.

Dividimos y No Conquistamos (D&!C)

Contents

1 Template

1.1 C++ Template

```
#include <bits/stdc++.h>
using namespace std;
#define TESTS
#define LOCAL

#define ll long long
#define ii pair<ll, ll>
#define F first
#define S second
#define forni(i, o, n) for (int i = o; i < n; i ++)
#define forn(i, n) forni(i, 0, n)
#define pub push_back
#define popf pop_front

#ifdef LOCAL
#define DBG(x) cout << "[" << x << "]";
#else
#define DBG(x) 42
#endif


void solve(){

}


int main(){
    ios::sync_with_stdio(0);cin.tie(0);
#ifdef LOCAL
    freopen("in.txt","r", stdio);
    freopen("out.txt","w", stdout);
#endif
    int tt = 1;
#ifdef TESTS
    cin >> tt;
#endif
    while(tt--)solve();
}
```

2 Data structures

2.1 BIT

```
#define LSOne(S) (S & -S)

struct BIT {
    vector<int> B;
    int n;
    BIT(int n = 1): B(n + 1), n(n+1){}
    BIT(vector<int> &v): B(v.size()+1), n(v.size()+1) {
        for (int i = 1; i <= n; i ++){
            B[i] += v[i-1];
            if (i + LSOne(i) <= n){
                B[i + LSOne(i)] += B[i];
            }
        }
    }
    void update(int i, int x){
        while (i <= n){
            B[i] += x;
            i += LSOne(i);
        }
    }
    int sum(int i){
        int res = 0;
        while (i > 0){
            res += B[i];
            i -= LSOne(i);
        }
        return res;
    }
    int range_sum(int l, int r){
        return sum(r) - sum(l - 1);
    }
};
```

2.2 DSU

```
struct DSU {
    vector<int> par, sz;
    int n;
    DSU(int n = 1): par(n), sz(n, 1), n(n) {
```

```cpp
        for (int i = 0;i < n; i ++) par[i] = i;
    }
    int find(int a){
        return a == par[a] ? a : par[a] = find(par[a]);
    }
    void join(int a, int b){
        a=find(a);
        b=find(b);
        if (a != b){
            if (sz[b] > sz[a]) swap(a,b);
            par[b] = a;
            sz[a] += sz[b];
        }
    }
};
```

2.3 Sparse Table

```cpp
int log2_floor(unsigned long long i) {
    return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}

const int MAXN = 10;
int K = log2_floor(MAXN);
int st[K + 1][MAXN];

// Load Array to st[0][i]
std::copy(array.begin(), array.end(), st[0]);

// Build
for (int i = 1; (1 << i) <= n; i ++){
    for (int j = 0; j + (1 << (i - 1)) < n; j ++){
        st[i][j] = min(st[i-1][j], st[i-1][j + (1 << (i - 1))]);
    }
}

// Query
int min_range(int l, int r){
    int C = log2_floor(r - l + 1);
    return min(st[C][l], st[C][r - (1 << C) + 1]);
}
```

2.4 Segment tree

```cpp
struct Node {
    long long sum = 0;
    long long min_val = LLONG_MAX;
    long long max_val = LLONG_MIN;
    long long lazy = 0;

    // Merge function to combine two nodes
    void merge(const Node& left, const Node& right) {
        sum = left.sum + right.sum;
        min_val = min(left.min_val, right.min_val);
        max_val = max(left.max_val, right.max_val);
    }

    // Update function for lazy propagation
    void apply(int l, int r, long long value) {
        sum += (r - l + 1) * value;
        min_val += value;
        max_val += value;
        lazy += value;
    }
};

struct SegTree {
    int n;
    vector<Node> tree;

    SegTree(int n) : n(n) {
        tree.resize(4 * n + 5);
    }

    SegTree(vector<int>& arr) : n(arr.size()) {
        tree.resize(4 * n + 5);
        build(arr, 0, 0, n-1);
    }

    // Push lazy value to children
    void push(int id, int l, int r) {
        if (tree[id].lazy == 0) return;

        int mid = (l + r) >> 1;
        tree[2*id + 1].apply(l, mid, tree[id].lazy);
        tree[2*id + 2].apply(mid+1, r, tree[id].lazy);
        tree[id].lazy = 0;
    }

    void build(vector<int>& arr, int id, int l, int r) {
        if (l == r) {
            tree[id].sum = arr[l];
            tree[id].min_val = arr[l];
            tree[id].max_val = arr[l];
            return;
        }

        int mid = (l + r) >> 1;
        build(arr, 2*id + 1, l, mid);
        build(arr, 2*id + 2, mid+1, r);
        tree[id].merge(tree[2*id + 1], tree[2*id + 2]);
    }

    // Range update with lazy propagation
    void update(int id, int l, int r, int ql, int qr, long long val) {
        if (ql > r || qr < l) return;

        if (ql <= l && r <= qr) {
            tree[id].apply(l, r, val);
            return;
        }

        push(id, l, r);
        int mid = (l + r) >> 1;
        update(2*id + 1, l, mid, ql, qr, val);
        update(2*id + 2, mid+1, r, ql, qr, val);
        tree[id].merge(tree[2*id + 1], tree[2*id + 2]);
    }

    // Range query
    Node query(int id, int l, int r, int ql, int qr) {
        if (ql > r || qr < l) return Node();

        if (ql <= l && r <= qr) {
            return tree[id];
        }

        push(id, l, r);
        int mid = (l + r) >> 1;
        Node left = query(2*id + 1, l, mid, ql, qr);
        Node right = query(2*id + 2, mid+1, r, ql, qr);

        Node result;
        result.merge(left, right);
        return result;
    }

    // Public interface
    void update(int l, int r, long long val) {
        update(0, 0, n-1, l, r, val);
    }

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

3 Dynamic Programming

3.1 Knapsack

```
int knapsack(vector<int>& values, vector<int>& weights, int W) {
    int n = values.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for(int i = 1; i <= n; i++) {
        for(int w = 0; w <= W; w++) {
            if(weights[i-1] <= w) {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w-weights[i-1]] + values[i-1]);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    return dp[n][W];
}
```

3.2 LIS

```
vector<int> getLIS(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n + 1, INT_MAX);  // dp[i] = smallest value that ends
        an LIS of length i
```

```cpp
    vector<int> len(n);                // Length of LIS ending at each
        position
    dp[0] = INT_MIN;

    for(int i = 0; i < n; i++) {
        int j = upper_bound(dp.begin(), dp.end(), arr[i]) - dp.begin();
        dp[j] = arr[i];
        len[i] = j;
    }

    // Find maxLen and reconstruct sequence
    int maxLen = 0;
    for(int i = n-1; i >= 0; i--) maxLen = max(maxLen, len[i]);

    vector<int> lis;
    for(int i = n-1, currLen = maxLen; i >= 0; i--) {
        if(len[i] == currLen) {
            lis.push_back(arr[i]);
            currLen--;
        }
    }
    reverse(lis.begin(), lis.end());
    return lis;
}
```

3.3 Edit Distance

```cpp

//3. Edit Distance - O(n*m)
int editDistance(string& s1, string& s2) {
    int n = s1.length(), m = s2.length();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1));

    // Base cases
    for(int i = 0; i <= n; i++) dp[i][0] = i;
    for(int j = 0; j <= m; j++) dp[0][j] = j;

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            if(s1[i-1] == s2[j-1]) {
                dp[i][j] = dp[i-1][j-1];
            } else {
                dp[i][j] = 1 + min({dp[i-1][j],     // deletion
                                  dp[i][j-1],       // insertion
                                  dp[i-1][j-1]});   // replacement
            }
        }
    }
    return dp[n][m];
}
```

3.4 Kadane

```cpp
pair<int, pair<int,int>> kadane(vector<int>& arr) {
    int maxSoFar = arr[0], maxEndingHere = arr[0];
    int start = 0, end = 0, s = 0;

    for(int i = 1; i < arr.size(); i++) {
        if(maxEndingHere + arr[i] < arr[i]) {
            maxEndingHere = arr[i];
            s = i;
        } else {
            maxEndingHere += arr[i];
        }

        if(maxEndingHere > maxSoFar) {
            maxSoFar = maxEndingHere;
            start = s;
            end = i;
        }
    }
    return {maxSoFar, {start, end}}; // max, l, r
}
```

4 Strings

4.1 Prefix Tree

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

using namespace std;

struct TrieNodeStruct {
    TrieNodeStruct* children[26];
    bool isEndOfWord;

    TrieNodeStruct() {
        isEndOfWord = false;
        for(int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};

struct TrieStruct {
    TrieNodeStruct* root;

    TrieStruct() {
        root = new TrieNodeStruct();
    }

    void insert(string word) {
        TrieNodeStruct* current = root;
        for(char c : word) {
            int index = c - 'a';
            if(current->children[index] == nullptr) {
                current->children[index] = new TrieNodeStruct();
            }
            current = current->children[index];
        }
        current->isEndOfWord = true;
    }

    bool search(string word) {
        TrieNodeStruct* current = root;
        for(char c : word) {
            int index = c - 'a';
            if(current->children[index] == nullptr) {
                return false;
            }
            current = current->children[index];
        }
        return current->isEndOfWord;
    }

    bool startsWithDirect(string prefix) {
        TrieNodeStruct* current = root;
        for(char c : prefix) {
            int index = c - 'a';
            if(current->children[index] == nullptr) {
                return false;
            }
            current = current->children[index];
        }
        return true;
    }
};
```

4.2 HASHING

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

using namespace std;

// este struct permite crear unordered_set de pares
struct pair_hash {
  inline std::size_t operator()(const std::pair<ll,ll> & v) const {
    return v.first*31+v.second;
  }
};

const int p = 31; //representa la potencia, inicializalo en base a la
    cantidad de letras que use el alfabeto
const int m = 1e9 + 9; //mod

vector<ll> precompute_pow(ll size){ //el size debe ser el largo del
    string evaluado
  vector<ll> res(size);
  res[0] = 1;

  for(int i = 1; i < size; i++)
    res[i] = (res[i - 1] * p) % m;

  return res;
}

vector<ll> precompute_hash(string s, vector<ll> pow){
  ll size = s.size();
  vector<ll> hs(size + 1, 0);

  for(int i = 0; i < size; i++)
```

```cpp
    hs[i+1] = (hs[i] * ((s[i] - 'a' + 1) * pow[i]) % m) % m;

  return hs;
}
```

4.3 KMP

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

vector<int> kmp(string pat, string sec){ //geeks4geeks implementation
    with some changes
  int m = pat.length();
  int n = sec.length();
  cout << m << "␣" << n << endl;

  vector<int> lps = getLps(pat);
  vector<int> res;

  int i = 0;
  int j = 0;

  while((n - i) >= (m - j)){
    if(pat[j] == sec[i]){
      i++;
      j++;
    }
    if(j == m){
      res.push_back(i - j);
      j = lps[j - 1];
    }
    else{
      if(i < n && pat[j] != sec[i]){
        if(j != 0) j = lps[ j - 1 ];
        else i = i + 1;
      }
    }
  }

  return res;
}

vector<int> getLps(string pat){ //geek4geeks implementatio with some
    changes
  vector<int> lps(pat.length(), 0);
  int len = 0;
  int i = 1;
  lps[0] = 0;
  while(i < pat.length()){
    if(pat[i] == pat[len]){
      len++;
      lps[i] = len;
      i++;
    }
    else //pat[i] != pat[len]
    {
      lps[i] = 0;
      i++;
    }
  }

  return lps;
}
```

4.4 LPS

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

vector<int> getLps(string pat){ //geek4geeks implementatio with some
    changes
  vector<int> lps(pat.length(), 0);
  int len = 0;
  int i = 1;
  lps[0] = 0;
  while(i < pat.length()){
    if(pat[i] == pat[len]){
      len++;
      lps[i] = len;
      i++;
    }
    else //pat[i] != pat[len]
    {
      lps[i] = 0;
      i++;
    }
  }

  return lps;
}
```

4.5 RK

```
#include <bits/stdc++.h>
#define ll long long

using namespace std;

vector<int> rabin_karp(string s, string t) { //implementacion de cp-algorithms
  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;
}
```

4.6 Z FUNCTION

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

using namespace std;

vector<int> z_function(string s) {
    int n = s.length();
    vector<int> z(n, 0);

    z[0] = n;

    int l = 0, r = 0;

    for(int i = 1; i < n; i++) {
        if(i <= r) {
            z[i] = min(r - i + 1, z[i - 1]);
        }

        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }

        if(i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }

    return z;
}


vector<int> find_pattern(string text, string pattern) {

    string s = pattern + "$" + text;
    vector<int> z = z_function(s);
    vector<int> result;


    for(int i = pattern.length() + 1; i < s.length(); i++) {
        if(z[i] == pattern.length()) {
            result.push_back(i - pattern.length() - 1);
        }
    }

    return result;
}
```

5 Graph

5.1 Tarjan

```cpp
const int N = 10;

vector<int> G[N];
vector<int> dfs_low(N, -1), dfs_num(N, -1), ap(N, 0); // ap for
    Articulation Points
int dfs_count = 0;
int root = -1; // For AP

void dfs(int u, int p = -1){
    dfs_low[u]=dfs_num[u]=dfs_count++;
    int child = 0;
    for (int v: G[u]){
        if (v == p) continue;
        if (dfs_num[v] == -1){
            child ++;
            dfs(v, u);
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
            if (dfs_low[v] > dfs_num[u]){
              // Bridge from u -> v
              cout << "Bridge␣" << u << "␣->␣" << v << "\n";
            }
            if (dfs_low[v] >= dfs_num[u]) {
                // u is AP
                ap[u] = 1;
            }
        } else dfs_low[u] = min(dfs_low[u], dfs_num[v]);
    }
    if (u == root){
        ap[u] = child > 1;
    }
}
```

5.2 Bellman Ford

```cpp
struct Edge {
    int a, b, cost;
};

int n, m, v;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n, INF);
    d[v] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (Edge e : edges)
            if (d[e.a] < INF)
                d[e.b] = min(d[e.b], d[e.a] + e.cost);
}
```

5.3 SCC

```cpp
vector<int> dfs_num(N, -1), dfs_low(N, -1), visited(N);
int dfs_count = 0;
int numSCC = 0;
stack<int> st;
void dfs(int u){
  dfs_low[u]=dfs_num[u]=dfs_count++;
  st.push(u);
  visited[u] = 1;
  for(int v: G[u]) {
    if (dfs_num[v] == -1) dfs(v);
    if (visited[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
  }
  if (dfs_num[u] == dfs_low[u]){
    numSCC ++;
    while(1){
      int v = st.top(); st.pop();
      visited[v] = 0;
      if (u == v) break;
    }
  }
}
```

5.4 Flow

```cpp
// Complexity (V * V * E);
struct Dinic {
    struct Edge {
        int to, rev;
        long long cap, flow;
        Edge(int to, int rev, long long cap) :
            to(to), rev(rev), cap(cap), flow(0) {}
    };

    vector<vector<Edge>> g;
    vector<int> level, ptr;
    queue<int> q;
    int n, source, sink;
    const long long INF = 1e18;

    Dinic(int n, int s, int t) : n(n), source(s), sink(t) {
        g.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int from, int to, long long cap) {
        g[from].emplace_back(to, g[to].size(), cap);
        g[to].emplace_back(from, g[from].size()-1, 0);  // Reverse edge
    }

    bool bfs() {
        while(!q.empty()) {
            q.pop();
        }
        fill(level.begin(), level.end(), -1);

        q.push(source);
        level[source] = 0;

        while(!q.empty() && level[sink] == -1) {
            int v = q.front();
            q.pop();

            for(const Edge& e : g[v]) {
                if(level[e.to] == -1 && e.flow < e.cap) {
                    level[e.to] = level[v] + 1;
                    q.push(e.to);
                }
            }
        }
        return level[sink] != -1;
    }

    long long dfs(int v, long long pushed) {
        if(v == sink || pushed == 0) return pushed;

        for(int& i = ptr[v]; i < (int)g[v].size(); i++) {
            Edge& e = g[v][i];

            if(level[e.to] != level[v] + 1 || e.flow >= e.cap) continue;

            long long flow = dfs(e.to, min(pushed, e.cap - e.flow));
            if(flow == 0) continue;

            e.flow += flow;
            g[e.to][e.rev].flow -= flow;
            return flow;
        }
        return 0;
    }

    long long max_flow() {
        long long flow = 0;

        while(bfs()) {
            fill(ptr.begin(), ptr.end(), 0);
            while(long long pushed = dfs(source, INF)) {
                flow += pushed;
            }
        }
        return flow;
    }

    // Get the actual flow passing through each edge
    vector<vector<long long>> get_flow() {
        vector<vector<long long>> flow(n, vector<long long>(n, 0));
        for(int v = 0; v < n; v++) {
            for(const Edge& e : g[v]) {
                if(e.cap > 0) {  // Only original edges, not residual
                    flow[v][e.to] = e.flow;
                }
            }
        }
        return flow;
    }

    // Find minimum cut
    vector<bool> min_cut() {
        vector<bool> reachable(n, false);
        queue<int> q;
        q.push(source);
        reachable[source] = true;

        while(!q.empty()) {
            int v = q.front();
            q.pop();

            for(const Edge& e : g[v]) {
                if(!reachable[e.to] && e.flow < e.cap) {
                    reachable[e.to] = true;
                    q.push(e.to);
                }
            }
        }
        return reachable;
    }
};

// Example usage:
/*
int main() {
    // Example: 6 vertices, source = 0, sink = 5
    int n = 6;
    Dinic flow(n, 0, 5);

    // Add edges: (from, to, capacity)
    flow.add_edge(0, 1, 16);
    flow.add_edge(0, 2, 13);
    flow.add_edge(1, 2, 10);
    flow.add_edge(1, 3, 12);
    flow.add_edge(2, 1, 4);
    flow.add_edge(2, 4, 14);
    flow.add_edge(3, 2, 9);
    flow.add_edge(3, 5, 20);
    flow.add_edge(4, 3, 7);
    flow.add_edge(4, 5, 4);

    // Calculate maximum flow
    long long max_flow = flow.max_flow();
    cout << "Maximum flow: " << max_flow << "\n";

    // Get minimum cut
    vector<bool> cut = flow.min_cut();
    cout << "Vertices on source side of min cut: ";
    for(int i = 0; i < n; i++) {
        if(cut[i]) cout << i << " ";
    }
    cout << "\n";

    // Get flow through each edge
    auto flow_matrix = flow.get_flow();
    cout << "Flow matrix:\n";
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(flow_matrix[i][j] > 0) {
                cout << i << " -> " << j << ": " << flow_matrix[i][j] <<
                     "\n";
            }
        }
    }

    return 0;
}
*/
```

5.5 Ford Fulkerson

```
#define ll long long
const ll INF = (ll)4e18;
struct Edge {
    int from, to;
    ll cap, flow;
    Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow
        (0) {}
};

struct MaxFlow {
    vector<Edge> edges;
    vector<vector<int>> adj;
    vector<int> level, ptr;
    int n;
```

```cpp
    queue<int> q;

    MaxFlow(int n) : n(n) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int from, int to, ll cap) {
        edges.emplace_back(from, to, cap);
        edges.emplace_back(to, from, 0);
        adj[from].push_back(edges.size() - 2);
        adj[to].push_back(edges.size() - 1);
    }

    bool bfs(int s, int t) {
        while(!q.empty()) q.pop();
        fill(level.begin(), level.end(), -1);

        q.push(s);
        level[s] = 0;

        while(!q.empty() && level[t] == -1) {
            int v = q.front();
            q.pop();

            for(int id : adj[v]) {
                if(level[edges[id].to] == -1 && edges[id].cap - edges[id
                    ].flow > 0) {
                    level[edges[id].to] = level[v] + 1;
                    q.push(edges[id].to);
                }
            }
        }
        return level[t] != -1;
    }

    ll dfs(int v, int t, ll pushed) {
        if(v == t || pushed == 0)
            return pushed;

        for(; ptr[v] < (int)adj[v].size(); ptr[v]++) {
            int id = adj[v][ptr[v]];
            int u = edges[id].to;

            if(level[u] != level[v] + 1) continue;

            ll tr = dfs(u, t, min(pushed, edges[id].cap - edges[id].flow
                ));
            if(tr > 0) {
                edges[id].flow += tr;
                edges[id ^ 1].flow -= tr;
                return tr;
            }
        }
        return 0;
    }

    ll max_flow(int s, int t) {
        ll flow = 0;
        while(bfs(s, t)) {
            fill(ptr.begin(), ptr.end(), 0);
            while(ll pushed = dfs(s, t, LLONG_MAX)) {
                flow += pushed;
            }
        }
        return flow;
    }

    vector<ll> get_flows() {
        vector<ll> flows;
        for(int i = 0; i < edges.size(); i += 2) {
            flows.push_back(edges[i].flow);
        }
        return flows;
    }
};
```

6 Math

6.1 BINARY POW

```cpp
#include <iostream>
using namespace std;

typedef long long ll;
```

```
ll mod = 1e9+7;

ll binary_pow(ll base, ll exp) {
    ll result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }

    return result;
}
```

6.2 CATALAN

```
ll catalan(ll n) {
    if (n == 0) return 1;
    ll catalan_num = (fact[2 * n] * inv_fact[n] % MOD) * inv_fact[n + 1]
        % MOD;
    return catalan_num;
}
```

6.3 COMBINATORICS

```
vector<ll> fact, inv_fact;
void precompute_factorials(ll n, ll mod) {
    fact.resize(n + 1);
    inv_fact.resize(n + 1);
    fact[0] = inv_fact[0] = 1;
    for (ll i = 1; i <= n; i++) {
        fact[i] = (fact[i - 1] * i) % mod;
    }
    inv_fact[n] = mod_inverse(fact[n], mod);
    for (ll i = n - 1; i >= 1; i--) {
        inv_fact[i] = (inv_fact[i + 1] * (i + 1)) % mod;
    }
}

ll binomial_coefficient(ll n, ll k, ll mod) {
    if (k > n) return 0;
    return (fact[n] * inv_fact[k] % mod) * inv_fact[n - k] % mod;
}
```

6.4 EUCLIDEAN EXTENDED

```
ll extendedGCD(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1;
    ll gcd = extendedGCD(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}

bool findSolutionWithConstraints(ll a, ll b, ll c, ll x_min, ll y_min,
    ll &x, ll &y) {
    ll g = extendedGCD(a, b, x, y);

    if (c % g != 0) return false;

    x *= c / g;
    y *= c / g;

    // Ajustamos las variables a/g y b/g para mover las soluciones
    a /= g;
    b /= g;

    if (x < x_min) {
        ll k = (x_min - x + b - 1) / b;  // Redondeo hacia arriba
        x += k * b;
        y -= k * a;
    } else if (x > x_min) {
        ll k = (x - x_min) / b;
        x -= k * b;
        y += k * a;
    }

    if (y < y_min) {
        ll k = (y_min - y + a - 1) / a;  // Redondeo hacia arriba
        x += k * b;
        y -= k * a;
    } else if (y > y_min) {
        ll k = (y - y_min) / a;
        x -= k * b;
        y += k * a;
    }

    return x >= x_min && y >= y_min;
}
```

6.5 EULER TOTIENT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;


vector<ll> compute_totients(ll n) {
    vector<ll> phi(n + 1);
    for (ll i = 0; i <= n; i++) {
        phi[i] = i;
    }

    for (ll i = 2; i <= n; i++) {
        if (phi[i] == i) { // i es primo
            for (ll j = i; j <= n; j += i) {
                phi[j] = phi[j] * (i - 1) / i;
            }
        }
    }

    return phi;
}
```

6.6 JOSEPHUS

```
#include <iostream>
using namespace std;

typedef long long ll;

ll josephus_iterative(ll n, ll k) {
    ll result = 0;
    for (ll i = 2; i <= n; ++i) {
        result = (result + k) % i;
    }
    return result;
}


ll josephus_recursive(ll n, ll k) {

    if (n == 1)
        return 0;

    return (josephus_recursive(n - 1, k) + k) % n;
}


ll josephus_power_of_2(ll n) {

    ll power = 1;
    while (power <= n) {
        power <<= 1;
    }
    power >>= 1;


    return 2 * (n - power);
}
```

6.7 MOBIUS

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;


vector<ll> compute_mobius(ll n) {
    vector<ll> mu(n + 1, 1);
    vector<bool> is_prime(n + 1, true);

    for (ll i = 2; i <= n; i++) {
        if (is_prime[i]) { // i es un primo
            for (ll j = i; j <= n; j += i) {
                mu[j] *= -1; // Multiplicamos por -1 para cada primo
                is_prime[j] = false;
            }
            for (ll j = i * i; j <= n; j += i * i) {
                mu[j] = 0; // Si tiene un cuadrado de un primo, se pone
                    en 0
            }
        }
    }

    return mu;
}


ll mobius(ll x) {
    ll count = 0;
    for (ll i = 2; i * i <= x; i++) {
        if (x % (i * i) == 0)
            return 0;
        if (x % i == 0) {
            count++;
            x /= i;
        }
    }

    if (x > 1) count++;

    return (count % 2 == 0) ? 1 : -1;
}
```

6.8 NTT

```
#include <bits/stdc++.h>
using namespace std;
using cd = complex<double>;
typedef long long ll;
const ll mod = 998244353;
const ll root = 31;
const ll root_1 = inverse(root, mod);
const ll root_pw = 1 << 23;

ll inverse(ll a, ll m) {
    ll res = 1, exp = m - 2;
    while (exp) {
        if (exp % 2 == 1) res = (1LL * res * a) % m;
        a = (1LL * a * a) % m;
        exp /= 2;
    }
    return res;
}

void ntt(vector<ll> & 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 (auto & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}

vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
    vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    ll n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    ntt(fa, false);
    ntt(fb, false);
    for (ll i = 0; i < n; i++)
        fa[i] = (fa[i] * fb[i]) % mod;
    ntt(fa, true);

    vector<ll> result(n);
    for (ll i = 0; i < n; i++)
        result[i] = fa[i];
    return result;
}
```

6.9 PRIME FACTORIZATION

```

vector<pair<ll, ll>> prime_factors(ll n) {
    vector<pair<ll, ll>> factors;
    for (ll i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            ll count = 0;
            while (n % i == 0) {
                n /= i;
                count++;
            }
            factors.push_back({i, count});
        }
    }
    if (n > 1) factors.push_back({n, 1});
    return factors;
}


vector<ll> divisors(ll n) {
    vector<ll> divs;
    for (ll i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            divs.push_back(i);
            if (i != n / i) { // Evita duplicar si n es un cuadrado
                perfecto
                divs.push_back(n / i);
            }
        }
    }
    sort(divs.begin(), divs.end()); // Ordena los divisores en orden
        ascendente
    return divs;
}
```

6.10 SIEVE

```
#include <iostream>
#include <vector>
using namespace std;

typedef long long ll;

vector<ll> sieve_of_eratosthenes(ll n) {

    vector<ll> primes;
    vector<ll> primoRel(n,0);
    for(int i = 2; i < n; i++){
        if(!primoRel[i]){
            primes.push_back(i);
            for(int j = i*i; j < n ;j+=i){
                primoRel[j] = i;
            }
        }
    }

    return primes;
}
```

6.11 fft

```
#include <bits/stdc++.h>
using namespace std;
using cd = complex<double>;
typedef long long ll;
const double PI = acos(-1);

```

```
void fft(vector<cd> &a, bool invert) {
    ll n = a.size();
    if (n == 1)
        return;
    vector<cd> a0(n / 2), a1(n / 2);
    for (ll i = 0; 2 * i < n; i++) {
        a0[i] = a[2 * i];
        a1[i] = a[2 * i + 1];
    }
    fft(a0, invert);
    fft(a1, invert);
    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (ll i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n / 2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n / 2] /= 2;
        }
        w *= wn;
    }
}

vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    ll n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (ll i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<ll> result(n);
    for (ll i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}



//Exponensiacion binommial-----------------------------------------

vector<ll> binomial_exponentiation(const vector<ll> &a, int exp) {
    vector<ll> result = {1};
    vector<ll> base = a;

    while (exp > 0) {
        if (exp % 2 == 1) {
            result = multiply(result, base);
        }
        base = multiply(base, base);
        exp /= 2;
    }

    while (result.size() > 1 && result.back() == 0) {
        result.pop_back();
    }

    return result;
}

//FFT PRECISO -------------------------------------------------------

#define ll long long
using namespace std;
const double pi = acos(-1);

typedef long double ld;
typedef complex<ld> cd;
const ld 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)
```

```cpp
            swap(a[i], a[j]);
    }

    // Cooley-Tukey FFT
    for (int len = 2; len <= n; len <<=1) {
        ld ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cosl(ang), sinl(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            int len2 = len >> 1;
            for (int j = 0; j < len2; ++j) {
                cd u = a[i + j];
                cd v = a[i + j + len2] * w;
                a[i + j] = u + v;
                a[i + j + len2] = u - v;
                w *= wlen;
            }
        }
    }


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

vector<ll> multiply(const vector<ll>& a, const vector<ll>& b) {
    const ll BASE = 1e6;

    int n = 1;
    while(n < (int)(a.size() + b.size()))
        n <<= 1;

    vector<cd> al(n), ah(n), bl(n), bh(n);
    for (size_t i = 0; i < a.size(); ++i) {
        al[i] = a[i] % BASE;
        ah[i] = a[i] / BASE;
    }
    for (size_t i = 0; i < b.size(); ++i) {
        bl[i] = b[i] % BASE;
        bh[i] = b[i] / BASE;
    }

    fft(al, false);
    fft(ah, false);
    fft(bl, false);
    fft(bh, false);

    vector<cd> lx(n), lh(n), hl(n), hh(n);
    for (int i = 0; i < n; ++i) {
        lx[i] = al[i] * bl[i];
        lh[i] = al[i] * bh[i];
        hl[i] = ah[i] * bl[i];
        hh[i] = ah[i] * bh[i];
    }

    fft(lx, true);
    fft(lh, true);
    fft(hl, true);
    fft(hh, true);

    vector<ll> result(n);
    for (int i = 0; i < n; ++i) {
        ll temp_ll = llround(lx[i].real());
        ll temp_lh = llround(lh[i].real());
        ll temp_hl = llround(hl[i].real());
        ll temp_hh = llround(hh[i].real());

        result[i] = temp_ll +
                    ((temp_lh + temp_hl) * BASE) +
                    (temp_hh * BASE * BASE);
    }

    return result;
}
```

7 Geometry

7.1 CONVEX HULL

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

```

```cpp
typedef long long ll;
typedef pair<ll, ll> Point;

ll cross_product(Point O, Point A, Point B) {
    return (A.first - O.first) * (B.second - O.second) - (A.second - O.
        second) * (B.first - O.first);
}

vector<Point> convex_hull(vector<Point>& points) {
    sort(points.begin(), points.end());
    points.erase(unique(points.begin(), points.end()), points.end());
    vector<Point> hull;

    // Parte inferior
    for (const auto& p : points) {
        while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], p) < 0)
            hull.pop_back();
        if (hull.empty() || hull.back() != p) {
            hull.push_back(p);
        }
    }

    // Parte superior
    int t = hull.size() + 1;
    for (int i = points.size() - 1; i >= 0; --i) {
        while (hull.size() >= t && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], points[i]) < 0)
            hull.pop_back();
        if (hull.empty() || hull.back() != points[i]) {
            hull.push_back(points[i]);
        }
    }

    hull.pop_back();
    return hull;
}
```

7.2 OPERATIONS

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

typedef long long ll;


ll cross_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
    ll x1 = P2.first - P1.first;
    ll y1 = P2.second - P1.second;
    ll x2 = P3.first - P1.first;
    ll y2 = P3.second - P1.second;
    return x1 * y2 - y1 * x2;
}


double distancia(pair<ll, ll> P1, pair<ll, ll> P2) {
    return sqrt((P2.first - P1.first) * (P2.first - P1.first) +
                (P2.second - P1.second) * (P2.second - P1.second));
}


ll dot_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
    ll x1 = P2.first - P1.first;
    ll y1 = P2.second - P1.second;
    ll x2 = P3.first - P1.first;
    ll y2 = P3.second - P1.second;
    return x1 * x2 + y1 * y2;
}
```

7.3 POLYGON AREA

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

typedef long long ll;
typedef pair<ll, ll> Point;


double polygon_area(const vector<Point>& polygon) {
    ll area = 0;
    int n = polygon.size();
    for (int i = 0; i < n; ++i) {
        ll j = (i + 1) % n;
        area += (polygon[i].first * polygon[j].second - polygon[i].
            second * polygon[j].first);
    }
    return abs(area) / 2.0;
}
```

7.4 RAY CASTING

```
#include <iostream>
#include <vector>
using namespace std;

typedef long long ll;
typedef pair<ll, ll> Point;


bool is_point_in_polygon(const vector<Point>& polygon, Point p) {
    bool inside = false;
    int n = polygon.size();
    for (int i = 0, j = n - 1; i < n; j = i++) {
        if ((polygon[i].second > p.second) != (polygon[j].second > p.
            second) &&
            p.first < (polygon[j].first - polygon[i].first) * (p.second
                - polygon[i].second) /
                      (polygon[j].second - polygon[i].second) + polygon[
                          i].first) {
            inside = !inside;
        }
    }
    return inside;
}
```