.

Overflow-Master

Contents

1 Template

1.1 C++ Template

```
#include <bits/stdc++.h>
using namespace std;
#define L(i, j, n) for (int i = (j); i < (int)n; i++)
#define LI(i, j, n) for (int i = (j); i <= (int)n; i++)
#define R(i, j, n) for (int i = (j); i > (int)n; i--)
#define RI(i, j, n) for (int i = (j); i >= (int)n; i--)
#define SZ(x) int((x).size())
#define ALL(x) begin(x), end(x)
#define vec vector
#define pb push_back

using ll = long long;
using ld = long double;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

const int MOD = (int)1e9 + 7;
const int oo = (int)1e9;

void solve() {}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int TC = 1;
    // cin >> TC;
    while (TC--) {
        solve();
    }
    return 0;
}
```

1.2 Policy Based

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template <typename Key, typename Val = null_type>
using indexed_set =
    tree<Key, Val, less<Key>, rb_tree_tag,
        tree_order_statistics_node_update>;
// indexed_set<char> s;
// char val = *s.find_by_order(0); // acceso por indice
// int idx = s.order_of_key('a'); // busca indice del valor
template <class Key, class Val = null_type>
using htable = gp_hash_table<Key, Val>;
// como unordered_map (o unordered_set si Val es vacio), pero sin metodo
     count
```

2 Graph

2.1 Dijkstra

```
vec<pll> G[N];
vec<ll> dijk(ll s) {
    vec<ll> dist(N, oo);
    dist[s] = 0;
    priority_queue<pll, vec<pll>, greater<pll>> pq;
    pq.push({0ll, s});
    while (!q.empty()) {
        auto [d, u] = pq.top();
        pq.pop();
        if (d != dist[u]) continue;
        for (auto [v, w] : G[u]) {
            if (dist[v] > d + w) {
                dist[v] = d + w;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}
```

2.2 Bellman-Ford

```
void bellmanFord(int n, int source, vec<vec<pii>> &g, vec<int> &d) {
    d.assign(n, INT_MAX);
    d[source] = 0;

    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n; ++j) {
```

```
            for (auto &[a, c] : g[j]) {
                if (d[j] != INT_MAX && d[a] > d[j] + c) {
                    d[a] = d[j] + c;
                }
            }
        }
    }
}
```

2.3 Floyd-Warshall

```
const int N = 10;
int G[N][N];
L(k, 0, n)
    L(i, 0, n)
        L(j, 0, n)
            G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
```

2.4 Disjoint Sets

```
struct UFDS {
    vec<int> p, size;
    int numSets, n;
    UFDS(int n) : p(n), size(n, 1), n(n) {
        for (int i = 0; i < n; i++) p[i] = i;
        numSets = n;
    }
    int find(int i) { return (p[i] == i) ? i : (p[i] = find(p[i])); }
    void join(int i, int j) {
        int a = find(i), b = find(j);
        if (a != b) {
            if (size[b] > size[a]) swap(a, b);
            p[b] = a;
            size[a] += size[b];
            numSets--;
        }
    }
};
```

2.5 Kruskal

```
struct Edge {
    int w, u, v;
    Edge(int wx, int ux, int vx) { w = wx, u = ux, v = vx; }
    bool operator<(const Edge &other) const { return w < other.w; }
};

int main() {
    int V, E;
    cin >> V >> E;
    vector<Edge> EL(E);
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        EL[i] = Edge(w, u, v);
    }
    sort(EL.begin(), EL.end());
    int mst_cost = 0, num_taken = 0;
    UFDS UF(V);
    for (auto &[w, u, v] : EL) {
        if (UF.isSameSet(u, v)) continue;
        mst_cost += w;
        UF.unionSet(u, v);
        ++num_taken;
        if (num_taken == V - 1) break;
    }

    return 0;
}
```

2.6 Prim

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

typedef pair<int, int> pii;
vector<vector<pii>> AL;
vector<int> taken;
priority_queue<pii, vector<pii>, greater<pii>> pq;

void process(int u) {
    taken[u] = 1;
    for (auto &[v, w] : AL[u]) {
        if (!taken[v]) {
            pq.emplace(w, v);
        }
    }
```

```
}

int main() {
    int V, E;
    cin >> V >> E;
    AL.assign(V, vector<pii>());
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        AL[u].emplace_back(v, w);
        AL[v].emplace_back(u, w);
    }
    taken.assign(V, 0);
    process(0);
    int mst_cost = 0, num_taken = 0;
    while (!pq.empty()) {
        auto [w, u] = pq.top();
        pq.pop();
        if (taken[u]) continue;
        mst_cost += w;
        process(u);
        ++num_taken;
        if (num_taken == V - 1) break;
    }
    cout << "MST cost: " << mst_cost << endl;
    return 0;
}
```

2.7 Tarjan

```
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;
    }
}
```

2.8 SCC

```
struct SCC {
    int n;
    vec<vec<int>> G, G2;
    vec<int> order, sccId, vi;
    vec<vec<int>> components;
    int sccCount;

    SCC(int n) : n(n) {
        G.assign(n, vec<int>());
        G2.assign(n, vec<int>());
        sccId.assign(n, -1);
        sccCount = 0;
    }

    void addEdge(int u, int v) {
        G[u].pb(v);
        G2[v].pb(u);
    }

    void dfs1(int u) {
        vi[u] = 1;
        for (int v : G[u]) {
            if (!vi[v]) dfs1(v);
        }
```

```
        order.pb(u);
    }

    void dfs2(int u, int id) {
        vi[u] = 1;
        sccId[u] = id;
        components[id].pb(u);
        for (int v : G2[u]) {
            if (!vi[v]) dfs2(v, id);
        }
    }

    void findSCC() {
        vi.assign(n, 0);
        order.clear();
        L(i, 0, n) {
            if (!vi[i]) dfs1(i);
        }

        vi.assign(n, 0);
        sccCount = 0;
        components.clear();

        reverse(ALL(order));
        for (int u : order) {
            if (!vi[u]) {
                components.pb(vec<int>());
                dfs2(u, sccCount++);
            }
        }
    }

    vec<vec<int>> getCondensedGraph() {
        vec<vec<int>> sccGraph(sccCount);
        set<pii> edges;

        L(u, 0, n) {
            for (int v : G[u]) {
                int fromScc = sccId[u], toScc = sccId[v];
                if (fromScc != toScc &&
                    edges.find({fromScc, toScc}) == edges.end()) {
                    sccGraph[fromScc].pb(toScc);
                    edges.insert({fromScc, toScc});
                }
            }
        }
        return sccGraph;
    }

    int getSCCId(int u) { return sccId[u]; }
    vec<int> getSCC(int i) { return components[i]; }
    int getCount() { return sccCount; }
};
```

2.9 Euler-Tour

```
struct edge {
    int y;
    list<edge>::iterator rev;  // NO DIRIGIDOS: iterador para arista
        reversa
    edge(int y) : y(y) {}
};

list<edge> g[N];

void add_edge(int a, int b) {
    g[a].push_front(edge(b));
    auto ia = g[a].begin();    // NO DIRIGIDOS
    g[b].push_front(edge(a));  // NO DIRIGIDOS
    auto ib = g[b].begin();    // NO DIRIGIDOS
    ia->rev = ib;              // NO DIRIGIDOS
    ib->rev = ia;              // NO DIRIGIDOS
}

vec<int> p;

void go(int x) {
    while (SZ(g[x])) {
        int y = g[x].front().y;
        g[y].erase(g[x].front().rev);  // NO DIRIGIDOS: eliminar
        g[x].pop_front();
        go(y);
    }
    p.pb(x);
}

```

```cpp
vec<int> get_path(int x) {
    p.clear();
    go(x);
    reverse(ALL(p));
    return p;
}

void solve() {
    int n, m;
    cin >> n >> m;

    // vec<int> inDeg(n, 0), outDeg(n, 0);  // DIRIGIDOS
    vec<int> deg(n, 0);  // NO DIRIGIDOS

    L(i, 0, m) {
        int a, b;
        cin >> a >> b;
        a--;
        b--;
        add_edge(a, b);
        // inDeg[b]++; // DIRIGIDOS
        // outDeg[a]++; // DIRIGIDOS
        deg[a]++;  // NO DIRIGIDOS
        deg[b]++;  // NO DIRIGIDOS
    }

    // DIRIGIDOS (camino euleriano):
    // Nodo 0: outDeg[0] = inDeg[0] + 1 (nodo inicial)
    // Nodo n-1: inDeg[n-1] = outDeg[n-1] + 1 (nodo final)
    // Resto: inDeg[i] = outDeg[i]
    // L(i, 1, n - 1) {
    //     if (inDeg[i] != outDeg[i]) {
    //         cout << "IMPOSSIBLE\n";
    //         return;
    //     }
    // }
    // if (outDeg[0] != inDeg[0] + 1 || inDeg[n - 1] != outDeg[n - 1] + 1) {
    //     cout << "IMPOSSIBLE\n";
    //     return;
    // }

    // NO DIRIGIDOS: verificar que todos los grados sean pares
    L(i, 0, n) {
        if (deg[i] % 2) {
            cout << "IMPOSSIBLE\n";
            return;
        }
    }

    vec<int> path = get_path(0);

    if (SZ(path) != m + 1) {
        cout << "IMPOSSIBLE\n";
    } else {
        for (auto x : path) {
            cout << x + 1 << " ";
        }
        cout << "\n";
    }
}
```

3 Dynamic Programming

3.1 Knapsack

```cpp
void solve() {
    vec<int> prices(n);
    vec<int> pages(n);
    vec<vec<int>> dp(n + 1, vec<int>(x + 1, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= x; j++) {
            if (prices[i] <= j) {
                dp[i + 1][j] = max(dp[i][j], pages[i] + dp[i][j - prices[i]]);
            } else {
                dp[i + 1][j] = dp[i][j];
            }
        }
    }
}
```

3.2 LIS

```cpp
int lis(vector<int>& arr) {
    int n = arr.size();
```

```
    vector<int> lis(n, 1);
    for (int i = 1; i < n; i++) {
        for (int prev = 0; prev < i; prev++) {
            if (arr[i] > arr[prev] && lis[i] < lis[prev] + 1) {
                lis[i] = lis[prev] + 1;
            }
        }
    }
    return *max_element(lis.begin(), lis.end());
}
```

3.3 LCS

```
int lcs(string &S1, string &S2) {
    vec<vec<int>> dp(m + 1, vec<int>(n + 1, 0));
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            if (S1[i - 1] == S2[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    return dp[m][n];
}
```

3.4 Edit Distance

```
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];
}
```

4 Search

4.1 Binary Search

```
int binSearch(int arr[], int low, int high, int x) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

4.2 Sliding Window

```
int main() {
    int cant = 0, start = 0, end = 0, sum = 0;
    while (end < n) {
        while (end < n && sum < x) {
            sum += arr[end];
            end++;
        }
        while (start <= end && sum > x) {
            sum -= arr[start];
            start++;
        }
        if (sum == x) {
            cant++;
            sum -= arr[start];
            start++;
        }
    }
    cout << cant;
    return 0;
```

```
}
```

4.3 Count Bits

```
void update_bits_and_sum(long mask, vec<int> &bits_used, long long &sum)
    {
    for (long j = mask; j > 0; j &= j - 1) {
        int bit = __builtin_ctzll(j);  // lowest bit ON (0-index)
        if (bits_used[bit] == 0) {
            sum += (1LL << bit);
        }
        bits_used[bit]++;
    }
}
```

5 Queries

5.1 Fenwick Tree (BIT)

```
#include <bits/stdc++.h>
#define ll long long
#define MOD 1000000007
using namespace std;

const int MAXN = 200000;
ll BIT[MAXN + 1];  // Array para el BIT
ll arr[MAXN + 1];  // Array original

void update(int idx, ll delta, int n) {
    while (idx <= n) {
        BIT[idx] += delta;
        idx += idx & -idx;
    }
}

ll query(int idx) {
    ll sum = 0;
    while (idx > 0) {
        sum += BIT[idx];
        idx -= idx & -idx;
    }
    return sum;
}

ll rangeQuery(int L, int R) { return query(R) - query(L - 1); }

int main() {
    int n, q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> arr[i];
        update(i, arr[i], n);  // init
    }

    while (q--) {
        ios::sync_with_stdio(0);
        cin.tie(0);
        ll type, a, b;
        cin >> type >> a >> b;
        if (type == 1) {
            ll delta = b - arr[a];
            arr[a] = b;
            update(a, delta, n);
        } else {
            cout << rangeQuery(a, b) << "\n";
        }
    }

    return 0;
}
```

5.2 Segment Tree

```
struct SegTree {
    int n;
    vec<int> A, st, lazy;
    int l(int p) { return p << 1; }
    int r(int p) { return (p << 1) + 1; }
    int conquer(int a, int b) {
        if (a == -1) return b;
        if (b == -1) return a;
        return a + b;
    }
    void build(int p, int L, int R) {
        if (L == R)
            st[p] = A[L];
```

```
        else {
            int m = L + (R - L) / 2;
            build(l(p), L, m);
            build(r(p), m + 1, R);
            st[p] = conquer(st[l(p)], st[r(p)]);
        }
    }
    void propagate(int p, int L, int R) {
        if (lazy[p] != -1) {
            st[p] = lazy[p];
            if (L != R) {
                lazy[l(p)] = lazy[r(p)] = lazy[p];
            }
            lazy[p] = -1;
        }
    }
    int query(int p, int L, int R, int i, int j) {
        if (i > j || L > j || R < i) return 0;
        propagate(p, L, R);
        if (L >= i && R <= j) return st[p];
        int m = L + (R - L) / 2;
        return conquer(query(l(p), L, m, i, j), query(r(p), m + 1, R, i,
             j));
    }
    void update(int p, int L, int R, int i, int j, int v) {
        if (i > j || L > j || R < i) return;
        propagate(p, L, R);
        if (L >= i && R <= j) {
            lazy[p] = v;
            propagate(p, L, R);
        } else {
            int m = L + (R - L) / 2;
            update(l(p), L, m, i, j, v);
            update(r(p), m + 1, R, i, j, v);
            st[p] = conquer(st[l(p)], st[r(p)]);
        }
    }
    SegTree(int sz) : n(sz), st(4 * n), lazy(4 * n, -1) {}
    SegTree(const vec<int> &init) : SegTree((int)init.size()) {
        A = init;
        build(1, 0, n - 1);
    }
    void update(int i, int j, int val) { update(1, 0, n - 1, i, j, val);
         }
    int query(int i, int j) { return query(1, 0, n - 1, i, j); }
};
```

5.3 Index Compression

```
template <class T>
struct Index {  // If only 1 use Don't need to copy T type
    vec<T> d;
    int sz;
    Index(vec<T> &a) : d(ALL(a)) {
        sort(ALL(d));                         // Sort
        d.erase(unique(ALL(d)), end(d));  // Erase continuous duplicates
        sz = SZ(d);
    }
    int of(T e) { return lower_bound(ALL(d), e) - begin(d); }  // get
        index
    T at(int i) { return d[i]; }  // get value of index
};
```

6 Math

6.1 Sieve

```
void solve(int n) {
    for (int x = 2; x <= n; x++) {
        if (sieve[x]) continue;
        for (int u = 2 * x; u <= n; u += x) {
            sieve[u] = 1;
        }
    }
}
```

6.2 LCM

```
int lcm(int a, int b) { return (a * b) / __gcd(a, b); }
```

6.3 Binomial Coefficient

```
using ll = long long;
const int MAXN = 1e6 + 5;
const ll MOD = 1e9 + 7;
ll factorial[MAXN];

```

```
void build_factorials() {
    factorial[0] = 1;
    for (int i = 1; i < MAXN; i++) {
        factorial[i] = factorial[i - 1] * i % MOD;
    }
}
ll binomial_coefficient(int n, int k) {
    if (k < 0 || k > n) return 0;
    ll denom = factorial[k] * factorial[n - k] % MOD;
    return factorial[n] * exp(denom, MOD - 2) % MOD;
}
```

6.4 Closest Pairs

```
using ld = long double;
vec<pair<ld, ld>> closestPair(vector<pair<ld, ld>> coord, int n) {
    sort(ALL(coord));
    set<pair<ld, ld>> s;
    ld squaredDistance = LLONG_MAX;
    vec<pair<ld, ld>> ans;
    int j = 0;
    for (int i = 0; i < n; ++i) {
        ld D = ceil(sqrt(squaredDistance));
        while (coord[i].first - coord[j].first >= D) {
            s.erase({coord[j].second, coord[j].first});
            j += 1;
        }

        auto start = s.lower_bound({coord[i].second - D, coord[i].first
            });
        auto end = s.upper_bound({coord[i].second + D, coord[i].first});

        for (auto it = start; it != end; ++it) {
            ld dx = coord[i].first - it->second;
            ld dy = coord[i].second - it->first;
            ld preDist = min(squaredDistance, dx * dx + dy * dy);
            if (preDist < squaredDistance) {
                pair<ld, ld> one = {it->second, it->first};
                pair<ld, ld> two = {coord[i].first, coord[i].second};
                ans = {one, two};
                squaredDistance = preDist;
            }
        }

        // Insert the point as {y-coordinate, x-coordinate}
        s.insert({coord[i].second, coord[i].first});
    }
    return ans;
}
```

6.5 Distance

```
double dist(double x1, double y1, double x2, double y2) {
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
```

6.6 Catalan

```
const int MOD = ....const int MAX = ....int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= n; i++) {
        catalan[i] = 0;
        for (int j = 0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i - j - 1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}
```

6.7 Binary Exponentiation

```
ll power(ll a, ll b, ll m) {
    a %= m;
    ll res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
```