi( n, 0 );
        for ( int i = 1 ; i < n ; ++i ) {
                pi[i] = pi[i-1];
                while ( pi[i] > 0 && cad[i] != cad[pi[i]] ) pi[i] = pi[pi[i
                    ]-1];
                if ( cad[i] == cad[pi[i]] ) ++pi[i];
        }
        return pi;
}
```

4.3 Suffix Array

Computa $A : 1..N \to 1..N$ donde $i < j$ implica $str[A(i)..N] < str[A(j)..N]$. También puede computar $LCP : i..N \to 1..N$ donde $LCP(i)$ identifica el tamaño del prefijo comun más grande entre $str[A(i-1)..N]$ y $str[A(i)..N]$.

```
/*
Autor: Oscar Vargas Pabon
Material de referencia para ICPC
Probado en Codeforces -> ITMO -> SuffixArray -> step4 -> A
*/

const char EOS = '$';
// end-of-string -> se supone que es un caracter
//             estrictamente menor a todos los demas del string

vector<int> suffixArray( string &str ) {
/* Halla el suffix-array (SA) en tiempo O(nlg n) */
    str.pb( EOS ); // ahorra edge-cases

    int i, n = str.size();
    vector<int> code( n ), master( n ), newCode( n );
    {
            // para las 'equivalence classes' cuando solo contienen un
                caracter
        vector<pair<char,int>> previousMaster( n );
        for ( i = 0 ; i < n ; ++i ) previousMaster[i] = pair<char,int> ( str[i
            ], i );
        sort( previousMaster.begin(), previousMaster.end() );

        master[previousMaster[0].second] = 0; // ahorra edge-cases
        for ( i = 1 ; i < n ; ++i ) {
            if ( previousMaster[i-1].first < previousMaster[i].first )//
                distinto al anterior
                code[previousMaster[i].second] = code[previousMaster[i-1].
                    second]+1;
            else // igual al anterior, mantiene el codigo
                code[previousMaster[i].second] = code[previousMaster[i-1].
                    second];
        }
            // actualiza el master (SA computado hasta ahora)
        for ( i = 0 ; i < n ; ++i ) master[i] = previousMaster[i].second;
    }

    int k = 1;
    while ( k < n && code[master.back()] < n-1 ) {
            // hace el 'cyclic-shift' del master
        for ( i = 0 ; i < n ; ++i ) master[i] = (master[i]-k+n)%n;

        {
            vector<int> copy = master; // hago el bucket-sort en O(n)
            vector<int> bucket_size( n+1, 0 );
            for ( i = 0 ; i < n ; ++i ) ++bucket_size[code[copy[i]]+1];//
                cuento los elementos de cada 'bucket'
            for ( i = 1 ; i <= n ; ++i ) bucket_size[i] += bucket_size[i-1];//
                hago el arreglo de prefijos
            for ( i = 0 ; i < n ; ++i ) {
                master[bucket_size[code[copy[i]]]++] = copy[i];//lleno master
                    otra vez
            }
        }

        newCode[master[0]] = 0; // creo las nuevas 'equivalence clases'
        for ( i = 1 ; i < n ; ++i ) {
            newCode[master[i]] = newCode[master[i-1]];
            if ( code[master[i-1]] != code[master[i]] || code[(master[i-1]+k)%
                n] != code[(master[i]+k)%n] ) {
                ++newCode[master[i]]; // la tupla de codigos es distinta
            }
        }
        code = newCode;
        k = ( k << 1 );
    }

    str.ppb(); // quito el EOS
    return master;
}

vector<int> lcpArray( vector<int> &sufix, string &str ) {
/* halla el longest-common-prefix array del sufixArray en tiempo O(n)
        Retorna lcp[i] -> lcp de sufix[i] y sufix[i+1] */
    str.push_back( EOS ); // me ahorra un edge-case
        int n = sufix.size();

    vector<int> inv ( n ); // para obtener la poscicion de i en el SA
    for ( int i = 0 ; i < n ; ++i ) inv[sufix[i]] = i;

    vector<int> lcp ( n-1 );
    int k = 0, j;
    for ( int i = 0 ; i < n-1 ; ++i ) {
        j = sufix[inv[i]-1]; // inv[i]-1 no se sale porque sufix[0] representa
            EOS
        while ( str[i + k] == str[j + k] ) ++k;
        lcp[inv[i]-1] = k;
        k = max( k-1, 0 );
    }
    str.pop_back(); // quito EOS
    return lcp;
}
```

4.4 Aho-Corasick automata

```
/*
Autor: Oscar Vargas Pabon
Lo probe en UVA 1449

La implementacion asume reinicializar 'states' - states=vector<Node>(1);
        * Notar que pattern no considera el vertice actual como un posible
            pattern
Remember to use 'next()' and 'pattern()' to do the queries because of the lazy
    stuff.
*/

const int K = 26; const char NORM='a';

class Node {
        public:
        int next[K];
        int output, parent, link, pattern;
        char letter;
```

```
        Node ( int p=0, char l=0, int o=-1 ) : parent(p), letter(l),output(o)
            {
                memset( next, -1, sizeof(next) );
                link=pattern=-1;
        }
};

vector<Node> states(1);

void add_string( const string & str, int strInd=0 ) {
        /* anade la cadena 'str' al trie, la marca con 'strInd' */
        /*O(|str|)*/
        int act = 0;
        for ( const char letter : str ) {
                char ch = letter-NORM;
                if ( states[act].next[ch] == -1 ) {
                        states[act].next[ch] = states.size();
                        states.push_back( Node( act, letter ) );
                }
                act = states[act].next[ch];
        }
        states[act].output = max( states[act].output, strInd );
}

int link( int act ) ;

int next( int act, char letter ) {
        /* hallo el siguiente estado a visitar segun el actual y el caracter
            */
        char ch = letter-NORM;
        if ( states[act].next[ch] == -1 ) {
                if ( act ) {
                        states[act].next[ch] = next( link(act), letter );
                } else states[act].next[ch] = 0;
        }
        return states[act].next[ch];
}

int link( int act ) {
        /* Halla el indice al nodo que representa el 'longest proper suffix'
            en el trie */
        if ( states[act].link == -1 ) {
                if ( act && states[act].parent ) {
                        states[act].link = next(link(states[act].parent),
                            states[act].letter);
                } else states[act].link = 0;
        }
        return states[act].link;
}

int pattern( int act ) {
        /* Halla el siguiente elemento que pertenece a un patron */
        if ( states[act].pattern == -1 ) {
                int nxt = link( act );

                if ( !nxt ) states[act].pattern = 0;
                else if ( states[nxt].output != -1 ) states[act].pattern = nxt
                    ;
                else states[act].pattern = pattern( nxt );
        }
        return states[act].pattern;
}

int main(){
        // Nota: estoy asumiendo que todos los patrones son distintos

        // ejemplo de hallar las ocurrencias de patrones en un texto
        int pt; cin >> pt; // cargo los patrones
        rep(i,0,pt){string cd;cin >> cd; add_string(cd,i);}
        string t; cin >> t;
        vector<int> ocur(pt,0);//cuenta las ocurrencias
        int act=0;// indica el estado del automata en el que estamos
        rep(i,0,t.size()) {
                act = next(act,t[i]);//salto al nuevo estado

                // esto lo hago en tiempo proporcional a los patrones
                int jmp=(states[jmp].output==-1)?pattern(act):act;
                while(jmp){++ocur[states[jmp].output];jmp=pattern(jmp);}
        }

        return 0;
}
```

4.5 Manacher

Note: its currently untested

```
/*
Autor: Oscar Vargas Pabon
Tomado de https://cp-algorithms.com/string/manacher.html
No ha sido testeado

Calcula p[i]=x donde [i-x,i+x] es un palindromo
Notar que no lo calculamos sobre la cadena, sino sobre
        c'='&'+a[0]+'$'+...+a[i]+...+'$'+a[n]+'&'
Si c'[i]='$' me refiero a un palindromo par (de lo contrario un impar)
*/

vector<int> manacher( const string &cad ){
        string c2="&$";
        for(char c:cad){c2.push_back(c);c2.push_back('$');}
        c2.push_back('%');
        int n=c2.size();
        vector<int> res(n-1,-1);
        int l=0,r=1;
        for(int i = 1 ; i < n-1 ; ++i ) {
                res[i]=min(r-i,res[l+(r-i)]);
                while(c2[i-res[i]]==c2[i+res[i]])++res[i];
                if(i+res[i]>r)l=i-res[i],r=i+res[i];
        }
        return res;
}
```

4.6 Hash

```
/*
Autor: Oscar vargas Pabon

Nota: en vez de precomputar ebase podriSa usar mpow para hacerlo en O(log n)

NO OLVIDAR INVOCAR 'precompute_prime' ANTES DE USAR

Testeado en; https://codeforces.com/problemset/problem/2132/G
*/

//defined in header <random> :- mersenne-twister
// mt19937_64 rng_64( chrono::steady_clock::now().time_since_epoch().count() )
    ;
// #define rep(i,strt,end) for(int i = strt ; i !=int(end) ; (int(strt)<int(
    end))?++i:--i )
// lint mpow(lint x,lint e,lint m){lint res=1ll;while(e){if(e&1ll)res=(res*x)%
    m;e>>=1;x=(x*x)%m;}return res;}
// asumo el anterior de mi template

const int N_HASH=2, PRIME_TESTING=3;
const lint LH_B=1e9,RH_B=1ll<<31;
lint base[N_HASH],modul[N_HASH],ebase[template_limit][N_HASH];

void precompute_prime(){
  function<lint(lint,lint)> rgen = [&](lint lb, lint rb ) {
```

```
    return lb+(rng_64())%(rb-lb+1ll);
  };
  for ( int i = 0 ; i < N_HASH ; ++i ) base[i]=rgen(2,LH_B);
  lint posi; bool prim;
  for ( int i = 0 ; i < N_HASH ; ++i ) {
    do {
      posi = rgen(LH_B,RH_B);
      prim = mpow(2,posi-1,posi)==1ll;
      for (int j = 0 ; j < N_HASH && prim ; ++j )
        prim=mpow(base[i],posi-1,posi)==1ll;
      for (int j = 0 ; j < PRIME_TESTING && prim ; ++j )
        prim=mpow(rgen(2,LH_B),posi-1,posi)==1ll;
    } while ( !prim ) ;
    modul[i] = posi;
  }
  // precomputar las potencias de la base _ tambien podria hacerlo en O(log n)
      con mpow
  rep(j,0,N_HASH) ebase[0][j]=1;
  rep(i,0,template_limit-1) rep(j,0,N_HASH) ebase[i+1][j]=(ebase[i][j]*base[j
      ])%modul[j];
}

class Hash{
public:
  array<lint,N_HASH> h;
  int len = 0;
  Hash() :len(0),h({0,0}) {};
  Hash(const string &cad) {
    len = cad.size();
    for ( int i = 0 ; i < N_HASH ; ++i ) {
      h[i]=0ll;
      for ( char c : cad ) h[i] = ((h[i]*base[i])%modul[i] + lint(c) )%modul[i
          ];
    }
  }
  Hash( char c ) {
        len=1;
        rep(i,0,N_HASH)h[i]= lint(c)%modul[i];
  }
  bool operator == ( const Hash &o ) const { return len==o.len&&h==o.h; }
  Hash operator + ( const Hash &o ) const {
    // *this + o
    Hash res;
    for ( int i = 0 ; i < N_HASH ; ++i ) {
      res.h[i] = ( (ebase[o.len][i]*h[i])%modul[i] + o.h[i] )%modul[i];
    }
    res.len = len+o.len;
    return res;
  }
  void lconc( char c ) {
        // hago this = c + this
        rep(i,0,N_HASH) h[i] = ( ebase[len][i]*lint(c) + h[i] ) % modul[i];
        ++len;
  }
  void rconc( char c ) {
          // hago this=this+c
        rep(i,0,N_HASH) h[i] = ( h[i]*base[i] + lint(c))%modul[i];
        ++len;
  }
  void ldeconc ( const Hash &l ) {
    // quito el prefijo l de *this
        // cout << l.len << " _ " << endl;
    for ( int i = 0 ; i < N_HASH ; ++i ) {
      h[i] -=(l.h[i]*ebase[len-l.len][i] )%modul[i];
          h[i] = ( h[i] + modul[i] ) %modul[i];
    }
    len -= l.len;
  }
  void rdeconc( const Hash &r ) {
    // quito el sufijo r de *this
    for ( int i = 0 ; i < N_HASH ; ++i ) {
      h[i] = ( (h[i]-r.h[i]) * mpow(ebase[r.len][i], modul[i]-2,modul[i]) ) %
          modul[i];
          h[i] = ( h[i] + modul[i] ) %modul[i];
    }
    len -=r.len;
  }
};
```

4.7 Booth algorithm for minimalShift

Returns the index that represents that cad[t:]+ cad[:t] is minimal

```
### returns the shift giving
### lexicographically minimal string
### among trans. phi(c_1S) |-> Sc_1
def booth(s: str) -> int:
    s = s + s
    n = len(s) // 2
    i, j, k = 0, 1, 0

    while i < n and j < n and k < n:
        if s[i + k] == s[j + k]:
            k += 1
        elif s[i + k] > s[j + k]:
            i = i + k + 1
            if i <= j:
                i = j + 1
            k = 0
        else:
            j = j + k + 1
            if j <= i:
                j = i + 1
            k = 0
    return min(i, j)
```

5 Algebra and friends

5.1 Convolutions

Calculamos $C(k) = \Sigma_{i+j=k} A(i) \cdot B(j)$. Recordar que con $B'(i) = B(N-i)$ podemos tambien calcular $C(k) = \Sigma_{i-j=k} A(i) \cdot B(j)$.

5.1.1 Fourier Fast Transform (FFT)

```
/*
Plagiado epicamente de https://cp-algorithms.com/algebra/fft.html
*/
using cd = complex<double>;
const double PI = acos(-1);

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

    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) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
```

```
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}
```

5.1.2 Numeric Theoretic Transform (NTT)

```
/*
Plagiado epicamente de https://cp-algorithms.com/algebra/fft.html


Recordar que puedo hacer $\sum_{i-j=k}a_ib_j$ utilizando j'=n-j.
        Tendre entonces que buscar en la poscicion k'=k+n.


const int mod = 924844033;
const int root = 44009197;
const int root_1 = mpow(root,mod-2,mod);
const int root_pw = 1 << 21;

const int mod = 998244353;
const int root = mpow(3,119,mod);
const int root_1 = mpow(root,mod-2,mod);
const int root_pw = 1 << 23;

*/
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1 << 20;

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

    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 = inverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
```

5.1.3 Operations on FormalPowerSeries

```
/*
Author: Oscar Vargas Pabon

Based on code by MarcosK and other people
        https://codeforces.com/contest/438/submission/340901913
and multiple blogs all around the place
        https://cp-algorithms.com/algebra/polynomial.html#inverse-series_1
        https://codeforces.com/blog/entry/56422
        https://codeforces.com/blog/entry/12513 - problem E

Tested in
        https://judge.yosupo.jp/problem/inv_of_formal_power_series
        https://judge.yosupo.jp/problem/exp_of_formal_power_series
        https://judge.yosupo.jp/problem/log_of_formal_power_series
        https://judge.yosupo.jp/problem/pow_of_formal_power_series
        https://judge.yosupo.jp/problem/sqrt_of_formal_power_series

The first version of this impl is in atcoder fps_24 A

I assume from my template:
everything     :: #define rep(i,strt,end) for(int i = strt ; i !=int(end) ; (
    int(strt)<int(end))?++i:--i )
everything     :: #define sz(vec) int(vec.size())
p_trunc        :: #define pob pop_back
p_mult,p_square:: int ilog2( int num ) { return 8*sizeof(int) - __builtin_clz(
     num ) - 1; }
modInverse     :: int mpow(int x,int e,int m){int res=1;while(e){if(e&1)res=(
    res*1ll*x)%m;e>>=1;x=(x*1ll*x)%m;}return res;}
TonelliShanks  :: mt19937_64 rng_64( chrono::steady_clock::now().
    time_since_epoch().count() );
*/

typedef int tfps;
typedef vector<tfps> fps;

/* START OF NTT */
const int mod = 998244353;
const int root = mpow(3,119,mod);
const int root_1 = mpow(root,mod-2,mod);
const int root_pw = 1 << 23;

int inverse(int vl,int md){return mpow(vl,md-2,md);}

void fft(fps & a, bool invert) {
    int n = a.size();

    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 = inverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
/* End of NTT */

void p_trunc(fps &F, int n,bool elim_0=1){
        F.resize(max(1,min(sz(F),n)));
        if(elim_0)while(sz(F)>1&&F.back()==0)F.pob();
}

void p_mult(fps &A, fps B){
        // A'=A*B in O(nlgn)
        int nm=A.size()+B.size();
        int lgi=ilog2(nm-1)+1;A.resize(1<<lgi);B.resize(1<<lgi);
        fft(A,0);fft(B,0);
        rep(i,0,sz(A))A[i]=(A[i]*1ll*B[i])%mod;
        fft(A,1);
        p_trunc(A,nm);
}
void p_square(fps &A){
        // A'=A*A in O(nlgn)
        int nm=A.size()*2;
        int lgi=ilog2(nm-1)+1;A.resize(1<<lgi);
        fft(A,0);
        for(int &ac:A)ac=(ac*1ll*ac)%mod;
        fft(A,1);
        p_trunc(A,nm);
}

void p_add(fps &F, int vl, int xi=0){
        // F+ vl*x^{xi}
        if(sz(F)<=xi)F.resize(xi+1);
        F[xi]+=vl;
}
void p_scale(fps &F,int vl){
        // F*vl
        rep(i,0,sz(F))F[i]=(F[i]*1ll*vl)%mod;
}
void p_add(fps &A, const fps &B, int sgn=1 ){
        // A + B*sgn ; I assume sgn\in\{-1,1\}
        A.resize(max(sz(A),sz(B)));
        rep(i,0,min(sz(A),sz(B)))A[i]=(A[i]+B[i]*sgn)%mod;
}
fps p_deriv(const fps &F){
        fps G(max(1,sz(F)-1));// G=D(F)
        rep(i,1,sz(F))G[i-1]=(F[i]*1ll*i)%mod;
        return G;
}
fps p_inte(const fps &F){
        fps G(sz(F)+1);//D(G)=F
        rep(i,0,sz(F))G[i+1]=(F[i]*1ll*mpow(i+1,mod-2,mod))%mod;
        return G;
}

fps p_inv(const fps &F,int n){
        assert(F[0]);// G*F=1
        fps G={mpow(F[0],mod-2,mod)};
        for(int e=2;e<2*n;e<<=1){
                fps ac=G;// gives a ~/2 speedup
                p_mult(ac,{F.begin(),F.begin()+min(sz(F),e)});
                rep(i,0,sz(ac))ac[i]=(mod-ac[i])%mod;

                p_add(ac,2); p_mult(G,ac);
                p_trunc(G,e,0);
        }
        p_trunc(G,n);
        return G;
}
fps p_log(const fps &F,int n){
        //first n terms of G=ln(F)=inte(D(F)/F)
        assert(F[0]==1);
        fps G=p_inv(F,n),dF=p_deriv(F);
        p_mult(G,dF); p_trunc(G,n-1);
        return p_inte(G);
}

fps p_exp(const fps &F,int n){
        assert(!F[0]);// first n terms of G=exp(F)
        fps G={1},ac;
        for(int e=2;e<n*2;e<<=1){
                ac=p_log(G,e); ac.resize(max(sz(ac),min(e,sz(F))));
                rep(i,0,sz(ac))ac[i]=(
                        (i<sz(F)?F[i]:0)- ac[i])%mod;
                // for some reason working with negatives
                // destroys something (NTT???)
                rep(i,0,sz(ac))if(ac[i]<0)ac[i]+=mod;
                p_add(ac,1);

                p_mult(G,ac); p_trunc(G,e,0);
        }
        p_trunc(G,n);
        return G;
}
fps p_pow(const fps &F,lint e,int n){
        // first n terms of G=F^e O(nlgn)
        if(!e)return {1};
        int xi=0;while(xi<sz(F)&&F[xi]==0)++xi;
        if( xi>=sz(F) || xi > n/e )return {0};

        int alp=mpow(F[xi],e%(mod-1),mod),ainv=mpow(F[xi],mod-2,mod);
        fps H(F.begin()+xi,F.end());p_scale(H,ainv);
        p_trunc(H,n-xi);

        int rx=xi*e;
        H=p_log(H,n-rx);
        p_scale(H,e%mod);
        H=p_exp(H,n-rx);

        fps G(n,0); H.resize(n-rx);
        rep(i,rx,n)G[i]=(H[i-rx]*1ll*alp)%mod;
        return G;
}
fps p_binpow( fps F, lint e, int n){
        // first n terms of G=F^e O(nlgnlge)
        fps G={1};while(e){
                if(e&1ll)p_mult(G,F);
                e>>=1;p_square(F);
                p_trunc(G,n);p_trunc(F,n);
        }
        return G;
}

int Tonelli_Shanks(int a, int mod) {
        //plagiado epicamente de https://judge.yosupo.jp/submission/270105

        if (a < 2) return a;
        if (mpow(a, (mod - 1) / 2, mod) != 1) return -1;
        if (mod % 4 == 3) return mpow(a, (mod + 1) / 4, mod);
```

```
    int b = 3;
    if (mod != 998244353) {
        while (mpow(b, (mod - 1) / 2, mod) == 1) {
            b=rng_64()%(mod-3) + 2;
        }
    }

    int q = mod - 1,Q = 0;
    while ( !(q&1)) Q++, q /= 2;

    int x = mpow(a, (q + 1) / 2, mod);
    b = mpow(b, q, mod);

    int shift = 2;
    while ((x *1ll* x) % mod != a) {
        int error = (((mpow(a, mod - 2, mod) *1ll* x) % mod) *1ll* x)
            % mod;
        if (mpow(error, 1 << (Q - shift), mod) != 1) {
            x = (x *1ll* b) % mod;
        }
        b = (b *1ll* b) % mod;
        ++shift;
    }
    return x;
}


fps p_sqrt( const fps &F,int n){
    // first n terms of G^2=F O(nlgn)
    const int i2 = mpow(2,mod-2,mod);

    int xi=0;while(xi<sz(F)&&F[xi]==0)++xi;
    if( xi>=sz(F) )return {0};

    fps G={Tonelli_Shanks(F[xi],mod)},ac;//sqrt{F[0]}
    if(G[0]==-1||(xi&1))return {};
    //I assume Tonelli_Shanks returns -1 when it's impossible
    // debug(xi,G.front());
    fps H(F.begin()+xi,F.end()); p_trunc(H,n-xi);



    for(int e=2;e<(n-xi)*2;e<<=1){
        ac=p_inv(G,e);
        p_mult(ac,H);p_trunc(ac,e,0);
        p_add(G,ac);p_scale(G,i2);
        p_trunc(G,e);
    }
    swap(H,G);
    G.resize(sz(G)+xi/2);
    rep(i,0,xi/2)G[i]=0;
    rep(i,xi/2,sz(G))G[i]=H[i-xi/2];
    return G;
}
```

5.1.4 Fast Walsh Hadamard Transform (FWH)

This allows to do $C(k) = \sum_{k=i^j} A_i B_j$ and the bitwise convolutions

```
/*
Shamelessly taken from https://github.com/mochow13/competitive-programming-
    library/blob/master/Math/Fast%20Walsh-Hadamard%20Transform.cpp
and adapted by yours truly, osvarp

currently UNTESTED
*/

template <typename T>
struct FWT {
    void fwt( vector<T> & io, bool sd ) {
        for (int d = 1; d < n; d <<= 1) {
            for (int i = 0, m = d<<1; i < n; i += m) {
                for (int j = 0; j < d; j++) { /// Don't forget
                    modulo if required
                    T x = io[i+j], y = io[i+j+d];

                    io[i+j] = (x+y), io[i+j+d] = (x-y);
                        // xor
                    if(!sd)io[i+j]/=2,io[i+j+d]/=2;///
                        Modular inverse if required here

                    // io[i+j] = (sd)?x+y:x-y;    // and
                    // io[i+j+d] = (sd)?x+y:y-x; // or
                }
            }
        }
    }
    // a, b are two polynomials and n is size which is power of two
    void convolution(vector<T> &A, vector<T> &B) {
        fwt(A,1); fwt(B,1);
        for (int i = 0; i < n; i++)
            a[i] = a[i]*b[i];/// Don't forget modulo if required
        fwt(A,0);
    }
    // for a*a
    void self_convolution(vector<T> &A) {
        fwt(A,1);
        for (int i = 0; i < n; i++)
            a[i] = a[i]*a[i];/// Don't forget modulo if required
        fwt(A,0);
    }
};
FWT<ll> fwt;
```

5.1.5 bitwise convolution

This allows to do $C(k) = \sum_{k=i|j} A_i B_j$ and $C(k) = \sum_{k=i\&j} A_i B_j$

```
/*
Autor: Oscar Vargas Pabon

and convolution tested in https://judge.yosupo.jp/problem/
    bitwise_and_convolution
or convolution currently untested

Note it wont work well for A*A

Taken from https://codeforces.com/blog/entry/119082

Im assuming from my template:
#define rep(i,strt,end) for(int i = strt ; i !=int(end) ; (int(strt)<int(end))
    ?++i:--i )
*/


template<typename T>
class btw_conv{
public:
    void resize(vector<T> &vec){
        int e=1;while(e<int(vec.size()))e*=2;
        vec.resize(e);
    }

    void trans_subset(vector<T> &vec,int sd){
        // I assume sd\in\{-1,1\}
        for(int e=1;e<int(vec.size());e*=2)rep(i,0,vec.size()){
            if(i&e)vec[i]+=vec[i^e]*sd;
        }
    }
```

```
	void or_conv(vector<T> &A,vector<T> &B){
		// the answer is shown in A; I assume |A|=|B|=2^x for some x
		trans_subset(A,1);trans_subset(B,1);
		rep(i,0,A.size())A[i]*=B[i];
		trans_subset(A,-1);
	}

	void trans_superset(vector<T> &vec,int sd){
		// I assume sd\in\{-1,1\}
		for(int e=1;e<int(vec.size());e*=2)rep(i,0,vec.size()){
			if(i&e)vec[i^e]+=vec[i]*sd;
		}
	}

	void and_conv(vector<T> &A,vector<T> &B){
		// the answer is shown in A; I assume |A|=|B|=2^x for some x
		trans_superset(A,1);trans_superset(B,1);
		rep(i,0,A.size())A[i]*=B[i];
		trans_superset(A,-1);
	}
};
```

5.1.6 divisibility convolution

This allows to do $C(k) = \sum_{k=gcd(i,j)} A_i B_j$ and $C(k) = \sum_{k=lcm(i,j)} A_i B_j$

```
/*
Autor: Oscar Vargas Pabon

GCD_CONV tested in https://judge.yosupo.jp/problem/gcd_convolution
LCM_CONV tested in https://codeforces.com/gym/105053/problem/G
                        https://judge.yosupo.jp/problem/
                           lcm_convolution
Note it wont work well for A*A

Taken from https://codeforces.com/blog/entry/119082

Im assuming from my template:
#define rep(i,strt,end) for(int i = strt ; i !=int(end) ; (int(strt)<int(end))
    ?++i:--i )
*/


template<typename T>
class div_conv{
	vector<int> prm;
public:
	div_conv(int limit){
		vector<bool> crb(limit,1);crb[0]=crb[1]=0;
		rep(i,0,limit)if(crb[i]){
			prm.push_back(i);
			if(i*1ll*i<lint(limit))for(int j=i*i;j<limit;j+=i){
				crb[j]=0;
			}
		}
	}

	void zeta_mult(vector<T> &vc){
		// vc'_x=\sum_{x|y}vc_y
		for(int p:prm)for(int i=(vc.size()-1)/p;i;--i){
			vc[i]+=vc[i*p];// vc[i]%=mod;
		}
	}
	void mobi_mult(vector<T> &vc){
		// vc_x=\sum_{x|y}vc'_y
		for(int p:prm)for(int i=1;i*p<int(vc.size());++i){
			vc[i]-=vc[i*p];// vc[i]%=mod;
		}
	}
	void gcd_conv(vector<T> &A,vector<T> &B){
		// I assume |A|=|B|
		// A'_x=\sum_{x=gcd(u,v)}A_uB_v
		zeta_mult(A);zeta_mult(B);
		rep(i,0,A.size())A[i]*=B[i];
		mobi_mult(A);
	}

	void zeta_div(vector<T> &vc){
		// vc_x=\sum_{y|x}vc'_y
		for(int p:prm)for(int i=1;i*p<int(vc.size());++i){
			vc[i*p]+=vc[i];// vc[i*p]%=mod;
		}
	}
	void mobi_div(vector<T> &vc){
		// vc'_x=\sum_{y|x}vc_y
		for(int p:prm)for(int i=(vc.size()-1)/p;i;--i){
			vc[i*p]-=vc[i];// vc[i*p] %=mod;
		}
	}

	void lcm_conv(vector<T> &A,vector<T> &B){
		// I assume |A|=|B|
		// A'_x=\sum_{x=lcm(u,v)}A_uB_v
		zeta_div(A);zeta_div(B);
		rep(i,0,A.size())A[i]*=B[i];
		mobi_div(A);
	}

};
```

5.2 extended gcd

Remember there exist __gcd(a,b); and its usable

```
// https://cp-algorithms.com/algebra/extended-euclid-algorithm.html
int gcd(int a, int b, int& x, int& y) {
	// a*x + b*y = gcd(a,b)
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

5.3 Xor Basis

```
/*
Autor: Oscar Vargas Pabon

Remember that the amount of ways of generating x, its 2**(|S|-|bs|) if x is
    generable
*/

// ultra slim impl
// untested
int red(const vector<int>&bs,int x){for(int ac:bs)x=min(x,x^ac);return x;}
bool add(vector<int>&bs,int x){x=red(x);if(x)bs.pb(x);return x;}
int mx(const vector<int>&bs){int x=0;for(int ac:bs)x=max(x,x^ac);return x;}

// esta idea la tuve en la superregional
// no esta testeado
class XorR{
	public:
	vector<int> bs,us,ori;
	XorR()=default;
	int red(int x){for(int ac:bs)x=min(x,x^ac);}
```

```
        int mx(){int x=0;for(int ac:bs)x=max(x,x^ac);}

        vector<int> raw_rec(int msk){
                vector<int> res;
                rep(i,0,n)if(msk&(1<<i))res.pb(ori[i]);
                return res;
        }
        vector<int> rec(int x){
                // reconstruye los elementos que generan x
                int msk=0;
                rep(i,0,bs.size())if(x>(x^bs[i])){
                        x^=bs[i]; msk^=us[i];
                }
                return raw_rec(msk);
        }
        vector<int> mx_rec(){
                // reconstruye los elementos que generan el maximo
                int msk=0,x=0;
                rep(i,0,bs.size())if(x<(x^bs[i])){
                        x^=bs[i]; msk^=us[i];
                }
                return raw_rec(msk);
        }
        bool add(int x){
                int s=1<<bs.size(),ox=x;
                rep(i,0,bs.size())if(x>(x^bs[i])){
                        x^=bs[i]; s^=us[i];
                }
                if(x)bs.pb(x),us.pb(s),ori.pb(ox);
                return x;
        }
};

// range static Xor basis
// tested in https://codeforces.com/contest/1100/problem/F
const int lgi=20;
class XorP{
        public:
        array<int,lgi> bs,tm;
        XorP(){
                rep(i,0,lgi)bs[i]=0;
                rep(i,0,lgi)tm[i]=-1;
        }

        int red(int x,int t){
                rep(i,lgi-1,-1)if(tm[i]>=t)x=min(x,x^bs[i]);
                return x;
        }
        void add(int x, int t){
                rep(i,lgi-1,-1) if( (x>>i)&1 ) {
                        if(tm[i] < t ){
                                swap(tm[i],t);swap(bs[i],x);
                        }

                        x^=bs[i];
                }
        }
        int mx(int t){
                int rs=0;
                rep(i,lgi-1,-1)if(tm[i]>=t)rs=max(rs,rs^bs[i]);
                return rs;
        }
};


// range static XorBasis finding the kth generable element and the
//  order of a given generable element
// used in https://codeforces.com/contest/2143/problem/F
const int lgi=30;
class XorB{
        public:

        vector<pair<int,int>> bs;
        XorB(){bs=vector<pair<int,int>>(lgi,{0,-1});}
        int redu(int x,int y){rep(i,lgi-1,-1)if(bs[i].second>=y)x=min(x,x^bs[i
            ].first); return x;}
        void add(int x,int y){
                pair<int,int> act={x,y};
                rep(i,lgi-1,-1)if((act.first>>i)&1){
                        if(bs[i].second>act.second)act.first^=bs[i].first;
                        else{
                                bs[i].first^=act.first;
                                swap(bs[i],act);
                        }
                }
        }
        // note that ord(kth(k,y),y)==k and kth(ord(x,y),y)==x
        int kth(int k,int y){
                // returns which is the kth vector (in increasing order) of
                    the span of bs(>=y)
                int msk=0;rep(i,0,lgi)if(bs[i].second>=y) {
                        msk|=(k&1)<<i; k>>=1;
                }

                int rs=0;rep(i,lgi-1,-1)if(bs[i].second>=y){
                        if(((rs>>i)&1)^((msk>>i)&1))rs^=bs[i].first;
                }
                return rs;
        }
        int ord(int x,int y){
                // returns which kth does x have on the span of bs(>=y)
                int k=0,e=0;rep(i,0,lgi)if(bs[i].second>=y){
                        k|=((x>>i)&1)<<e;
                        ++e;
                }
                return k;
        }

        // degrees of freedom
        int fred(int y){int rs=0;rep(i,0,lgi)if(bs[i].second>=y)++rs;return rs
            ;}

};

/* Xor basis modulo m */
/* taken from errogorn's blog, maroonrk's comment https://codeforces.com/blog/
    entry/98376
        I simply adapted it to my style. It seems interesting to find the kth
            generable elements. However,
                it may scale up quickly for certain values of mod.
*/
lint gcd(lint a, lint b, lint& x, lint& y) {
        // a*x+b*y==gcd(a,b);
    if (b == 0ll) {
        x = 1;
        y = 0;
        return a;
    }
    lint x1, y1;
    lint d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}


const lint mod=360; const int v_sz=10;
class MBasis{
        public:
        vector<vector<lint>> bas;
```

```
        MBasis(){ bas=vector<vector<lint>>(v_sz,vector<int>(v_sz,0)); }
        void v_sum( vector<lint>&va,const vector<lint> &vb,int k){ rep(i,0,
            v_sz)va[i]+=vb[i]*k; }
        void add( vector<lint> &v ) {
                rep(i,0,v_sz)if(v[i]){
                        int x,y;int g=gcd(v[i],bas[i][i],x,y);
                        int z=(bas[i][i])?bas[i][i]/g : mod/g,w=- v[i]/g;
                        rep(j,0,v_sz)tie(bas[i][j],v[j])=make_pair((v[j]*x+bas
                            [i][j]*y)%mod,(z*v[j]+w*bas[i][j])%mod);
                }
        }
        bool red( vector<lint> &v ){
                rep(i,0,v_sz)if(v[i]){
                        if(!bas[i][i]||v[i]%bas[i][i])return 0;
                        v_sum(v,bas[i],- v[i]/bas[i][i]);
                }
                return 1;
        }

        vector<lint> mx_span(){
                vector<lint> rs(v_sz,0);
                rep(i,0,v_sz)if(bas[i][i]) v_sum( rs, bas[i], (mod-1-v[i])/bas
                    [i][i] );
                return rs;
        }

        lint sz_span(){
                lint res=1;rep(i,0,v_sz)if(bas[i][i]){
                        res*=mod/bas[i][i];
                }
                return res;
        }

};
```

5.4 prime stuff

The code has pollard rho that works in expected $O(\sqrt{4}n)$ and miller-rabin that we can (?) assume it won't fail as a prime tester.

```
/*
Accepted version of the solution to the Factoring Large Integers problem
Use Factorization with Pollards Rho + Primality de Miller-Rabin
Note: The IO part was cut down by edition

Author: Juan Diego Collazos
Date : 20/5/2025
Problem : https://onlinejudge.org/external/114/11476.pdf

Reference:
Pollards Rho Algorithm for Prime Factorization: https://www.geeksforgeeks.org/
    pollards-rho-algorithm-prime-factorization/
Primality Test | Set 3 (Miller-Rabin) : https://www.geeksforgeeks.org/
    primality-test-set-3-miller-rabin/
*/

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

ll gcd(ll a, ll b) {
  while (b != 0) {
    ll t = b;
    b = a % b;
    a = t;
  }
  return a;
}

ll mulmod(ll a, ll b, ll mod) {
  ll ans = 0;
  a %= mod;
  while (b > 0) {
    if (b & 1)
      ans = (ans + a) % mod;
    a = (a * 2) % mod;
    b >>= 1;
  }
  return ans;
}

ll powmod(ll base, ll exp, ll mod) {
  ll ans = 1;
  base %= mod;
  while (exp > 0) {
    if (exp & 1)
      ans = mulmod(ans, base, mod);
    base = mulmod(base, base, mod);
    exp >>= 1;
  }
  return ans;
}

bool is_prime(ll n, int k = 5){
  bool ans;
  if (n < 2)
    ans = false;
  else if (n == 2 || n == 3)
    ans = true;
  else if (n % 2  == 0)
    ans = false;
  else{
    ll r = 0, d = n - 1;
    while (d % 2 == 0){
      d >>= 1;
      r++;
    }
    ans = true;
    int i = 0;
    while(i < k && ans){
      ll a = rng() % (n - 3) + 2;
      ll x = powmod(a, d, n);
      if (x != 1 && x != n -1){
        ll j = 0;
        bool composite = true;
        while(j < r - 1 && composite){
          x = powmod(x, 2, n);
          if (x == n - 1)
            composite = false;
          ++j;
        }
        if (composite)
          ans = false;
      }
      ++i;
    }
  }
  return ans;
}

ll f(ll x, ll n, ll c){
  return (mulmod(x, x, n) + c) % n;
}

ll pollards_rho(ll n) {
  ll ans;
  if (n % 2 == 0)
```

```
    ans = 2;
  else if(is_prime(n))
    ans = n;
  else{
    ans = -1;
    while(ans == -1){
      ll c = rng() % n;
      ll x = 2, y = 2, d = 1;
      while(d == 1) {
        x = f(x, n, c);
        y = f(f(y, n, c), n, c);
        d = gcd(abs(x - y), n);
      }
      if (d != n)
        ans = d;
    }
  }
  return ans;
}

void factor(ll n, vector<ll>& factors){
  if (n != 1){
    if (is_prime(n)){
      factors.push_back(n);
    }else{
      ll k = pollards_rho(n);
      factor(k, factors);
      factor(n / k, factors);
    }
  }
}
```

6 Combinatorics

Muchas de estas propiedades fueron sacadas de https://cp-algorithms.com/combinatorics

6.1 Properties

Remember $n! = n \cdot (n-1)!$ and $(n)_k = \frac{n!}{(n-k)!} = n(n-1)\cdots(n-k+2)(n-k+1)$ (this last one generalizes for $n \notin \mathbb{N}$).

6.1.1 n Choose k

Remember that it can be calculated by the recursive formula, by precomputing factorials modulo p, or in $min(k, n-k)$ by calculating $(k!)^{-1} \cdot (n)_k (mod p)$.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\sum_{i=0}^{n} x^i \binom{n}{i} = (x+1)^n$$

$$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}$$

$$\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$$

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

$$\sum_{i=0}^{n} i\binom{n}{i} = n \cdot 2^{n-1}$$

Wilson's theorem:

$$(p-1)! \equiv -1(\text{mod p})$$

Lucas theorem:

$$\binom{n}{k} \equiv \prod \binom{n_i}{k_i} (\text{mod p})$$

where $n = \sum n_i \cdot p^i$ and $k = \sum k_i \cdot p^i$

Vandermonde's identity:

$$\sum_{k=0} n \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$$

6.1.2 Stirling God

Stirling number of second kind $\left\{ {n \atop k} \right\}$ counts the amount of ways to partition n unlabelled objects in k non-empty subsets.

$$\left\{ {n+1 \atop k} \right\} = k\left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

$$\left\{ {n \atop k} \right\} = \frac{1}{k!}\sum_{i\geq 0}(-1)^{k-i}\binom{k}{i} i^n$$

$$\left\{ {n \atop k} \right\} \equiv \binom{z}{w} (mod 2)$$

Where $z = n - \left\lceil \frac{k+1}{2} \right\rceil$ and $w = \left\lfloor \frac{k-1}{2} \right\rfloor$.

As bell numbers B_n count the amount of ways to partition n elements then

$$B_n = \sum_{i=0}^{n} \left\{ {n \atop k} \right\}$$

Stirling number of the first kind (eventually).

6.1.3 Formal Power Series

Note: This is still poor in properties.

$$(1+F)^{\lambda} = \sum_{n\geq 0} \binom{\lambda}{n} F^n$$

Remember it generalizes to $\lambda \notin \mathbb{N}$ by using the $(n)_k$ definition of $\binom{n}{k}$.

$$(1-x)^{-1} = \sum_{n\geq 0} x^n \text{ for } 0 \leq x < 1$$

$$e^x = \sum_{n\geq 0} \frac{x^n}{n!}$$

$$(1-x)^{-k} = \sum_{n\geq 0} \binom{n+k-1}{n} x^n$$

$$-ln(1-x) = \sum_{x\geq 1} \frac{x^n}{n}$$

$$F \cdot G = \sum_{n\geq 0} \left(\sum_{i=0}^{n} f_i g_{n-i} \right) x^n \quad \text{for OGF}$$
$$= \sum_{n\geq 0} \left(\sum_{i=0}^{n} \binom{n}{i} f_i g_{n-i} \right) x^n \quad \text{for EGF}$$

$$F^k = \sum_{n\geq 0} \left(\sum_{i_1+i_2+\cdots+i_k=n} \prod_{j=1}^{k} f_{i_j} \right) x^n \quad \text{for OGF}$$
$$= \sum_{n\geq 0} \left(\sum_{i_1+i_2+\cdots+i_k=n} \binom{n}{i_1 \cdot i_2 \cdots i_k} \prod_{j=1}^{k} f_{i_j} \right) x^n \quad \text{for EGF}$$

$$\sum_{n\geq 0} \sum_{k\geq 0} \binom{n}{k} x^n y^k = \frac{1}{1-x-xy}$$

6.1.4 Modular Arithmetic

$\phi(n)$ is the amount of coprime elements to n in the range $[0, n)$.

$$\phi(a \cdot b) = \phi(a)\phi(b)$$
$$\phi(p) = p - 1$$
$$\phi(m) = m \prod_{i=1}^{k} (1 - \frac{1}{p_i}) \text{ where } m = \prod_{i=0}^{k} p_i^{e_i}$$
$$a^{\phi(m)} \equiv 1 (\text{mod } m)$$

6.1.5 Chinese Remainder Theorem

The problem is to find x such that

$$\begin{cases} x \equiv a_1 (\text{mod } m_1) \\ \vdots \\ x \equiv a_k (\text{mod } m_k) \end{cases}$$

and all m_i's are pairwise coprime.

$$R^* = \sum_{i=1}^{k} a_i \left[inv_{m_i} \left(\prod_{j\neq i} m_j \right) \prod_{j \neq i} m_j \right]$$

Remember R^* works modulo $\prod m_j$.

6.1.6 Lagrange Interpolation

Find lowest degree polinome that satisfies $P(x_i) = y_i \forall_i$ given $[(x_1, y_1), \cdots, (x_k, y_k)]$.

$$P = \sum_{i=0}^{k} \left(\frac{\prod_{j\neq i}(x_j - x)}{\prod_{j\neq i}(x_j - x_i)} \right) y_i$$

Remember that if I only want P to evaluate on a specific x, then I can just make x in the equation as the value to avoid building the explicit polinomial.

6.1.7 Catalan numbers

Catalan Numbers

$$C_0 = C_1 = 1$$
$$C_i = \sum_{k=0}^{i-1} C_k C_{i-1-k}$$
$$C_i = \frac{1}{n+1} \binom{2n}{n}$$
$$C_i = \frac{4i-2}{i+1} C_{i-1}, C_0 = 1$$

6.1.8 Fibonacci

Fibonacci

$$fibo(0) = 0, fibo(1) = 1, fibo(n) = fibo(n-1) + fibo(n-2)$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} fibo(n+1) & fibo(n) \\ fibo(n) & fibo(n-1) \end{bmatrix}$$

6.2 Techniques

Principle of Inclusion-Exclusion (PIE)

Remember there are more and more ways to write PIE that can help with other stuff (I think one of those is mobius function, but more on that when im good).

$$|\cup S_i| = \sum_{\emptyset \neq J \in \{1..N\}} (-1)^{|J|-1} |\cap_{i\in J} S_i|$$

Stars and bars

$$\binom{n+k-1}{n}$$

Number of ways to add n identical objects into k labeled boxes.

Armonica: $\sum_{k=1}^{n} \frac{1}{k} = O(lgn)$

7 Geometry

7.1 geomTemplate

```
/*
Autor: Oscar Vargas Pabon

GRAFOS PLANARES:::::
c:componentes conexos ; R:regiones
m:aristas; n:nodos
R+n=m+c+1

Probado en UVA 13117 y 11894
*/
const double eps=1e-8;
typedef double Tpt;
class Pt{
	public:
	Tpt x,y;
	Pt()=default;
	Pt(Tpt x, Tpt y ) : x(x),y(y){};
	Pt rot90(){return{-y,x};}

	// NOTA: puedo querer prescindir del sqrt
	Tpt norm()const{return sqrt(x*x+y*y);}
	Tpt norm2()const{return x*x+y*y;}
```

```
        Pt scale(Tpt v)const{return{v*x,v*y};}
        Pt operator -()const{return{-x,-y};}
        Pt operator -(const Pt&o)const{return{x-o.x,y-o.y};}
        // suma de vectores
        Pt operator +(const Pt&o)const{return{x+o.x,y+o.y};}

        // point product; 0->ortogonal; +->same dir ; - ->opposite dir
        Tpt operator *(const Pt&o)const{return x*o.x+y*o.y;}
        // cross product; 0->colinear; +->left side; - ->right side
        Tpt operator %(const Pt&o)const{return x*o.y-y*o.x;}

// abs(v%u)==u.norm2()*v.norm2()*sin(theta)
// (v*u)==u.norm2()*v.norm2()*cos(theta)
// v%u == area paralelogramo delimitado por u y v
        bool operator<(const Pt &o)const{return make_pair(x,y)<make_pair(o.x,o
            .y);}
        bool operator==(const Pt&o)const{return x==o.x&&y==o.y;}
        bool operator!=(const Pt&o)const{return!(*this==o);}
};
ostream & operator << (ostream &out, const Pt &p){ out << "("<<p.x<<","<<p.y<<
    ")"; return out; }

class Ln{
        public:
        Pt p,v;// L(t)=p+tv
        Ln()=default;

        // L(0)=a; L(1)=b; L(t) 0<=t<=1 esta en el segmento ab
        Ln(Pt a,Pt b):p(a),v(b-a){};
        Pt eval(Tpt t) { return p+v.scale(t);}
        // hallo t tal que L(t)== interseccion entre this y o
        // recordar que puedo trabajar en enteros hasta la division
        Tpt inter(const Ln&o){return{((p-o.p)%o.v)/(v%o.v)};}

        // dados v,u vectores; el valor h donde {0,v(h),u} forman un triangulo
            rectangulo
        // con con angulo recto A{o,v(h),u} =90 es tal que u*v=h*(v*v)
};

ostream & operator << (ostream &out, const Ln &l){ out << "<L(t)="<<l.p<<"+t"
    <<l.v<<">"; return out; }
```

7.2 Convex Hull (2D)

```
/*
Autor: Oscar Vargas Pabon

Works in O(n lgn)
Think on impl with vector or smthng
There are stuff / versions that are incremental

Tested in https://codeforces.com/problemset/problem/1017/E
          https://open.kattis.com/problems/convexhull

I assume from my template:
#define rep(i,strt,end) for(int i = strt ; i !=int(end) ; (int(strt)<int(end))
    ?++i:--i )
#define pb push_back
#define pob pop_back
*/

vector<Pt> convex_hull(Pt *arr,int n){
        sort(arr,arr+n);
        vector<Pt> up,dwn;
        auto crss=[](const Pt&a,const Pt &b,const Pt &bs){return (a-bs)%(b-bs)
            ;};
        rep(i,0,n){
                Pt &ac=arr[i]; int pz=up.size(),dz=dwn.size();
                while(pz>1&& crss(ac,up[pz-2],up[pz-1])>=0 ){
                        --pz; up.pob();
                }
                if(pz>=1&&up.back().x==ac.x&&up.back().y<ac.y)up.pob();
                if(up.empty()||up.back().x<ac.x||up.back().y<ac.y)up.pb(ac);

                while(dz>1&& crss(ac,dwn[dz-2],dwn[dz-1])<=0 ){
                        --dz; dwn.pob();
                }
                if(dz>=1&&dwn.back().x==ac.x&&dwn.back().y>ac.y)dwn.pob();
                if(dwn.empty()||dwn.back().x<ac.x||dwn.back().y>ac.y)dwn.pb(ac
                    );
        }
        vector<Pt> res=dwn;
        rep(i,up.size()-1,-1){
                if( !(res.back()==up[i]) ) res.pb(up[i]);
        }
        // idebug(up);idebug(dwn);
        if(int(res.size())>1&&res.front()==res.back())res.pob();
        return res;
}


vector<int> hullInd(const vector<Pt>& v) {
        //adapted from https://github.com/bqi343/cp-notebook

        int ind = int(min_element(all(v))-begin(v));
        vector<int> cand, C={ind};
        rep(i,0,v.size()) if (v[i] != v[ind]) cand.pb(i);
        sort(all(cand),[&](int a, int b) {
                // sort by angle, tiebreak by distance
                Pt x = v[a]-v[ind], y = v[b]-v[ind]; Tpt t = x%y;
                return t != 0 ? t > 0 : x*x < y*y;
        });
        auto crss=[](const Pt&a,const Pt &b,const Pt &bs){return (a-bs)%(b-bs)
            ;};
        for (int c:cand){
                while (int(C.size()) > 1 && crss(v[C.back()],v[c],v[C[C.size()
                    -2]]) <= 0)
                        C.pob();
                C.pb(c); }
        return C;
}
vector<Pt> hull(const vector<Pt>& v) {
        vector<int> w = hullInd(v); vector<Pt> res;
        for(int t:w) res.pb(v[t]);
        return res; }
```

8 linAlg

8.1 LP Simplex

Recordar que la implementacion de simplex requiere que el sistema este en 'slack' form

$$\begin{cases} c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,k}x_k = y_1 \\ \vdots \\ c_{m,1}x_1 + c_{m,2}x_2 + \cdots + c_{m,k}x_k = y_m \end{cases}$$

$$\forall_i x_i \geq 0$$

Condiciones del estilo $\sum_i c_i \cdot x_i \leq y$ se convierten a 'slack' form anadiendo una 'slack' variable s que tiene coeficiente 0 en todas las filas (y objetivo) y 1 en la fila actual. El sistema queda como $\sum_i c_i \cdot x_i + s = y$ para $s \geq 0$. Las condiciones del estilo $\sum_i c_i \cdot x_i \geq y$ son convertidas a $\sum_i (-c_i) \cdot x_i \leq -y$ (y luego a 'slack' form como lo anterior).

Lo use en el gym 101242 I (Road Times ICPC WF 2016).

```
/*
Tomado del notebook de DebureoMinkyuParty
https://github.com/koosaga/DeobureoMinkyuParty
*/
using T = long double;
```

```
const int N = 410, M = 30010;
const T eps = 1e-7;
int n, m;
int Left[M], Down[N];
// time complexity: exponential. fast $O(MN^2)$ in experiment. dependent on
    the modeling.
// Ax <= b, max c^T x.
T a[M][N], b[M], c[N], v, sol[N];
bool eq(T a, T b) { return fabs(a - b) < eps; }
bool ls(T a, T b) { return a < b && !eq(a, b); }
void init(int p, int q) {
        n = p; m = q; v = 0;
        for(int i = 1; i <= m; i++){
                for(int j = 1; j <= n; j++) a[i][j]=0;
        }
        for(int i = 1; i <= m; i++) b[i]=0;
        for(int i = 1; i <= n; i++) c[i]=sol[i]=0;
}
void pivot(int x,int y) {
        swap(Left[x], Down[y]);
        T k = a[x][y]; a[x][y] = 1;
        vector<int> nz;
        for(int i = 1; i <= n; i++){
                a[x][i] /= k;
                if(!eq(a[x][i], 0)) nz.push_back(i);
        }
        b[x] /= k;
        for(int i = 1; i <= m; i++){
                if(i == x || eq(a[i][y], 0)) continue;
                k = a[i][y]; a[i][y] = 0;
                b[i] -= k*b[x];
                for(int j : nz) a[i][j] -= k*a[x][j];
        }
        if(eq(c[y], 0)) return;
        k = c[y]; c[y] = 0;
        v += k*b[x];
        for(int i : nz) c[i] -= k*a[x][i];
}
// 0: found solution, 1: no feasible solution, 2: unbounded
int lp_solve() {
        for(int i = 1; i <= n; i++) Down[i] = i;
        for(int i = 1; i <= m; i++) Left[i] = n+i;
        while(1) { // Eliminating negative b[i]
                int x = 0, y = 0;
                for(int i = 1; i <= m; i++) if (ls(b[i], 0) && (x == 0 || b[i]
                    < b[x])) x = i;
                if(x == 0) break;
                for(int i = 1; i <= n; i++) if (ls(a[x][i], 0) && (y == 0 || a
                    [x][i] < a[x][y])) y = i;
                if(y == 0) return 1;
                pivot(x, y);
        }
        while(1) {
                int x = 0, y = 0;
                for(int i = 1; i <= n; i++)
                if (ls(0, c[i]) && (!y || c[i] > c[y])) y = i;
                if(y == 0) break;
                for(int i = 1; i <= m; i++)
                if (ls(0, a[i][y]) && (!x || b[i]/a[i][y] < b[x]/a[x][y])) x =
                    i;
                if(x == 0) return 2;
                pivot(x, y);
        }
        for(int i = 1; i <= m; i++) if(Left[i] <= n) sol[Left[i]] = b[i];
        return 0;
}
```

8.2 MatMultiply

```
/*
Autor: Oscar Vargas Pabon

mat_mult tested in UVA 13298 (A Fibonacci Family Formula)
*/

typedef int tmat;
typedef vector<vector<tmat>> mat;
mat mat_mult(const mat &m1,const mat &m2){
        // O(n*m*h)
        assert(!m2.empty()); assert(!m1.empty()); assert(m2.size()==m1[0].size
            ());
        int n = m1.size(),m=m2.size(),h=m2[0].size();
        mat res(n,vector<tmat>(h,0));
        // Adamant says this order is goty
        // https://codeforces.com/blog/entry/129292
        rep(i,0,n) rep(k,0,m) rep(j,0,h) res[i][j] = (res[i][j]+ m1[i][k]*1ll*
            m2[k][j])%mod;
        return res;
}
```

8.3 slae

When dealing with modulo 2 stuff bitset can also carry hard

```
// https://cp-algorithms.com/linear_algebra/linear-system-gauss.html
// The input to the function gauss is the system matrix
// a. The last column of this matrix is vector
// b

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

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

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

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

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

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

```
            return INF;
    return 1;
}
```

9 Cosas random

9.1 Calcular $\lfloor log_2 x \rfloor$

Nota, tambien está en Fenwick Tree.

En C++

```
sizeof(int)*8 - __builtin_clz(x)-1;
```

En Python

```
x.bit_length()-1
```

9.2 My tipsNtricks stuff

Remember the trick of the transpose graph in $O((n+m)lgn)$ using the set to keep track of the unvisited ones.

```
/*
Autor: Oscar Vargas Pabon
These are various tricks to be used at will
*/

// iterating over ranges [l_i,r_i]\subseteq[L,R] such that (assuming integer division)
//////   \forall_{j,h\in[l_i,r_i]}n/j=n/h
/// can be seen in https://codeforces.com/problemset/problem/2072/G
void diviter(int n, int L, int R ){
        int l = L,r;
        while( l <= R ){
                int nh = n/l;
                r = min(R,n/nh);

                /// do stuff with the range i\in[l,r] n/i=nh

                l=r+1;
        }
}

// ham-path
// phi(nd,msk)= strt\in N(nd) OR OR_{e\in N(nd)} phi(nd,msk|(1<<nd))
// In cases where I'm only looking for cycles containing msk, then I will start
// from every node; however, I don't need to erase the memory. The reason is that
// if such cycle exists I can find it with any strt\in msk, if there is none it won't
// matter which strt\in msk I choose.
// This idea can be seen in https://codeforces.com/problemset/problem/1804/E



/////// Range Xor Shenanigans (assuming range is [l,r])
// Question: Is subset in range interesting? Is it doable in O(1)?
// I used this in industrial nim and in https://codeforces.com/problemset/problem/2056/F2

// calculating xor in range
int xor_range(int m){
        //calculates XOR_{x=0}^m x in O(1). Observe it includes m.
        int res=(m&1)?0:m;
        return res^=((m+1)>>1)&1;
}

// The idea is that if I fix the bits that are always set, then I can exclude them
// x= 0010100
// y= ??1?1??
// meaning that if I exclude the 1's it becomes a xor_range problem
// this involves calculating the new m (since the actual m may not me superset )
//    and re-adding again the 'erased' bits after xor_range
int super_set(int m,int x){
        //calculates XOR_{y=0}^m (y&x==x)*y in O(lgU)
        if(x>m)return 0;

        //Finding last superset in range
        int lgi=ilog2(m-1),lst=x;
        rep(e,lgi,-1)if((lst|(1<<e))<m)lst|=1<<e;


        int neo=0,xx=x,e=0;
        while(lst){//Reducing to xor_range
                if(!(xx&1))neo|=(lst&1)<<e++;
                lst>>=1;xx>>=1;
        }
        int skw=xor_range(neo);xx=x;

        int res=0; e=0;
        while(skw){//reconstruct answer from xor_range
                if(!(xx&1)){
                        res|=(skw&1)<<e;
                        skw>>=1;
                }
                xx>>=1; ++e;
        }

        // add the bits in x (which are always set)
        if(x)res|=((neo+1)&1)*x;
        return res;
}

///// I think it is the key to aladdin from COCI 2009/2010 that I found in the CCPL
/// floor sum shenanigans
// taken from https://asfjwd.github.io/2020-04-24-floor-sum-ap/
long long FloorSumAP(long long a, long long b, long long c, long long n){}
// calculating \sum_{x=0}^n \lfloor (ax+b)/c \rfloor
  if(!a) return (b / c) * (n + 1);
  if(a >= c || b >= c) return ( ( n * (n + 1) ) / 2) * (a / c) + (n + 1) * (b / c) + FloorSumAP(a % c, b % c, c, n);
  long long m = (a * n + b) / c;
  return m * n - FloorSumAP(c, c - b - 1, a, m - 1);
}
```

9.3 template

9.3.1 c++

```
/*
 --------
|    ___ |
|  ,',.(`|
| :  `'  |
| :)  _  (|
|  `:_)_,|
|--------|

Autor: Oscar Vargas Pabon
Fecha:

*/

#include <bits/stdc++.h>

typedef long long lint;

using namespace std;
```

```cpp
#define debug(args...) { string _s = #args; replace(_s.begin(), _s.end(), ',', ' '); stringstream _ss(_s); istream_iterator<string> _it(_ss); raw_debug(_it, args);}
void raw_debug(istream_iterator<string> it) {cerr<<endl;}
template<typename T, typename... Args>
void raw_debug(istream_iterator<string> it, T a, Args... args) { cerr <<"<"<< *it << "->" << a << "> "; raw_debug(++it, args...); }
#define idebug(v) {cout<<'['<<#v<<']';for(const auto &el:v)cout << ' ' << el; cout << endl;}
#define adebug(ar,n) {cout<<'['<<#ar<<']';for(int i=0;i<n;++i)cout << ' ' << ar[i]; cout << endl;}

#define rep(i,strt,end) for(int i = strt ; i !=int(end) ; (int(strt)<int(end))?++i:--i )
#define rall(vec) vec.rbegin(), vec.rend()
#define all(vec) vec.begin(), vec.end()
#define pb push_back
#define pob pop_back
#define pf push_front
#define pof pop_front

mt19937_64 rng_64( chrono::steady_clock::now().time_since_epoch().count() );
int ilog2( int num ) { return 8*sizeof(int) - __builtin_clz( num ) - 1; }
lint mpow(lint x,lint e,lint m){lint res=1ll;while(e){if(e&1ll)res=(res*x)%m;e>>=1;x=(x*x)%m;}return res;}

const int template_limit = 1e6;
int a[template_limit], b[template_limit];

void solve() {

}

int32_t main(){
        ios_base::sync_with_stdio(false);
    cin.tie(NULL);
        cout << setprecision(12) << fixed;

    int t = 2;
    cin >> t; ++t;
    while ( --t ) {
                solve();
    }
        return 0;
}
```

9.3.2 Python

Python es una calculadora glorificada. Pero tiene algunos usos...

```python
# ==============================================
# ENTRADA Y SALIDA EN PYTHON (PROGRAMACION COMPETITIVA)
# ==============================================
# Autor: Ejemplo generado por ChatGPT
# Descripcion:
# Este archivo muestra distintas maneras eficientes de
# recibir e imprimir datos en Python para programacion competitiva.

import sys

# ------------------------------
# 1. Lectura basica de una linea
# ------------------------------
# sys.stdin.readline() lee una linea completa mas rapido que input().
linea = sys.stdin.readline().strip()
print("Entrada leida:", linea)

# ------------------------------
# 2. Leer varios numeros en una linea
# ------------------------------
a, b = map(int, sys.stdin.readline().split())
print("Suma:", a + b)

# ------------------------------
# 3. Leer multiples lineas con cantidad conocida
# ------------------------------
n = int(sys.stdin.readline())
for i in range(n):
    x, y = map(int, sys.stdin.readline().split())
    print(f"Resultado {i+1}:", x + y)

# ------------------------------
# 4. Redefinir input para simplicidad
# ------------------------------
input = sys.stdin.readline  # ahora input() sera rapido
n = int(input())
for _ in range(n):
    a, b = map(int, input().split())
    print(a * b)

# ------------------------------
# 5. Salida rapida con sys.stdout.write
# ------------------------------
sys.stdout.write("Esto es una salida rapida\n")

# ------------------------------
# 6. Salida eficiente acumulando resultados
# ------------------------------
input = sys.stdin.readline
n = int(input())
resultados = []
for _ in range(n):
    a, b = map(int, input().split())
    resultados.append(str(a - b))
sys.stdout.write("\n".join(resultados) + "\n")

# ------------------------------
# 7. Leer hasta EOF (fin de archivo)
# ------------------------------
# Muy util cuando no se da el numero de casos.
for linea in sys.stdin:
    datos = linea.strip().split()
    if not datos:
        continue
    nums = list(map(int, datos))
    print("Suma de la linea:", sum(nums))
```