Índice

1. Setup

Template corto

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forr(i,a,b)  for(int i = int(a); i < int(b); i++)
#define all(v)       begin(v), end(v)
#define mp(a,b)      make_pair(a,b)
#define pb           push_back

int main (int argc, char** argv) { if (argc == 2) freopen("input", "r
    ", stdin);



   return 0;
}
```

correr_interactivo.sh: Compilar y ejecutar $1

```
clear && make -s $1 && ./$1
```

Makefile

```
CC = g++
CPPFLAGS = -Wall -g \
-fsanitize=undefined -fsanitize=bounds \
-std=c++17 -O0
```

correr_archivo.sh: Compilar y ejecutar $1 con el input $2

```
clear && make -s $1 && ./$1 $2
```

compilar.sh: Compilar $1 y mostrar primeras $2 lineas de error

```
clear && make -s $1 2>&1 | head -$2
```

Template completo

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forall(it,v) for (auto it = begin(v); it != end(v); it++)
#define forr(i,a,b)  for(int i = int(a); i < int(b); i++)
#define forn(i,n)    forr(i,0,n)
#define all(v)       begin(v), end(v)
#define mp(a,b)      make_pair(a,b)
#define pb           push_back
#define fst          first
#define snd          second
#define endl         '\n'
#define dprint(x)    cerr << #x << " = " << (x) << endl
#define raya         cerr << "================" << endl
#define templT       template <class T>
#define templAB      template <class A, class B>
```

```
templAB ostream& operator << (ostream& o, pair<A,B>& p) { return o <<
    ↪ p.fst << " " << p.snd; }
templT  ostream& operator << (ostream& o, vector<T>& v) { forall(it,v
    ↪ ) { o << *it << " "; } return o; }
using ll = long long;

int main (int argc, char** argv) { ios::sync_with_stdio(0); cin.tie
    ↪ (0); cout.tie(0); if (argc == 2) freopen("input", "r", stdin);



   return 0;
}
```

2. STL

2.1. Vector

2.1.1. Busqueda binaria (lower_bound)

Primer igual

```
templT int primer_igual (vector<T>& arr, T x) {
    auto it = lower_bound(all(arr), x);
    if (it == arr.end() || *it != x) return -1;
    return it - arr.begin();
}
```

Último igual

```
templT int ultimo_igual (vector<T>& arr, T x) {
    if (arr.begin() == arr.end()) return -1;
    auto it = prev(upper_bound(all(arr), x));
    if (*it != x) return -1;
    return it - arr.begin();
}
```

Primer mayor

```
templT int primer_mayor (vector<T>& arr, T x) {
    auto it = upper_bound(all(arr), x);
    if (it == arr.end()) return -1;
    return it - arr.begin();
}
```

Último menor

```
templT int ultimo_menor (vector<T>& arr, T x) {
    if (arr.begin() == arr.end()) return -1;
    auto it = prev(lower_bound(all(arr), x));
    if (*it >=) return -1;
    return it - arr.begin();
}
```

2.1.2. Operaciones de conjuntos y modificación

Funciones que modifican rangos

Función	Params	Ejemplo
copy	first last result	`B.resize(A.size()); copy(all(A), B)`
fill	first last val	`memo.resize(MAXN); fill(all(memo), -1)`
rotate	first middle last	`rotate(begin(A), begin(A) + 3, end(A));`

Operaciones de conjuntos con vectors ordenados (lineal)

```
// Siempre hacer resize al final asi:

vector<int> A = { 5, 10, 15, 20, 25};
vector<int> B = {10, 20, 30, 40, 50};

vector<int> U(A.size() + B.size());

auto it = set_union(all(A), all(B), begin(U));

U.resize(it - U.begin());
```

Función	Descripción
set_union	Unión
set_intersection	Intersección
set_difference	Elementos que están en el primero y no en el segundo
set_symmetric_difference	Elementos que están en uno pero no los dos (como el xor)

2.2. Set indexado (policy based)

Set indexado

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace  __gnu_pbds;

templT struct SetIndexado {
    tree<
        T, null_type, less<T>,
        rb_tree_tag, tree_order_statistics_node_update
    > s;
    void add  (T   x) { s.insert(x); }
    int  idx  (T   x) { return  s.order_of_key(x); }
    bool has  (T   x) { return  s.find(x) != ms.end(); }
    T    ith  (int i) { return *s.find_by_order(i); }
};
```

Multiset indexado

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

```
using namespace __gnu_pbds;

templT struct MultisetIndexado {
    int t = 0; tree<
        pair<T, int>, null_type, less<pair<T, int>>,
        rb_tree_tag, tree_order_statistics_node_update
    > ms;
    void add  (T   x) { ms.insert(mp(x, t++)); }
    int  nle  (T   x) { return ms.order_of_key(mp(x, -1)); }
    int  nleq (T   x) { return ms.order_of_key(mp(x, INT_MAX)); }
    int  cnt  (T   x) { return nleq(x) - nle(x); }
    T    ith  (int i) { return (*ms.find_by_order(i)).fst; }
};
```

2.3. Truquitos con la STL

2.3.1. Compresión de coordenadas

Para números enteros (con lower_bound)

```
// Obtener valor original con D[A[i]]
vector<ll> CompCoordenadas (vector<ll>& A) {
   int N = A.size();
   vector<ll> D = A;
   sort(all(D));
   D.resize(unique(all(D)) - D.begin());
   forn(i, N) A[i] = lower_bound(all(D), A[i]) - D.begin();
   return D;
}
```

Versión genérica (con map)

```
templT map<T, int> CompCoordenadas (vector<T>& A) {
   map<T, int> ord;
   int n = 0;
   for (auto  v :  A)  ord[v];
   for (auto& e : ord) e.snd = n++;
   return ord;
}
```

2.3.2. Intervalos consecutivos (simular cortar un palito son set)

```
struct IntervalosConsecutivos {
    set<int> I;
    map<int, int> L;
    IntervalosConsecutivos (int i, int j) {
        I.insert(i);
        I.insert(j);
        L[j - i]++;
    }
    void cortar (int k) {
        int i = *prev(I.lower_bound(k));
        int j = *(I.lower_bound(k));
        L[j - i]--;
        if (L[j - i] == 0) L.erase(j - i);
        L[k - i]++;
        L[j - k]++;
        I.insert(k);
    }
    int max_intervalo () {
        return (*L.rbegin()).fst;
    }
};
```

3. Range queries

3.1. Range queries comunes

3.1.1. Suma estático (prefix + diff arrays)

Range sum (prefix array)

```
templT vector<T> prefix_array (vector<T>& A) {
    vector<T> P(A.size());
    P[0] = A[0];
    forn(i, P.size() - 1) P[i+1] = P[i] + A[i+1];
    return P;
}

// Retorna A[i] + ... +  A[j]
templT T query_prefix_array (vector<T>& P, int i, int j) {
    T res = P[j];
    if (i > 0) res -= P[i-1];
    return res;
}
```

Range update (diff array)

```
templT vector<T> diff_array (vector<T>& A) {
    vector<T> D(A.size());
    D[0] = A[0];
    forn(i, D.size() - 1) D[i+1] = A[i+1] - A[i];
    return D;
}

// Aplica +x en A[i] ... A[j]
templT void update_diff_array (vector<T>& D, int i, unsigned j, T x)
    ↪ {
    D[i] += x;
    if (j + 1 < D.size()) D[j+1] -= x;
}
```

3.1.2. Suma dinámico (fenwick tree)

Range sum point set

```
using FT = ll;
using Fenwick = unordered_map<int, FT>;
FT FT_prefix (Fenwick& A, int i) {
    FT res = 0;
    for (int j = i; j >= 0; j = j & (j + 1), j--) res += A[j];
    return res;
}
void FT_add (Fenwick& A, int N, int i, FT x) {
    for (; i < N; i = i | (i + 1)) A[i] += x;
}
FT FT_sum (Fenwick& A, int i, int j) {
    return FT_prefix(A, j) - FT_prefix(A, i - 1);
}
void FT_set (Fenwick& A, int N, int i, FT x) {
    FT_add(A, N, i, - FT_sum(A, i, i));
    FT_add(A, N, i, + x);
}
```

Range add point get

```
using FT = ll;
using Fenwick = unordered_map<int, FT>;
FT FT_prefix (Fenwick& A, int i) {
    FT res = 0;
    for (int j = i; j >= 0; j = j & (j + 1), j--) res += A[j];
    return res;
}
void FT_update (Fenwick& A, int N, int i, FT x) {
    for (; i < N; i = i | (i + 1)) A[i] += x;
}
FT FT_get (Fenwick& A, int i) {
    return FT_prefix(A, i);
}
void FT_add (Fenwick& A, int N, int i, int j, FT x) {
    FT_update(A, N, i, x);
    FT_update(A, N, j+1, -x);
}
```

3.1.3. Range minimum query (RMQ) (sparse table + segment tree)

RMQ estático 1D (sparse table)

```
using ST = int;
using SparseT = vector<vector<ST>>;
SparseT ST_build (vector<ST>& A, int N) {
    SparseT res(20, vector<ST>(N));
    res[0] = A;
    forr(w, 1, 20) forn(i, N - (1 << w) + 1)
        res[w][i] = min(res[w - 1][i], res[w - 1][i + (1 << (w - 1))
            ↪ ]);
    return res;
}
ST ST_rmq (SparseT& S, int i, int j) {
    int w = 63 - __builtin_clzll(j - i + 1);
    return min(S[w][i], S[w][j - (1 << w) + 1]);
}
```

RMQ + point set (segment tree)

3.2. Segment tree point set

Template

```
struct STNode {
   // Completar
};

STNode operator * (STNode a, STNode b) {
   // Completar
}

const STNode ST_ID = {
   // Completar
}

using STree = vector<STNode>;
STree segtree_build (STree& hojas) {
   int N = hojas.size();
   STree S(N << 1);
   forn(i, N) S[i + N] = hojas[i];
   for (int i = N - 1; i; i--) S[i] = S[i << 1] * S[i << 1 | 1];
   return S;
}

STNode segtree_query (STree& S, int i, int j) {
   int N = S.size() >> 1;
   STNode res = ST_ID;
   int l = i + N;
   int r = j + N + 1;
   for (; l < r; l >>= 1, r >>= 1) {
      if (l & 1) res = res * S[l++];
      if (r & 1) res = res * S[--r];
   }
   return res;
}

void segtree_update (STree& S, int i, STNode x) {
   int N = S.size() >> 1;
```

```
   S[i += N] = x;
   for (; i > 1; i >>= 1) S[i >> 1] = S[i] * S[i ^ 1];
}
```

RMQ

```
struct STNode { int val; };
STNode operator * (STNode a, STNode b) { return (a.val < b.val) ? a :
    ↪ b; }
const STNode ST_ID = { INT_MAX };

vector<int> A = { ... };
vector<STNode> hojas(A.size());
transform(all(A), begin(hojas), [](int x) { return (STNode){x}; });
STree T = segtree_build(hojas);
```

XOR

```
struct STNode { int val; };

STNode operator * (STNode a, STNode b) { return { a.val ^ b.val }; }

const STNode ST_ID = { 0 };

vector<int> A = { ... };
vector<STNode> hojas(A.size());
transform(all(A), begin(hojas), [](int x) { return (STNode){x}; });
STree T = segtree_build(hojas);
```

4. Grafos

4.1. Toposort de un DAG

```
using AdjList = vector<vector<int>>;

vector<int> Toposort (AdjList& G) {
    int N = G.size();
    vector<int> indegree(N), res;
    forn(u, N) for (int v : G[u]) indegree[v]++;
    // Elegir crierio de priorizacion cambiando el orden en el que se
        ↪ sacan
    // (por defecto el menor)
    using Bag = priority_queue<int, vector<int>, greater<int>>;
    Bag bag;
    forn(u, N) if(indegree[u] == 0) bag.push(u);
    while (bag.size()) {
        int u = bag.top();
        bag.pop();
        res.push_back(u);
        for (int v : G[u]) {
            indegree[v]--;
            if (indegree[v] == 0) bag.push(v);
        }
    }
    return res;
}
```

4.2. DAG condensado

```
using AdjList = vector<vector<int>>;

AdjList DAGCondensado (AdjList& G) {
   int N = G.size();
   vector<bool> visitado(N);

   vector<int> orden;
   function<void(int)> get_orden = [&](int u) -> void {
      visitado[u] = true;
      for (int v : G[u]) if (!visitado[v]) get_orden(v);
      orden.pb(u);
   };
   forn(u, N) if (!visitado[u]) get_orden(u);
   reverse(all(orden));

   AdjList T(N);
   forn(u, N) for (int v : G[u]) T[v].pb(u);

   vector<int> comp, raiz(N), raices;
   function<void(int)> extraer_comp = [&](int u) -> void {
      visitado[u] = true;
      comp.pb(u);
      for (int v : T[u]) if (!visitado[v]) extraer_comp(v);
   };

   visitado.assign(N, false);
   for (int u : orden) if (!visitado[u]) {
      extraer_comp(u);
      int r = comp.front();
      for (int v : comp) raiz[v] = r;
      raices.pb(r);
      comp.clear();
   }

   // Opcion 1: hacer compresion de coordenadas: O(nlogn) lento
   int c = 0;
   map<int, int> coords;
   for (int  r :  raices) coords[r];
   for (auto& e : coords) e.snd = c++;
   AdjList SCC(raices.size());
   forn(u, N) for (int v : G[u]) {
      int ru = coords[raiz[u]], rv = coords[raiz[v]];
```

```
        if (ru != rv) SCC[ru].pb(rv);
    }

    return SCC;

    // Opcion 2: no hacer compresion y devolver raices (rapido)
    // AdjList SCC(N);
    // forn(u, N) for (auto [v, w] : G[u]) {
        // int ru = raiz[u], rv = raiz[v];
        // if (ru != rv) SCC[ru].pb({rv, w});
        // else (RC[ru]) += R(w);
    // }
}
```

4.3. Bipartite check

```
using AdjList = vector<vector<int>>;

bool EsBipartito (AdjList& G) {
    vector<int> color(G.size(), -1);
    color[0] = 0;
    queue<int> bag;
    for (bag.push(0); bag.size();) {
        int u = bag.front();
        bag.pop();
        for (int v : G[u]) {
            if (color[u] == color[v]) return false;
            if (color[v] == -1) {
                color[v] = 1 - color[u];
                bag.push(v);
            }
        }
    }
    return true;
}
```

4.4. Encontrar puentes y articulaciones

```
using AdjList = vector<vector<int>>;
using Edge = pair<int, int>;
pair<vector<Edge>, vector<int>> GetPuentesArticulaciones (AdjList& G)
    ↪ {
    int N = G.size(), time = 0;
    vector<bool> visitado(N);
    vector<int> tin(N, -1), tlow(N, -1), articulaciones;
    vector<Edge> puentes;
    function<void(int, int)> dfs = [&](int u, int p) -> void {
        visitado[u] = true;
        tin[u] = tlow[u] = time++;
        int hijos = 0;
        for (int v : G[u]) {
            if (v == p) continue;
            if (visitado[v]) tlow[u] = min(tlow[u], tin[v]);
            else {
                dfs(v, u);
                hijos++;
                tlow[u] = min(tlow[u], tlow[v]);
                if (tlow[v] >  tin[u]) puentes.pb({u,v});
                if (tlow[v] >= tin[u] && p != -1) articulaciones.pb(u);
            }
        }
        if (p == -1 && hijos > 1) articulaciones.pb(u);
    };
    forn(r, N) if (!visitado[r]) dfs(r, -1);
    return mp(puentes, articulaciones);
}
```

4.5. Menores caminos

Dijkstra

```
struct Hedge { ll weight; int node; };
bool operator < (const Hedge& a, const Hedge& b) { return a.weight >
    ↪ b.weight; }
using AdjList = vector<vector<Hedge>>;

void Dijkstra (AdjList& G, int s, vector<ll>& dist, vector<int>&
    ↪ parent) {
    int N = G.size();
    dist.assign(N, LLONG_MAX);
    dist[s] = 0;
    parent.assign(N, -1);
    parent[s] = s;
    priority_queue<Hedge> bag;
    for (bag.push({0, s}); bag.size();) {
        auto [d, u] = bag.top();
        bag.pop();
        if (d > dist[u]) continue;
        for (auto [w, v] : G[u]) {
            ll relax = d + w;
            if (relax < dist[v]) {
                dist[v] = relax;
                parent[v] = u;
                bag.push({relax, v});
            }
        }
    }
}
```

Floyd-Warshall

```
templT using Matriz = vector<vector<T>>;
const ll INF = LLONG_MAX / 4;
```

```
void FloydWarshall (Matriz<ll>& D) {
   int N = D.size();
   forn(u, N) D[u][u] = 0;
   forn(k, N) forn(u, N) forn(v, N) if (D[u][k] < INF) if (D[k][v] <
      ↪ INF)
     D[u][v] = min(D[u][v], D[u][k] + D[k][v]);
   // Opcional: chequear ciclos negativos
   forn(u, N) forn(v, N) forn(k, N) if (D[u][k] < INF && D[k][k] < 0
      ↪ && D[k][v] < INF)
     D[u][v] = -INF;
}
```

5. Programacion Dinamica

5.1. LIS (Longest Increasing Subsequence)

Obtener largo del LIS

```
// Usa compresion de coordenadas y segtree point set RMQ (tomar el
   ↪ maximo)
int LIS (vector<int>& A) {
   int N = A.size();
   auto C = Compress(A);

   vector<STNode> hojas(N, {0});
   STree dp = segtree_build(hojas);

   segtree_update(dp, C[A[0]], {1});
   forr(i, 1, N) {
      int x = C[A[i]];
      int subres = 0;
      if (x-1 >= 0) subres = segtree_query(dp, 0, x-1).val;
      segtree_update(dp, x, {1 + subres});
   }

   return segtree_query(dp, 0, N - 1).val;
}
```

Construir LIS lexicograficamente menor

```
struct STNode { int len, idx, val, parent; };
bool operator < (STNode a, STNode b) {
   if (a.len != b.len) return a.len < b.len;
   return a.val > b.val;
}
STNode operator * (STNode a, STNode b) { return max(a,b); }
const STNode ST_ID = { -INT_MAX, -1, INT_MAX, -1 };

vector<int> LIS (vector<int>& A) {
   int N = A.size();
   auto C = Compress(A);

   STNode def = {0, -1, INT_MAX, -1};
   vector<STNode> hojas(N, def);
   STree dp = segtree_build(hojas);

   vector<STNode> res(N);
   res[0] = {1, 0, A[0], -1};
   segtree_update(dp, C[A[0]], {1, 0, A[0], -1});
   forr(i, 1, N) {
      int x = C[A[i]];
      STNode subres = def;
      if (x-1 >= 0) subres = segtree_query(dp, 0, x-1);
      STNode r = {1 + subres.len, i, A[i], subres.idx};
      segtree_update(dp, x, r);
      res[i] = r;
   }

   vector<int> path;
   STNode best = *max_element(all(res));
   STNode x;
   for (x = best; x.parent != -1; x = res[x.parent]) path.pb(x.idx);
   path.pb(x.idx);
   reverse(all(path));

   return path;
}
```

LIS en arbol (largo del LIS de raiz a cada nodo)

```
// Usa compresion de coordenadas y segtree point set RMQ (tomar el
   ↪ maximo)
using AdjList = vector<vector<int>>;
vector<int> LIS (AdjList& G, vector<int>& valor_nodo, int root = 0) {
   int N = valor_nodo.size();
   auto C = Compress(valor_nodo);

   STNode def = { 0 };
   vector<STNode> hojas(N, def);
   STree dp = segtree_build(hojas);

   vector<int> res(N);

   segtree_update(dp, C[valor_nodo[root]], {1});
   function<void(int)> dfs = [&](int u) {
      int x = C[valor_nodo[u]];
      int old = segtree_query(dp, x, x).val;
      int subres = {0};
      if (x-1 >= 0) subres = segtree_query(dp, 0, x-1).val;
      segtree_update(dp, x, {1 + subres});
```

```
        res[u] = segtree_query(dp, 0, N-1).val;
        for (int v : G[u]) dfs(v);
        segtree_update(dp, x, {old});
    };
    dfs(root);

    return res;
}
```

6. Matemática

6.1. Aritmética

Techo de la división

```
#define ceildiv(a,b) ((a + b - 1) / b)
```

Piso de la raiz cuadrada

```
using ll = long long;

ll isqrt (ll x) {
    ll s = 0;
    for (ll k = 1 << 30; k; k >>= 1)
        if ((s+k) * (s+k) <= x) s += k;
    return s;
}
```

Piso del log2

```
#define log2fl(x) (x ? 63 - __builtin_clzll(x) : -1)
```

Aritmética en Zp

```
const ll mod = 1e9 + 7;

ll resta_mod (ll a, ll b) { return (a - b + mod) % mod; }

ll pow_mod (ll x, ll n) {
    ll res = 0;
    while (n) {
        if (n % 2) res = res * x % mod;
        n /= 2;
        x = x * x % mod;
    } return res;
}

ll div_mod (ll a, ll b) { return a * pow_mod(b, mod - 2) % mod; }
```

6.2. Teoria de numeros

Criba

```
struct Criba {
    bool c[1000001]; vector<int> p;
    Criba () {
        p.reserve(1<<16);
        for (int i = 2; i <= 1000000; i++) if (!c[i]) {
            p.pb(i);
            for (int j = 2; i*j <= 1000000; j++) c[i*j] = 1;
        }
    }
    bool isprime (int x) {
        for (int i = 0, d = p[i]; d*d <= x; d = p[++i])
            if (!(x % d)) return false;
        return x >= 2;
    }
};
```

Phollards Rho

```
ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}


//COMPILAR CON G++20
ll mulmod(ll a, ll b, ll m) {
 return ll(__int128(a) * b % m);
}

ll expmod (ll b, ll e, ll m){//O(log b)
        if(!e) return 1;
        ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
        return e%2? mulmod(b,q,m) : q;
}

bool es_primo_prob (ll n, int a)
{
        if (n == a) return true;
        ll s = 0,d = n-1;
        while (d % 2 == 0) s++,d/=2;

        ll x = expmod(a,d,n);
        if ((x == 1) || (x+1 == n)) return true;

        forn (i, s-1){
                x = mulmod(x, x, n);
                if (x == 1) return false;
                if (x+1 == n) return true;
        }
        return false;
}
```

```
bool rabin (ll n){ //devuelve true si n es primo
        if (n == 1)     return false;
        const int ar[] = {2,3,5,7,11,13,17,19,23};
        forn (j,9)
                if (!es_primo_prob(n,ar[j]))
                        return false;
        return true;
}

ll rho(ll n){
    if( (n & 1) == 0 ) return 2;
    ll x = 2 , y = 2 , d = 1;
    ll c = rand() % n + 1;
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = gcd( x - y , n );
        else d = gcd( y - x , n );
    }
    return d==n? rho(n):d;
}

map<ll,ll> prim;
void factRho (ll n){ //O (lg n)^3. un solo numero
        if (n == 1) return;
        if (rabin(n)){
                prim[n]++;
                return;
        }
        ll factor = rho(n);
        factRho(factor);
        factRho(n/factor);
}
```

6.3. Geometria

Template geometria

```
using flt = long double;
const flt EPS = 1e-9;
bool  flt_leq (flt a, flt b) { return a < b + EPS; }
bool  flt_eq  (flt a, flt b) { return -EPS <= a - b && a - b <= EPS;
    ↪ }

using Sca = long long;
struct Vec { Sca x, y; };
Vec  operator + (Vec a, Vec b) { return { a.x + b.x, a.y + b.y }; }
Vec  operator - (Vec a, Vec b) { return { a.x - b.x, a.y - b.y }; }
Sca  operator * (Vec a, Vec b) { return a.x * b.x + a.y * b.y; }
Sca  operator ^ (Vec a, Vec b) { return a.x * b.y + a.y * b.x; }
bool operator < (Vec a, Vec b) { return (a.x != b.x) ? (a.x < b.x) :
    ↪ (a.y < b.y); }
ostream& operator << (ostream &o, Vec& p) { auto x = mp(p.x, p.y);
    ↪ return o << x; }

Sca norma2 (Vec p) { return p.x * p.x + p.y * p.y; }

struct pto{
        ll x, y; int t;
        pto(ll x=0, ll y=0, int t = -1):x(x),y(y), t(t){}
        pto operator-(pto a){return pto(x-a.x, y-a.y);}
        ll operator*(pto a){return x*a.x+y*a.y;}
        ll operator^(pto a){return x*a.y-y*a.x;}
        bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
        bool operator<(const pto &a) const{return x<a.x || (x==a.x &&
            ↪  y<a.y);}
  bool operator==(pto a){return x==a.x && y==a.y;}
};
//stores convex hull of P in S, CCW order
//left must return >=0 to delete collinear points!
void CH(vector<pto>& P, vector<pto> &S){
        S.clear();
        sort(P.begin(), P.end());//first x, then y
        forn(i, sz(P)){//lower hull
                while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i]))
                    ↪  S.pop_back();
                S.pb(P[i]);
        }
        S.pop_back();
        int k=sz(S);
        dforn(i, sz(P)){//upper hull
                while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i
                    ↪ ])) S.pop_back();
                S.pb(P[i]);
        }
        S.pop_back();
}
```

7. Estructuras locas

7.1. Disjoint set union

```
struct DSU {
    vector<int> p, w; int nc;
    DSU (int n) {
        nc = n, p.resize(n), w.resize(n);
        forn(i,n) p[i] = i, w[i] = 1;
```

```
    }
    int get (int x) { return p[x] == x ? x : p[x] = get(p[x]); }
    void join (int x, int y) {
        x = get(x), y = get(y);
        if (x == y) return;
        if (w[x] > w[y]) swap(x,y);
        p[x] = y, w[y] += w[x];
    }
    bool existe_camino (int x, int y) { return get(x) == get(y); }
};
```

7.2. Binary trie

```
struct BinaryTrieVertex { vector<int> next = {-1, -1}; };

using BinaryTrie = vector<BinaryTrieVertex>;

void binary_trie_add (BinaryTrie& trie, int x) {
    int v = 0;
    for (int i = 31; i >= 0; i--) {
        bool b = (x & (1 << i)) > 0;
        if (trie[v].next[b] == -1) {
            trie[v].next[b] = trie.size();
            trie.emplace_back();
        }
        v = trie[v].next[b];
    }
}

int binary_trie_max_xor (BinaryTrie& trie, int x) {
    int v = 0, res = 0;
    for (int i = 31; i >= 0; i--) {
        bool b = (x & (1 << i)) > 0;
        if (trie[v].next[!b] != -1) {
            v = trie[v].next[!b];
            if (!b) res |= (1 << i);
        }
        else {
            v = trie[v].next[ b];
            if ( b) res |= (1 << i);
        }
    } return res;
}

// Inicializar asi:
BinaryTrie trie(1);
```

8. Sin categorizar

8.1. Búsqueda binaria sobre un predicado

```
int primer_true (int i, int j, function<bool(int)> P, int def) {
   while (j - i > 1) {
      int m = i + ((j - i) >> 1);
      P(m) ? j = m : i = m;
   }
   if (P(i)) return i;
   if (P(j)) return j;
   return def;
}

int ultimo_false (int i, int j, function<bool(int)> P, int def) {
   while (j - i > 1) {
      int m = i + ((j - i) >> 1);
      P(m) ? j = m : i = m;
   }
   if (!P(j)) return j;
   if (!P(i)) return i;
   return def;
}
```

8.2. Enumerar subconjuntos de un conjuto con bitmask

```
// Imprimir representaciones en binario de todos los numeros "[0,
   ↪ ..., 2^N-1]"
forn(mask, (1 << N)) {
    forn(i, N) cout << "01"[(mask & (1 << i)) > 0] << "\0\n"[i == N
        ↪ -1];
}

// Iterar por los bits de cada subconjunto
forn(mask, (1 << N)) {
    forn(i, N) {
        bool on = (mask & (1 << i)) > 0;
        if (on) { ... }
        else { ... }
    }
}
```

8.3. Hashing Rabin Karp

```
using ll = long long;

const ll primo = 27, MAX_PRIME_POW = 1e6;

ll prime_pow[MAX_PRIME_POW];
void get_prime_pow () {
    prime_pow[0] = 1;
    forn(i, MAX_PRIME_POW) prime_pow[i+1] = prime_pow[i] * primo %
        ↪ mod;
}
```

```
vector<ll> get_rolling_hash (string& s) {
    vector<ll> rh(s.size() + 1);
    rh[0] = 0;
    // Ojo: es 'A' o 'a' ???
    forn(i, s.size()) rh[i+1] = (rh[i] * primo % mod + s[i] - 'A') %
        ↪ mod;
    return rh;
}

ll hash_range_query (vector<ll>& rh, int i, int j) {
    j++;
    return (rh[j] - (rh[i] * prime_pow[j - i] % mod) + mod) % mod;
}
```

8.4. Lowest common ancestor (LCA)

```
#define log2fl(x) (x ? 63 - __builtin_clzll(x) : -1)
using AdjList = vector<vector<int>>;

struct LCA {
    AdjList& G; int N, R; // Grafo (ROOTEADO), #vertices y raiz
    int M; vector<int> e, f, d; AdjList st;
    void dfs (int u, int de = 0) {
        d[u] = de, f[u] = e.size(), e.pb(u);
        for (int v : G[u]) dfs(v,de+1), e.pb(u);
    }
    int op (int a, int b) {
        if (a == -1) return b;
        if (b == -1) return a;
        return d[a] < d[b] ? a : b;
    }
    void make () {
        f.resize(N), d.resize(N), dfs(R), M = e.size();
        st.resize(20, vector<int>(M));
        st[0] = e; scn(w,1,19) scn(i,0,M - (1 << w))
            st[w][i] = op(st[w-1][i], st[w-1][i + (1 << (w-1))]);
    }
    int q (int u, int v) {
        int i = f[u], j = f[v];
        if (i > j) swap(i,j);
        int w = log2fl(j - i + 1);
        return op(st[w][i], st[w][j - (1 << w) + 1]);
    }
    int di (int u, int v) {
        int c = q(u,v);
        return d[u] + d[v] - 2*d[c];
    }
};

bool visited[500001]; void rootear (int u) {
    visited[u] = 1;
    for (int v : grafo_original[u]) if (!visited[v]) {
        grafo_rooteado[u].pb(v);
        rootear(v);
    }
}


// Usar asi:
rootear(r);
LCA lca = {grafo_rooteado, N, r}; lca.make();
```

8.5. Euler tour

```
typedef vector<vector<int>> adj;
typedef vector<vector<pair<int,i64>>> wadj;

struct ETour {
    adj& G; int N, R;
    vector<int> t, f, d;
    void dfs (int u, int de = 0) {
        d[u] = de, f[u] = t.size(), t.pb(u);
        for (int v : G[u]) { dfs(v,de+1); t.pb(u); }
    }
    void make () { f.resize(N), d.resize(N), dfs(R); }
};

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

    adj G; int N; cin >> N; G.resize(N);
    scn(u,1,N-1) {
        int p; cin >> p; p--;
        G[p].pb(u);
    }

    ETour et = {G, N, 0}; et.make();
    forall(v,et.t) { cout << *v + 1 << " "; } cout << endl;
    forall(v,et.t) { cout << et.d[*v] << " "; } cout << endl;
    forall(v,et.t) { cout << et.f[*v] << " "; } cout << endl;

    return 0;
}
/*
1 3 2 3 5 3 1 4 1
0 1 2 1 2 1 0 1 0
0 1 2 1 4 1 0 7 0
*/
```

9. Brainstorming

- Graficar como puntos/grafos

- Pensarlo al revez
- ¿Que propiedades debe cumplir una solución?
- Si existe una solución, ¿existe otra más simple?
- ¿Hay elecciones independientes?
- ¿El proceso es parecido a un algoritmo conocido?
- Si se busca calcular $f(x)$ para todo x, calcular cuánto contribuye x a $f(y)$ para los otros y
- Definiciones e identidades: *¿que significa* que un array sea palindromo? (ejemplo)