Snippets 2025

DóndeEstásCR7

09/08/2025

Índice

Algos

1.1 Fast Io

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#define cpu() ios::sync_with_stdio(false);cin.tie(nullptr);

using namespace std;
using namespace __gnu_pbds;
template <class T>
using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag, tree_order_statistics_node_update>;

#define pb push_back
#define sz(a) ((int)(a).size())
#define ff first
#define ss second
#define all(a) (a).begin(), (a).end()
#define allr(a) (a).rbegin(), (a).rend()
#define approx(a) fixed << setprecision(a)

template <class T> void read(vector<T> &v);
template <class F, class S> void read(pair<F, S> &p);
template <class T, size_t Z> void read(array<T, Z> &a);
template <class T> void read(T &x) {cin >> x;}
template <class R, class... T> void read(R& r, T&... t){read(r); read(t...);};
template <class T> void read(vector<T> &v) {for(auto& x : v) read(x);}
template <class F, class S> void read(pair<F, S> &p) {read(p.ff, p.ss);}
template <class T, size_t Z> void read(array<T, Z> &a) { for(auto &x : a) read(x); }

template <class F, class S> void pr(const pair<F, S> &x);
template <class T> void pr(const T &x) {cout << x;}
template <class R, class... T> void pr(const R& r, const T&... t) {pr(r); pr(t...);}
template <class F, class S> void pr(const pair<F, S> &x) {pr("{", x.ff, ", ", x.ss, "}\n");}
void ps() {pr("\n");}
template <class T> void ps(const T &x) {pr(x); ps();}
template <class T> void ps(vector<T> &v) {for(auto& x : v) pr(x, ' '); ps();}
template <class T, size_t Z> void ps(const array<T, Z> &a) { for(auto &x : a) pr(x, ' '); ps(); }
template <class F, class S> void ps(const pair<F, S> &x) {pr(x.ff, ' ', x.ss); ps();}
template <class R, class... T> void ps(const R& r,  const T &...t) {pr(r, ' '); ps(t...);}

using ll = long long;
const double PI = 3.141592653589793;
const ll MX = 1e9 + 1;

void solve() {

}

int main() {
  cpu();

  int t = 1;
  //cin >> t;
  while (t--) {
    solve();
  }

  return 0;
}
```

Bit Manipulation

Técnicas para manipular bits individuales y operaciones a nivel de bit. Incluye macros útiles para competencias de programación.

2.1 Bits

Macros esenciales para manipulación de bits: verificar potencias de 2, establecer/limpiar bits, contar bits, y operaciones con LSB/MSB.

```cpp
using ull = unsigned long long;
const ull UNSIGNED_LL_MAX = 18'446'744'073'709'551'615;
#define isPowerOfTwo(S) ((S) && !((S) & ((S) - 1))) // Verifica si S es potencia de dos (y distinto de cero)
#define nearestPowerOfTwo(S) (1LL << lround(log2(S))) // Retorna la potencia de dos mas cercana a S
```

```cpp
#define modulo(S, N) ((S) & ((N) - 1)) // Calcula S % N cuando N es potencia de dos

#define isOn(S, i) ((S) & (1LL<<(i))) // Verifica si el bit esta encendido (bit en 1)
#define setBit(S, i) ((S) |= (1LL<<(i))) // Enciende el bit (Lo pone en 1)
#define clearBit(S, i) ((S) &= ~(1LL<<(i))) // Apaga el bit (Lo pone en 0)
#define toggleBit(S, i) ((S) ^= (1LL<<(i))) // Invierte el estado del bit (0 <-> 1)
#define setAll(S, n) ((S) = ((n)>=64 ? ~0LL : (1LL << (n))-1)) // Enciende los primeros 'n' bits (idx-0)

#define lsb(S) ((S) & -(S)) // Extrae el bit menos significativo 0100 (Least Significant Bit)
#define idxLastBit(x) __builtin_ctzll(x) // Numero de ceros a la derecha (Posicion del LSB, idx-0)
#define msb(S) (1LL << (63 - __builtin_clzll(S))) // Extrae el bit mas significativo 0100 (Most Significant Bit)
#define idxFirstBit(x) (63 - __builtin_clzll(x)) // Posicion del MSB (63 - ceros a la izquierda, idx-0)
#define countAllOnes(x) __builtin_popcountll(x)
#define turnOffLastBit(S) ((S) & ((S) - 1)) // Apaga el ultimo bit encendido (el menos significativo)
#define turnOnLastZero(S) ((S) | ((S) + 1)) // Enciende el ultimo cero menos significativo
#define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1)) // Apaga todos los bits encendidos mas a la derecha consecutivos
#define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1)) // Enciende los ceros consecutivos mas a la derecha

// Mascara de bits (mask -> subconjunto) O(2^N)
for (int mask = 0; mask < (1 << N); mask++)

// Recorrer subconjuntos de un superconjunto (menos el vacio)
int b = 0b1011; // Representacion binaria de un decimal en int
for (int i = b; i; i = (i - 1) & b) {
  cout << bitset<4>(i) << "\n";
}

void printBin(ll x) {
  // 63 -> unsigned ll, 62 -> ll, 31 -> unsigned int, 30 -> int
  for (ll i = 63; i >= 0; i--)
    cout << ((x >> i) & 1);
  cout << '\n';
}
```

Combinatory

3.1 Combinatory

```cpp
vector<mint> inverse, fact, inv_fact;

void generateBC(int N = 1e5){
  const int mod = mint().MOD;

  inverse.resize(N + 1); fact.resize(N + 1);
      inv_fact.resize(N + 1);
  inverse[1] = 1;

  for(int i = 2; i <= N; i++)
    inverse[i] = mod - (mod / i * inverse[mod % i]);

  fact[0] = inv_fact[0] = 1;
  for(int i = 1; i <= N; i++){
    fact[i] = fact[i - 1] * mint(i);
    inv_fact[i] = inv_fact[i - 1] * inverse[i];
  }
};

mint C(int n, int k){
  if(k > n) return mint(0);
  assert(n < fact.size() && k < fact.size());
  return fact[n] * inv_fact[k] * inv_fact[n - k];
}
```

Graph

Algoritmos de grafos: DFS, BFS, componentes fuertemente conexas, y otras estructuras de datos para problemas de grafos.

4.1 Bfs

```cpp
vector<bool> vis(n+1);
queue<int> q;
function<void(int)> bfs = [&](int start) {
  vis[start] = true;
  q.push(start);
  while (!q.empty()) {
    int sz = q.size();
    while (sz--) {
      int u = q.front();
      q.pop();
      for (int& v : adj[u]) {
        if (vis[v]) continue;
        vis[v] = true;
        q.push(v);
      }
    }
  }
};

for (int u = 1; u <= n; u++) {
  if (vis[u]) continue;
  bfs(u);
}
```

4.2 Bipartite

```cpp
int N, M; cin >> N >> M;
vector<vector<int>> adj(N + 1);
while (M--) {
  int u, v; cin >> u >> v;
  adj[u].push_back(v);
  adj[v].push_back(u);
}

vector<bool> vis(N + 1);
vector<int> col(N + 1, 0);
// bipartite graph
function<bool(int, int)> dfs = [&](int u, int c) {
  vis[u] = 1;
  col[u] = c;

  for (auto v : adj[u]) {
    if (vis[v] && col[u] == col[v]) return false;
    else if (!vis[v] && !dfs(v, c ^ 1)) return false;
  }
  return true;
};

for (int i = 1; i <= N; i++) {
  if (vis[i]) continue;
  if (dfs(i, 1) == false) {
    cout << "IMPOSSIBLE";
    return;
  }
}

for (int i = 1; i <= N; i++) cout << (col[i] ? 1 : 2) << ' ';
```

4.3 Dfs

```cpp
vector<bool> vis(n+1);
function<void(int)> dfs = [&](int u) {
  vis[u] = true;
  for (int& v : adj[u]) {
    if (vis[v]) continue;
    dfs(v);
  }
};

for (int u = 1; u <= n; u++) {
  if (vis[u]) continue;
  dfs(u);
}
```

4.4 Dfs 2D

```cpp
int N, M; cin >> N >> M;
vector<vector<char>> grid(N, vector<char>(M));
for (int i = 0; i < N; i++) {
  for (int j = 0; j < M; j++) {
    cin >> grid[i][j];
  }
}

vector<vector<bool>> vis(N, vector<bool>(M));
vector<int> dx = {-1, 1, 0, 0}, dy = {0, 0, -1, 1};
function<void(int, int)> dfs = [&](int x, int y) {
  vis[x][y] = 1;

```

```
  for (int d = 0; d < 4; d++) {
    int nx = x + dx[d], ny = y + dy[d];
    if (0 <= nx && nx < N && 0 <= ny && ny < M &&
        grid[nx][ny] == '.' && !vis[nx][ny]) dfs(nx, ny);
  }
};

int comp = 0;
for (int i = 0; i < N; i++) {
  for (int j = 0; j < M; j++) {
    if (vis[i][j] || grid[i][j] == '#') continue;
    dfs(i, j);
    comp++;
  }
}

cout << comp;
```

4.5 Disjoint Set Union Dsu

```
struct DSU{
  vector<int> p, size;
  DSU(int n){
    p.resize(n + 1), size.resize(n + 1,1);
    for(int i = 1; i <= n; i++) p[i] = i;
  }

  int find(int x){
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
  }

  void merge(int x, int y){
    x = find(x), y = find(y);
    if(x == y) return;
    if(size[x] < size[y]) swap(x, y);
    size[x] += size[y];
    p[y] = x;
  }
};
```

4.6 Djisktra

```
template <class T> using pq = priority_queue<T>;
template <class T> using pqg = priority_queue<T, vector<T>,
    greater<T>>;

void solve() {
  int n, m; cin >> n >> m;
  vector<vector<pair<int, ll>>> adj(n+1);
  while (m--) {
    int u, v; ll w; cin >> u >> v >> w;
    adj[u].push_back({v, w});
  }

  vector<ll> dist(n+1, MX);
  pqg<pair<ll, int>> q;
  q.push({0LL, 1});
  dist[1] = 0LL;
  while (!q.empty()) {
    auto [d, u] = q.top();
    q.pop();
    if (dist[u] < d) continue;
    for (auto [v, w] : adj[u]) {
      ll new_d = d + w;
      if (new_d < dist[v]) {
        dist[v] = new_d;
        q.push({dist[v], v});
      }
    }
  }

  for (int u = 1; u <= n; u++) cout << dist[u] << ' ';
  cout << '\n';
}
```

4.7 Lowest Common Ancestor Lca

```
struct LCA{
  int n, l, timer = 0;
  vector<vector<int>> up, adj;
  vector<int> depth, in, out;

  LCA(int _n) {
    n = _n + 1;
    l = ceil(log2(n));
    up.resize(n, vector<int>(l + 1));
    adj.resize(n);
    depth.resize(n);
    in.resize(n);
```

```
    out.resize(n);
  }

  void add_edge(int p, int u){
    adj[p].push_back(u);
    adj[u].push_back(p);
  }

  void dfs(int u = 1, int p = 1){
    up[u][0] = p;
    depth[u] = depth[p] + 1;
    in[u] = ++timer;
    for(int level = 1; level <= l; level++){
      up[u][level] = up[up[u][level - 1]][level - 1];
    }
    for(int v : adj[u]){
      if(v == p) continue;
      dfs(v, u);
    }
    out[u] = ++timer;
  }

  bool is_ancestor(int p, int u){
    return in[p] <= in[u] && out[p] >= out[u];
  }

  int query(int u, int v){
    if(is_ancestor(u, v)) return u;
    if(is_ancestor(v, u)) return v;

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

  int ancestor(int u, int k){
    if(depth[u] <= k) return -1;
    for(int bit = 0; bit <= l; bit++){
      if(k >> bit & 1) u = up[u][bit];
    }
    return u;
  }

  int distance(int u, int v){
    return depth[u] + depth[v] - 2 * depth[query(u, v)];
  }
};
```

4.8 Scc

Algoritmo de Tarjan para encontrar componentes fuertemente conexas (SCC) en un grafo dirigido.

```
// "These works to find a componente fuertemente conexa
    that it's in directed graph"
struct SCC{
  int N = 0, id;
  vector<vector<int>> adj;
  vector<int> ind, low;
  stack<int> s;
  vector<bool> in_stack;
  vector<vector<int>> components;
  vector<int> component_id;

  //1-indexed
  SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
  SCC(const vector<vector<int>> & _adj){ adj = _adj, N =
      adj.size(); }

  void add_edge(int from, int to){
    adj[from].push_back(to);
  }

  void dfs(int u){
    low[u] = ind[u] = id++;
    s.push(u);
    in_stack[u] = true;
    for(int v : adj[u]){
      if(ind[v] == -1){
        dfs(v);
        low[u] = min(low[u], low[v]);
      }else if(in_stack[v]){
        low[u] = min(low[u], ind[v]);
      }
    }
    if(low[u] == ind[u]){
      components.emplace_back();
      vector<int> & comp = components.back();
      while(true){
        assert(!s.empty());
        int x = s.top(); s.pop();
        in_stack[x] = false;
        component_id[x] = components.size() - 1;
        comp.push_back(x);
        if(x == u) break;
      }
    }
  }

```

```
  vector<vector<int>> get(){
    ind.assign(N, - 1); low.assign(N, -1);
        component_id.assign(N, -1);
    s = stack<int>();
    in_stack.assign(N, false);
    id = 0;
    components = {};
    for(int i = 1; i < N; i++)
      if(ind[i] == -1) dfs(i);

    // reverse(components.begin(), components.end());
        return components; // SCC in topological order
    return components; // SCC in reverse topological order
  }
};
```

4.9 Topological Sort

```
vector<int> top_sort(vector<vector<int>>& adj){
  int n = adj.size();
  bool cycle = false;
  vector<int> sorted, color(n);
  function<void(int)> dfs = [&](int u){
    color[u] = 1;
    for(int v : adj[u]){
      if(color[v] == 0 && !cycle) dfs(v);
      else if(color[v] == 1) cycle = true;
    }
    color[u] = 2;
    sorted.push_back(u);
  };
  for(int i = 1; i < n; i++){
    if(color[i] == 0 && !cycle) dfs(i);
  }
  if(cycle){return {};}
  reverse(sorted.begin(), sorted.end());
  return sorted;
}
```

Number Theorry

5.1 Euler Toliente

```
class EulerTotiente {
  public:
  //* metodo en O(sqrt(n))
  template <typename T>
  T euler_classic(T n) {
    T result = n;
    for(T i = 2; i * i <= n; i++) {
      if(n % i == 0) {
        while(n % i == 0) n /= i;
        result -= result / i;
      }
    }
    if(n > 1) {
      result -= result / n;
    }
    return result;
  }

  //* metodo en O(nlog(log(n))
  void euler_faster(int n) {
    vector<int> phi(n + 1);
    for(int i = 0; i <= n; i++) {
      phi[i] = i;
    }
    for(int i = 2; i <= n; i++) {
      if(phi[i] == i) {
        for(int j = i; j <= n; j += i) {
          phi[j] -= phi[j] / i;
        }
      }
    }
    for(int i = 1; i <= n; i++) {
      cout << i << ' ' << phi[i] << '\n';
    }
  }
};
```

5.2 Number Theory

```
// Divisores de N: Hasta N = 10^6
vector<int> divisores(int N) {
```

```cpp
  vector<int> divs;
  for (int d = 1; d * d <= N; d++) {
    if (N % d == 0) {
      divs.push_back(d);
      if (N / d != d) divs.push_back(N / d);
    }
  }
  return divs;
}

// Factorizacion de N: Hasta N = 10^6
vector<pair<int, int>> factorizar(int N) {
  vector<pair<int, int>> facts;
  for (int p = 2; p * p <= N; p++) {
    if (N % p == 0) {
      int exp = 0;
      while (N % p == 0) {
        exp++;
        N /= p;
      }
      facts.push_back({ p, exp });
    }
  }
  if (N > 1) facts.push_back({ N, 1 });
  return facts;
}

// Primalidad: Hasta N = 10^6 - O(sqrt(N))
bool isPrime(int N) {
  if (N < 2) return false;
  for (int d = 2; d * d <= N; d++) {
    if (N % d == 0) return false;
  }
  return true;
}

// Maximo comun divisor (GCD): Algoritmo de Euclides
int gcd(int a, int b) {
  if (a > b) swap(a, b);
  if (a == 0) return b;
  return gcd(b % a, a);
}

// Minimo comun multiplo (LCM): Calculado con GCD
int lcm(int a, int b) {
  return (a * b) / gcd(a, b);
}
```

5.3 Phi Euler

```cpp
//* Phi Euler
//* Phi(n) = contar la cantidad de numero coprimos entre 1 a n
int phi(int n) {
    int ans = n;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            ans -= ans / i;
        }
    }
    if(n > 1) {
        ans -= ans / n;
    }
    return ans;
}


//* phi(n) -> complex: O(log(log(n)))
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

5.4 Potenciación Binaria

```cpp
//* Binpow
long long binpow(long long a, long long b, long long m) {
    a %= m;
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
```

```
    }
    return res;
}
```

5.5 Sieve

```
// Criba de Eratostenes: Hasta N = 10^6
// Con bitset<N> Hasta N = 10^8 en 1s
void sieve(vector<bool>& is_prime) {
  int N = (int) is_prime.size();
  if (!is_prime[0]) is_prime.assign(N+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int p = 2; p * p <= N; p++) {
    if (is_prime[p]) {
      for (int i = p * p; i <= N; i += p) {
        is_prime[i] = false;
      }
    }
  }
}
```

5.6 Sum Of Divisors

```
//* Sum of divs
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;
}
```

Segment Tree

6.1 Find Two Numbers

```
// "find two number where the sum is x, and gcd(a, b) > 1" b
auto find = [&](ll x){
  for(int d = 2; d <= x / 2; d++){
    if(x % d == 0){
      ll m = 1, n = (x / d) - 1;
      ll a = d * m, b = d * n;
      if(__gcd(a, b) > 1){
        cout<< a << ' ' << b;
        ps();
        return;
      }
    }
  }
};
```

6.2 Segment Tree Recursivo

```
template<typename T>
struct segment_tree{
  int N;
  T Z = 0;
  vector<T> tree;
  segment_tree(int N) : N(N) {
    tree.resize(2 * N);
  }

  segment_tree(vector<T>& A){
    N = (int) A.size();
    tree.resize(2 * N);
```

```
    build(A, 1, 0, N - 1);
  }

  auto& operator[](size_t i) { return tree[i]; } // this
      funciton works for get element int this position
private:

  T op(T& a, T& b){ return a + b; }
  // O (n)
  void build(vector<T>& values, int node, int l, int r){
    // if l and r are equal both are leaf node
    // left node = [l, m]
    // m = (l + r) / 2
    // left and right are nodes
    // left interval = [l, m], right intervla = [m + 1, r]
    // after complete fill nodes of left and right, we need
        to fill the [l, r] node
    if(l == r){
      tree[node] = values[l];
      return;
    }
    int m = (l + r) >> 1;
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    build(values, left, l, m);
    build(values, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  // O (log N)
  void modify(int pos, T value, int node,  int l,  int r){
    // if l and r are equal, we found our node and update it
    if(l == r){
      tree[node] = value;
      return;
    }
    int m = (l + r) >> 1; // we get the mid
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    if(pos <= m) modify(pos, value, left, l, m);
    else modify(pos, value, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  void update(int pos, T value, int node,  int l,  int r){
    // if l and r are equal, we found our node and update it
    if(l == r){
      tree[node] = op(tree[node], value);
      return;
    }
    int m = (l + r) >> 1; // we get the mid
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    if(pos <= m) update(pos, value, left, l, m);
    else update(pos, value, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  // O(log N)
  T query(int ql, int qr, int node, int l, int r){
    if(r < ql || l > qr) return Z; // CHECK
    if(ql <= l && r <= qr) return tree[node];
    int m = (l + r) >> 1;
    int left = node + 1;
    int right = node + 2 * (m - l + 1);
    T ansL = query(ql, qr, left, l, m);
    T ansR = query(ql, qr, right, m + 1, r);
    return op(ansL, ansR);
  }
public:
  void build(vector<T>& values){ build(values, 1, 0, N -
      1); }

  void modify(int pos, T value){ modify(pos, value, 1, 0, N
      - 1); }

  void update(int pos, T value){ update(pos, value, 1, 0, N
      - 1); }

  T query(int ql, int qr){ return query(ql, qr, 1, 0, N -
      1); }
};
```

6.3 Segment Tree V2

```
// "This segment_tree I understand better how it works"
template<typename T>
struct seg_tree {
    int N;
    T Z = 0;
    vector<T> tree;

    seg_tree(int N) : N(N) {
        tree.resize(4 * N);
    }
```

```
11
12      seg_tree(vector<T>& A) {
13          N = (int)A.size();
14          tree.resize(4 * N);
15          build(A, 1, 0, N-1);
16      }
17
18  private:
19      T op(T a, T b) {
20          return a + b;
21      }
22
23      void build(vector<T>& a, int node, int left, int right)
            {
24          if(left == right) {
25              tree[node] = a[left];
26              return;
27          }
28          int mid = (left + right) >> 1;
29          build(a, 2 * node, left, mid);
30          build(a, 2 * node + 1, mid + 1, right);
31          tree[node] = op(tree[2 * node], tree[2 * node + 1]);
32      }
33
34      void modify(int pos, T value, int node, int left, int
            right) {
35          if(left == right) {
36              tree[node] = value;
37              return;
38          }
39          int mid = (left + right) >> 1;
40          if(pos <= mid)
41              modify(pos, value, 2 * node, left, mid);
42          else
43              modify(pos, value, 2 * node + 1, mid + 1,
                    right);
44          tree[node] = op(tree[2 * node], tree[2 * node + 1]);
45      }
46
47      T query(int l, int r, int node, int left, int right) {
48          if(r < left || l > right) return Z;
49          if(l <= left && right <= r) return tree[node];
50          int mid = (left + right) >> 1;
51          T leftSum = query(l, r, 2 * node, left, mid);
52          T rightSum = query(l, r, 2 * node + 1, mid + 1,
                right);
53          return op(leftSum, rightSum);
54      }
55
56  public:
57      void build(vector<T>& a) { build(a, 1, 0, N-1); }
58      void modify(int pos, T value) { modify(pos, value, 1,
            0, N-1); }
59      T query(int l, int r) { return query(l, r, 1, 0, N-1); }
60  };
```

6.4 Segment Tree V3

```
1  // snippet seg_tree_2 "Description" b
2  template<class T>
3  struct segment_tree{
4      int n;
5      vector<T> tree;
6
7      segment_tree(int n){
8          this -> n = n;
9          tree.resize(2 * n);
10     }
11
12     segment_tree(vector<T>& values){
13         this -> n = values.size();
14         tree.resize(2 * n);
15         for(int i = 0; i < n; i++) upd(i, values[i]);
16     }
17
18     //CHANGE
19     T compare(T a, T b){
20         return a + b;
21     }
22
23     void modify(int index, T value){
24         index += n;
25         tree[index] = value;
26         for(index >>= 1; index >= 1; index >>= 1)
               tree[index]= compare(tree[2 * index], tree[2 *
               index + 1]);
27     }
28
29     void upd(int index, T value){
30         index += n;
31         tree[index] = compare(tree[index], value);
32         for(index >>= 1; index >= 1; index >>= 1)
               tree[index]= compare(tree[2 * index], tree[2 *
               index + 1]);
33     }
34
35     //BOTTOM - TOP
36     T query(int first, int last){
37         first += n, last += n;
38         T ans = 0;
39         while(first <= last){
```

```
            if(first % 2 == 1) ans = compare(ans,
                tree[first++]);
            if(last % 2 == 0) ans = compare(ans,
                tree[last--]);
            first >>= 1, last >>= 1;
        }
        return ans;
    }
};
```