Cuadernillo

Sparkies

Octubre 2025

Índice

1. Data Structures 1
- 1.1. Disjoint Set Union Find (DSU) 1
- 1.2. Segment Tree 2
  - 1.2.1. Lazy Recursive 2
  - 1.2.2. Iterative 3
  - 1.2.3. Persistent 3
- 1.3. Fenwick Tree 4
- 1.4. Treap 4
- 1.5. Mo Algo 5
- 1.6. tipsNtricks 6

2. Trees 6
- 2.1. Centroid 6
- 2.2. Heavy Light Decomposition (HLD) 7
- 2.3. Lowest Common Ancestor (LCA) 7
  - 2.3.1. Euler Tour version 8
  - 2.3.2. Binary Lifting 8

3. Graph 9
- 3.1. Dinitz 9
- 3.2. Blossoms 10
- 3.3. Euler Circuit 10

4. Strings 11
- 4.1. Z function 11
- 4.2. Prefix function 11
- 4.3. Suffix Array 11
- 4.4. Aho-Corasick automata 12
- 4.5. Manacher 13
- 4.6. Hash 13

5. Algebra and friends 14
- 5.1. Fourier Fast Transform (FFT) 14
- 5.2. Numeric Theoretic Transform (NTT) 14
- 5.3. extended gcd 15
- 5.4. Xor Basis 15

6. Combinatorics 17
- 6.1. Properties 17
  - 6.1.1. n Choose k 17
  - 6.1.2. Stirling God 17
  - 6.1.3. Formal Power Series 17
  - 6.1.4. Modular Arithmetic 18
  - 6.1.5. Chinese Remainder Theorem 18
  - 6.1.6. Lagrange Interpolation 18
  - 6.1.7. Catalan numbers 18
  - 6.1.8. Fibonacci 18
- 6.2. Techniques 18

7. Geometry 18
- 7.1. geomTemplate 18
- 7.2. Convex Hull (2D) 19

8. Cosas random 19
- 8.1. Calcular $\lfloor log_2 x \rfloor$ 19
- 8.2. SOS DP 19
- 8.3. My tipsNtricks stuff 19
- 8.4. template 20
- 8.5. Agregar a futuro 21

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/
*/

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;
list<Node*> to_erase; // guarda todo lo que se debe borrar;
Node* build( const vector<Data> &arr, int l=0,int r=-1 ) {
        if ( r==-1 ) t_sz=r=arr.size()-1;
        Node *nd = new Node(); to_erase.push_back(nd);
        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 = new Node(); to_erase.push_back(neo);
        if ( l >= r ) neo->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;
}
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={};
}
```

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 );
}
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. Dinitz

```
/*
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 ) {
        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;
}
```

3.2. 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;
}
```

3.3. Euler Circuit

Nota: Mi impl puede no funcionar bien para no dirigidos. Lo pondre en los todos.

```
/*
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;
}
```

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;
  }
};
```

5. Algebra and friends

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. 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;
    }
}
```

5.2. 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 = 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);
    }
}
```

5.3. 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;
}
```

5.4. 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;
    }

};
```

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.

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

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

$$\left\{{n \atop k}\right\} \equiv \binom{z}{w}(mod2)$$

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: I need some template for its operations (pending). 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 $\binom{n}{k}$.

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

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

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

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

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

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

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

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==pt);}
};
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

Tested in https://codeforces.com/problemset/problem/1017/E
*/

vector<Pt> convex_hull(Pt *arr,int n){
        sort(arr,arr+n);
        vector<Pt> up,dwn;
        rep(i,0,n){
                Pt &ac=arr[i]; int pz=up.size(),dz=dwn.size();
                while(pz>1&& (ac-up[pz-1])%(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&& (ac-dwn[dz-1])%(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;
}
```

8. Cosas random

8.1. 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

8.2. SOS DP

Calculate $S[x] = \sum_{y \subseteq x} a[y]$ for all $0 \leq x < 2^N$

```
rep(x,0,1<<N)S[x]=a[x];
rep(st,0,N)rep(x,0,1<<N)if((x>>st)&1){
        S[x]+=S[x^(1<<st)];
}
```

8.3. My tipsNtricks stuff

```
/*
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);
}
```

8.4. template

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

Autor: Oscar Vargas Pabon
Fecha:

*/

#include <bits/stdc++.h>

typedef long long lint;

using namespace std;

#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;
}
```

8.5. Agregar a futuro

- Sparse Table (idempotent/D&C)
- Cribe with functions
- Number theoretic stuff ($\mathbb{Z}_n$)
- DP optimization (Knuth, D&C, Lambda, CHT)
- optimal $\langle O(n), O(1)\rangle$ online range-minimum queries on a static array (with ideas of the offline one in $O((n+q)\alpha(n))$
- Operations on Formal Power Series
- geometry, geometry, geometry. Add sphere geometry (I studied that a bit)
- Sparse Segment Tree with Lazy upd
- some more floor sums (in honnor of Aladdin)
- Fix my euler circuit issue on undirected graphs
- test my persistent segment tree with lazy evaluation and add it
- study more number theory (I havent fully understood mobius and dirichlet convolution)
- study more combinatorics and add what its missing (like the p-analogs????)
- test my Manacher algo impl