Deja de dudar Leonel *grosería*

Universidad Mayor de San Simón

September 5, 2025

Contents

1 Strings

1.1 String A is a rotation of B?

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

bool areRotations(string &s1, string &s2) {
        s1 += s1;
        return s1.find(s2) != string::npos;
}
```

1.2 Hashing

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

const int mod = 1e9 + 7;
const int p = 31; // para mayusculas y minusculas 53
int getHash(string s) {
  int ans = 0;
  int n = s.size();
  int base = 1;
  for (int i = 0; i < n; i++) {
    ans += (s[i]-'a'+1) * base % mod;
    ans %= ans;
    base = base * p % mod;
  }
  return ans;
}
```

1.3 KMP

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

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

1.4 Prefix Function

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

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

```
            j++;
        pi[i] = j;
    }
    return pi;
}
```

1.5 Rabin Karp

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

vector<int> rabinKarp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurrences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurrences.push_back(i);
    }
    return occurrences;
}
```

1.6 Z-Function

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

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

2 Graph Algorithms

2.1 Bellman Ford

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

using ll = long long;
```

```
vector<int> bellmanFord(int V, vector<vector<int>>& edges,
    int src) {
        vector<int> dist(V, 1e8);
        dist[src] = 0;
        for (int i = 0; i < V; i++) {

                for (vector<int> edge : edges) {
                        int u = edge[0];
                        int v = edge[1];
                        int wt = edge[2];
                        if (dist[u] != 1e8 && dist[u] + wt <
                             dist[v]) {
                if(i == V - 1)
                    return {-1};
                dist[v] = dist[u] + wt;
            }
                }
        }

    return dist;
}
```

2.2 BFS

```
#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int INF = 1e9;
const double EPS = 1e-9;

void bfs(int v, const vector<vector<int>>& g, vector<bool>&
    visi) {
    deque<int> cola;
    cola.push_back(v);
    visi[v] = true;
    while (!cola.empty()) {
        int v = cola.front();
        cola.pop_front();
        for (int vec : g[v]) {
            if (!visi[vec]) {
                cola.push_back(vec);
                visi[vec] = true;
            }
        }
    }
}
```

2.3 DFS

```
#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int INF = 1e9;
const double EPS = 1e-9;

void dfs(int v, const vector<vector<int>>&g, vector<bool>&
    visi) {
    if (!g[v]) {
        visi[v] = true;
        for (int vec : g[v]) {
            dfs(vec, g, visi);
        }
    }
}
```

2.4 Disjoint Set

```
#include <bits/stdc++.h>

using namespace std;

using ll = long long;
```

```
const int INF = 1e9;
const double EPS = 1e-9;

struct ConjuntoDisjunto {
    vector<int> padre;
    vector<int> rango;

    ConjuntoDisjunto(int n) {
        padre.resize(n + 1);
        rango.resize(n + 1, 1);
        for (int i = 1; i <= n; ++i) {
            padre[i] = i;
        }
    }

    int encontrar(int v) {
        if (padre[v] != v) {
            padre[v] = encontrar(padre[v]);
        }
        return padre[v];
    }

    void unir(int u, int v) {
        int raizU = encontrar(u);
        int raizV = encontrar(v);

        if (raizU != raizV) {
            if (rango[raizU] < rango[raizV]) {
                padre[raizU] = raizV;
            } else if (rango[raizU] > rango[raizV]) {
                padre[raizV] = raizU;
            } else {
                padre[raizV] = raizU;
                rango[raizU]++;
            }
        }
    }
};
```

2.5 Flood Fill

```
#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int INF = 1e9;
const double EPS = 1e-9;

struct FloodFill {
    vector<vector<int>> tab;
    vector<vector<int>> visi;
    int n, m;

    void init(vector<vector<int>> &tablero) {
        tab = tablero;
        n = tab.size();
        m = tab[0].size();
        visi.assign(n, vector<int>(m, 0));
    }

    int floodfill(int x, int y) {
        if (x < 0 || y < 0 || x >= n || y >= m || visi[x][y]
             || tab[x][y] == 0)
            return 0;
        visi[x][y] = 1;
        int ret = 1;
        //int dir[2][4] = {{0, 0, 1, -1}, {1, -1, 0, 0}};
        int dir[2][8] = {{0, 0, 1, -1, 1, 1, -1, -1}, {1,
           -1, 0, 0, 1, -1, 1, -1}};
        for (int i = 0; i < 8; i++)
            ret += floodfill(x + dir[0][i], y + dir[1][i]);
        return ret;
    }
};
```

2.6 Is Bipartite?

```
int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v]) {
                if (side[u] == -1) {
                    side[u] = side[v] ^ 1;
                    q.push(u);
                } else {
                    is_bipartite &= side[u] != side[v];
                }
            }
        }
    }
}

cout << (is_bipartite ? "YES" : "NO") << endl;
```

2.7 Kruskal

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

using ll = long long;
//debe guardarse el grafo como (peso,v1,v2)
struct unionFind{
    vector<int> p;
    unionFind(int n) : p(n,-1){}
    int find(int x){
        if(p[x] == -1) return x;
        return p[x] = find(p[x]);
    }

    bool join(int x, int y){
        x = find(x);
        y = find(y);
        if(x==y) return 0;
        p[y]=x;
        return 1;
    }
};
ll kruskal(vector<pair<ll,pair<ll,ll>>> ed, int n){
    //n es nro de vertices
    sort(ed.begin(),ed.end());
    unionFind dsu(n);
    int cntAristas = 0;
    ll res = 0;
    for(auto e: ed){
        ll peso = e.first;
        int u = e.second.first;
        int v = e.second.second;
        if(dsu.join(u,v)){
            cntAristas++;
            res+=peso;
        }
        if(cntAristas == n-1){
            return res;
        }
    }
    if(cntAristas < n-1){
        return -1;
    }
    return res;
}
```

2.8 Lowest Common Ancestor

```
int n, l;
vector<vector<int>> adj;
```

```
int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

2.9 TopoSort

```
#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int INF = 1e9;
const double EPS = 1e-9;

int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
        }
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    reverse(ans.begin(), ans.end());
}
```

3 Data Structures

3.1 Fenwick Tree

```cpp
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

 struct BIT { // 1-indexed, your first element of the array is at index 1
    vector<ll> bit;
    ll n;
    BIT(int n) : bit(n+1), n(n) {}

    ll lsb(int i) { return i & -i; } // least significant bit

         void add(int i, ll x) {
        for (; i <= n; i += lsb(i)) bit[i] += x;
    }

    ll sum(int r) {
        ll res = 0;
        for (; r > 0; r -= lsb(r)) res += bit[r];
        return res;
    }

    ll sum(int l, int r) {
        return sum(r) - sum(l-1);
    }

    void set(int i, ll x) {
        add(i, x - sum(i, i));
    }
};
```

3.2 QuadTree

```cpp
#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int INF = 1e9;
const double EPS = 1e-9;

//QUADTREE
struct QuadTree{
    char valor;
    bool esHoja;
    QuadTree *izquierdaSuperior,*derechaSuperior,*izquierdaInferior, *derechaInferior;
    QuadTree(char valor1){
        valor=valor1;
        izquierdaSuperior=derechaSuperior=izquierdaInferior=derechaInferior=nullptr;
        if(valor=='p'){
            esHoja=false;
        }
        else{
            esHoja=true;
        }
    }

};

//CONSTRUCTOR
QuadTree* construir(string valor,long long &indice){
    if(indice>=valor.length()){
        return nullptr;
    }
    QuadTree *nodo = new QuadTree (valor[indice]);
    char letra = valor[indice];
    if(letra!='p'){
        nodo->esHoja = true;
        indice++;
    }
    else{
        indice++;
        nodo->izquierdaSuperior=construir(valor,indice);
        nodo->derechaSuperior =construir(valor,indice);
        nodo->izquierdaInferior=construir(valor,indice);
        nodo->derechaInferior=construir(valor,indice);
    }
    return nodo;
}
//METODO DE CONTAR NODOS CON CIERTA CONDICION
int contarCuadradosNegros(QuadTree *nodoReferencia,int
    tamano){
    if(nodoReferencia->esHoja == true){
        if(nodoReferencia->valor=='f'){
            return tamano*tamano;
        }
        else{
          return 0;
        }
    }
    long long medio = tamano/2;
    long long resultado = contarCuadradosNegros(
        nodoReferencia->izquierdaSuperior,medio)
    +contarCuadradosNegros(nodoReferencia->derechaSuperior,
        medio)+
    contarCuadradosNegros(nodoReferencia->izquierdaInferior,
        medio)+
    contarCuadradosNegros(nodoReferencia->derechaInferior,
        medio);
    return resultado;
}
//UNION DE QUADTREES
QuadTree* combinar(QuadTree *nodo1, QuadTree *nodo2){
    if(nodo1==nullptr){
        return nodo2;
    }
    if(nodo2==nullptr){
        return nodo1;
    }
    if(nodo1->esHoja==true && nodo2->esHoja==true ){
        if(nodo1->valor=='f' || nodo2->valor=='f'){
            return new QuadTree('f');
        }
        else{
            return new QuadTree('e');
        }
    }
    if(nodo1->esHoja==true){
        if(nodo1->valor=='f'){
            return new QuadTree('f');
        }
        else{
            return nodo2;
        }
    }
    if(nodo2->esHoja==true){
        if(nodo2->valor=='f'){
            return new QuadTree('f');
        }
        else{
            return nodo1;
        }
    }
    QuadTree *nuevoQuad = new QuadTree ('p');
    nuevoQuad->izquierdaSuperior=combinar(nodo1->
        izquierdaSuperior,nodo2->izquierdaSuperior);
    nuevoQuad->derechaSuperior=combinar(nodo1->
        derechaSuperior,nodo2->derechaSuperior);
    nuevoQuad->izquierdaInferior=combinar(nodo1->
        izquierdaInferior,nodo2->izquierdaInferior);
    nuevoQuad->derechaInferior=combinar(nodo1->
        derechaInferior,nodo2->derechaInferior);
    return nuevoQuad;
}
```

3.3 Segment Tree Max and Min

```
#include <bits/stdc++.h>
```

```
using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

struct segtreeMaxMin{
    //el maximo es el first y el minimo es el second
    int n;
    vector<pair<ll,ll>> tree;

    void init(int nn) {
        tree.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        tree.resize(size * 2);
    }

    void update(int i, int sl, int sr, int pos, ll diff) {
        if (sl <= pos && pos <= sr) {
            if (sl == sr) {
                tree[i] = {tree[i].first+diff,tree[i].second
                    +diff};
            } else {
                int mid = (sl + sr) / 2;
                update(i * 2 + 1, sl, mid, pos, diff);
                update(i * 2 + 2, mid + 1, sr, pos, diff);
                tree[i] = {max(tree[i * 2 + 1].first, tree[i
                     * 2 + 2].first),min(tree[i * 2 + 1].
                    second, tree[i * 2 + 2].second)};
            }
        }
    }

    void update(int pos, ll diff) {
        update(0, 0, n - 1, pos, diff);
    }

    pair<ll,ll> query(int i, int sl, int sr, int l, int r) {
        if (l <= sl && sr <= r) {
            return tree[i];
        } else if(sr < l || r < sl) {
            return{INT64_MIN,INT64_MAX};
        } else {
            int mid = (sl + sr) / 2;
            auto a = query(i * 2 + 1, sl, mid, l, r);
            auto b = query(i * 2 + 2, mid + 1, sr, l, r);
            return {max(a.first,b.first),min(a.second,b.
                second)};
        }
    }

    pair<ll,ll> query(int l, int r) {
        return query(0, 0, n - 1, l, r);
    }
};
```

3.4 Segment Tree XOR

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

// Example of a Segment tree of Xor
struct Node {
    ll a = 0;
};

Node e() {
    Node node;
    return node;
}

Node op(Node a, Node b) {
    Node node;
    node.a = a.a ^ b.a;
    return node;
}
// >>>>>>>> Implement

struct segtree {
    vector<Node> nodes;
    ll n;

    void init(int n) {
        auto a = vector<Node>(n, e());
        init(a);
    }

    void init(vector<Node>& initial) {
        nodes.clear();
        n = initial.size();
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        nodes.resize(size * 2);
        build(0, 0, n-1, initial);
    }

    void build(int i, int sl, int sr, vector<Node>& initial)
         {
        if (sl == sr) {
            nodes[i] = initial[sl];
        } else {
            ll mid = (sl + sr) >> 1;
            build(i*2+1, sl, mid, initial);
            build(i*2+2, mid+1,sr,initial);
            nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
        }
    }

    void update(int i, int sl, int sr, int pos, Node node) {
        if (sl <= pos && pos <= sr) {
            if (sl == sr) {
                nodes[i] = node;
            } else {
                int mid = (sl + sr) >> 1;
                update(i * 2 + 1, sl, mid, pos, node);
                update(i * 2 + 2, mid + 1, sr, pos, node);
                nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
            }
        }
    }

    void update(int pos, Node node) {
        update(0, 0, n - 1, pos, node);
    }

    Node query(int i, int sl, int sr, int l, int r) {
        if (l <= sl && sr <= r) {
            return nodes[i];
        } else if(sr < l || r < sl) {
            return e();
        } else {
            int mid = (sl + sr) / 2;
            auto a = query(i * 2 + 1, sl, mid, l, r);
            auto b = query(i * 2 + 2, mid + 1, sr, l, r);
            return op(a, b);
        }
    }

    Node query(int l, int r) {
        return query(0, 0, n - 1, l, r);
    }

    Node get(int i) {
        return query(i, i);
    }
};
```

4 Number Theory

4.1 Binary Pow

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

long long binpow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
//optimizado para numeros grandes
int binpow(int a, int b, int n) {
    int res = 1;
    a = a % n;
    while (b > 0) {
        if (b & 1)
            res = (res * a) % n;

        a = (a * a) % n;
        b >>= 1;
    }
    return res;
}
```

4.2 Binomial Coefficient

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

int binomial_coeff(int n, int k) {
   int res = 1;

   // Since C(n, k) = C(n, n-k)
   if (k > n - k) {
      k = n - k;
   }

   // Calculate value of
   // [n * (n-1) *---* (n-k+1)] / [k * (k-1) *----* 1]
   for (int i = 0; i < k; ++i) {
      res *= (n - i);
      res /= (i + 1);
   }

   return res;
}

//other version

const int ta = 1010;
ll bino[ta][ta];
void init()
{
    for(int i = 0; i < ta; i++)
        bino[i][0] = 1;
    for(int i = 1; i < ta; i++)
        for(int j = 1; j < ta; j++)
            bino[i][j] = (bino[i - 1][j]
                        + bino[i - 1][j-1]) % MOD;
}
```

4.3 Cribe

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;
```

```
vector<bool> sieve(int n) {
    vector<bool> is_prime(n + 1, true);
    is_prime[0] = is_prime[1] = false;

    for (int i = 2; i <= n; i++) {
        if (!is_prime[i]) continue;
        for (int u = 2 * i; u <= n; u += i) {
            is_prime[u] = false;
        }
    }

    return is_prime;
}
```

4.4 Divisors of a number

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

vector<int> findDivisors(int n) {
    vector<int> v;
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            v.push_back(i);
            int other = n / i;
            if (other != i) { // case i * i = n
                v.push_back(other);
            }
        }
    }
    return v;
}

long long SumOfDivisors(long long num) {
     long long total = 1;
     for (int i = 2; (long long)i * i <= num; i++) {
          if (num % i == 0) {
               int e = 0;
               do {
                    e++;
                    num /= i;
               } while (num % i == 0);

               long long sum = 0, pow = 1;
               do {
                    sum += pow;
                    pow *= i;
               } while (e-- > 0);
               total *= sum;
          }
     }
     if (num > 1) {
          total *= (1 + num);
     }
     return total;
}
```

4.5 Modular Exponentiation

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

//O(log(y))
int power(int x, int y, int MOD) {
     int res = 1;
     x = x % MOD;
     if (x == 0) return 0;
     while (y > 0){
          if (y & 1)
               res = (res*x) % MOD;
```

```
        y = y>>1;
        x = (x*x) % MOD;
    }
    return res;
}
```

4.6 Modular Inverse

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

int inv(int a) {
  return a <= 1 ? a : m - (long long)(m/a) * inv(m % a) % m;
}

//precalculando:
inv[1] = 1;
for(int a = 2; a < m; ++a){
    inv[a] = m - (long long)(m/a) * inv[m%a] % m;
}

//version ekisd
int invMod(int a, int mod) {
    int res = 1, exp = mod - 2;
    while (exp > 0) {
        if (exp % 2 == 1) res = (res * a) % mod;
        a = (a * a) % mod;
        exp /= 2;
    }
    return res;
}
```

4.7 Prime Factors of a number

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

vector<int> prime_factors(int n) {
   vector<int> v;
   while (n % 2 == 0) {
      v.push_back(2);
      n /= 2;
   }

   int sroot = sqrt(n);

   for (int i = 3; i <= sroot; i += 2) {
      while (n % i == 0) {
         v.push_back(i);
         n /= i;
      }
   }

   if (n > 2) {
      v.push_back(n);
   }

   return v;
}
```

5 Geometry

5.1 Geometry Functions

```
vec to_vec(const point& a, const point& b) {
  return vec(b.x - a.x, b.y - a.y);
}
ll dot(const vec& u, const vec& v) {
  return (u.x * v.x + u.y * v.y);
}
```

```
ll cross(const vec& u, const vec& v) {
  return (u.x * v.y - u.y * v.x);
}
// > 0 -> left turn ; = 0 -> collinear ; < 0 -> right turn
ll orient(const point& p, const point& q, const point& r) {
  return cross(to_vec(p, q), to_vec(p, r));
}
// return true if point p lies on the disk with diameter ab
bool in_disk(const point& a, const point& b, const point& p)
    {
  return dot(to_vec(p, a), to_vec(p, b)) <= 0;
}
// return true if point p is on segment ab, false otherwise
bool on_segment(const point& a, const point& b, const point
    &p) {
  return orient(a, b, p) == 0 && in_disk(a, b, p);
}
bool above(const point& a, const point& b) {
  return a.y >= b.y;
}
bool crosses_ray(const point& p, const point& q, const point
    & a) {
  return (above(q, a) - above(p, a)) * orient(a, p, q) > 0;
}
bool ccw(const point& p, const point& q, const point& r) {
  return orient(p, q, r) >= 0;
}
// return 1 if is insise, -1 if is outside and 0 if is on
    the perimeter
int in_polygon(const point& a, vector<point>& pol) {
  int num_crossings = 0;
  int n = pol.size();
  point p, q;
  for (int i = 0; i < n; i++) {
    p = pol[i];
    q = pol[(i + 1) % n];
    if (on_segment(p, q, a)) {
      return 0;
    }
    num_crossings += crosses_ray(p, q, a);
  }
  return ((num_crossings & 1) ? 1 : -1);
}
```

5.2 Convex Hull

```
vector<point> hull_andrew(vector<point>& pts) {
  int n = pts.size(), k = 0;
  vector<point> H(2 * n);
  sort(pts.begin(), pts.end());
  for (int i = 0; i < n; i++) {
    while ((k >= 2) && !ccw(H[k - 2], H[k - 1], pts[i])) k
        --;
    H[k++] = pts[i];
  }
  for (int i = n - 2, t = k + 1; i >= 0; i--) {
    while ((k >= t) && !ccw(H[k - 2], H[k - 1], pts[i])) k
        --;
    H[k++] = pts[i];
  }
  H.resize(k - 1);
  return H;
}
```

5.3 Structs

```
struct point {
  ll x, y;
  point() {x = y = 0L;}
  point(ll _x, ll _y) : x(_x), y(_y) {}
  bool operator < (point other) const {
    if (x == other.x) {
      return y < other.y;
    }
    return x < other.x;
  }
};

struct vec {
  ll x, y;
  vec(ll _x, ll _y) : x(_x), y(_y) {}
};
```

6 Other

6.1 Binary Search

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

int l = -1, r = n;
while (r - l > 1) {
    int m = (l + r) / 2;
    if (f(m)) {
        r = m; // 0 = f(l) < f(m) = 1
    } else {
        l = m; // 0 = f(m) < f(r) = 1
    }
}
```

6.2 Generating Subsets Recursively

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 1e9 + 7;
const double EPS = 1e-9;

void search(int k){
   if(k == n+1){
   //hacer algo aaa
   }else{
      subset.push_back(k);
      search(k+1);
      subset.pop_back();
      search(k+1);
   }
}
```

6.3 Increasing Subsequences

```
typedef long long ll;
#define srt(a) sort(a.begin(),a.end());
#include <bits/stdc++.h>

using namespace std;
int c;
vector<int> v,sub;

int subseq(int posAct){
    if(posAct == 0) return 1;
    if(sub[posAct] != -1) return sub[posAct];
    sub[posAct] = 1;
    for(int i = 1; i<c; i++){
        if(v[posAct-i] < v[posAct]){
            if( posAct - i >= 0){
                sub[posAct] = max(sub[posAct] , subseq(
                    posAct-i) +  1);
            }else{
                break;
            }
        }
    }
    return sub[posAct];
}

signed main (){
    std::ios::sync_with_stdio(false);cin.tie(0);
    cin>>c;
    v.resize(c);
    for(int i = 0; i<c; i++){
        cin>>v[i];
    }
    sub.assign(c,-1);
    int res = 1;
    for(int i = 1; i<c; i++){
        res = max(res, subseq(c-i));
    }
    cout<<res;
    return 0;
}
```

6.4 Knapsack

```
typedef long long ll;
#define srt(a) sort(a.begin(),a.end());

#include <bits/stdc++.h>

using namespace std;
vector<ll> val, wei;
vector<vector<ll>>dp;
ll w,n;
ll robar(int pos, int peso){
    if(pos == n) return 0;
    if(dp[pos][peso] != -1) return dp[pos][peso];

    dp[pos][peso] = robar(pos+1,peso); //sin tomar
    if(peso + wei[pos] <= w){
        dp[pos][peso] = max(dp[pos][peso] , robar(pos+1,
            peso+wei[pos]) + val[pos]); //tomando
    }
    return dp[pos][peso];
}

signed main (){
    std::ios::sync_with_stdio(false);cin.tie(0);
    cin>>n>>w;
    val.resize(n); wei.resize(n);

    for(int i = 0; i<n; i++){
        cin>>wei[i]>>val[i];
    }
    dp.assign(105,vector<ll> (100005,-1));
    ll res = robar(0,0);
    cout<<res<<endl;
  return 0;
}
```

6.5 Longest Increasing Subsequence

```
//cuadratico
int c;
vector<int> v,sub;
int subseq(int posAct){
    if(posAct == 0) return 1;
    if(sub[posAct] != -1) return sub[posAct];
    sub[posAct] = 1;
    for(int i = 1; i<c; i++){
        if(v[posAct-i] < v[posAct]){
            if( posAct - i >= 0){
                sub[posAct] = max(sub[posAct] , subseq(
                    posAct-i) +  1);
            }else{
                break;
            }
        }
    }
    return sub[posAct];
}
signed main (){
    std::ios::sync_with_stdio(false);cin.tie(0);
    cin>>c;
    v.resize(c);
    for(int i = 0; i<c; i++){
        cin>>v[i];
    }
    sub.assign(c,-1);
    int res = 1;
    for(int i = 1; i<c; i++){
        res = max(res, subseq(c-i));
    }
    cout<<res;
  return 0;
}


//opcion greedy nlogn

void printLis(int end,vector<int>&p, vector<int>&v){
        if(p[end] == -1) {cout<<v[end]<<endl;return;}
        printLis(p[end],p,v);
        cout<<v[end]<<endl;
}

signed main (){
        std::ios::sync_with_stdio(false);cin.tie(0);
        vector<int> v;
        int x;
        while(cin>>x){
                v.push_back(x);
        }
        int n = (int)v.size();
        int k=0,lie;
        vector<int>l(n,0),li(n,0),p(n,-1);
        for(int i = 0; i<n;i++){
                int pos = lower_bound(l.begin(),l.begin()+k,
                    v[i]) - l.begin();
                l[pos] = v[i];
                li[pos] = i;
                p[i] = -1;
                if(pos){
                        p[i] = li[pos-1];
                }
                if(pos == k){
                        k=pos+1;
                        lie = i;
                }
        }
        cout<<k<<endl;
        cout<<"-"<<endl;
        printLis(lie,p,v);
        return 0;
}
```