Academy of Cryptography Techniques ACM-ICPC Notebook 2025

Mục lục

1 Initial Setup

1.1 Template

```cpp
#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; i++)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; i--)
#define FORE(i, v) for (__typeof((v).begin()) i = (v).begin(); i != (v).end(); i++)
#define ALL(v) (v).begin(), (v).end()
#define ff first
#define ss second
#define MASK(i) (1LL << (i))
#define BIT(x, i) (((x) >> (i)) & 1)
#define __builtin_popcount __builtin_popcountll
using namespace std;
template <class X, class Y>
bool minimize(X &x, const Y &y)
{
    if (x > y)
    {
        x = y;
        return true;
    }
    else
        return false;
}
template <class X, class Y>
bool maximize(X &x, const Y &y)
{
    if (x < y)
    {
        x = y;
        return true;
    }
    else
        return false;
}
template <class T>
T Abs(const T &x)
{
    return (x < 0 ? -x : x);
}
/// template by buiduckhanh
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    // freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);
}
```

2 Graph

2.1 DFS

```cpp
void dfs(int u)
{
    visited[u] = true;
    for (auto v: adj[u])
    {
        if(!visited[v])
            dfs(v);
    }
}
```

2.2 DAG (Directed Acyclic Graph)

```cpp
int trace[100005], state[100005];
bool haveCircle = false;

void dfs_dag(int u)
{
    state[u] = 1;
    for (int v : adj[u])
    {
        if (state[v] == 0)
        {
            trace[v] = u;
            dfs_dag(v);
        }
        else if (state[v] == 1)
        {
            print(u, v);
            haveCircle = true;
            exit(0);
        }
    }
    state[u] = 2;
}

void print(int start, int end)
{
    vector<int> ans;
    ans.push_back(end);
    for (int cur = start; cur != end; cur = trace[cur])
        ans.push_back(cur);
    ans.push_back(end);
    cout << ans.size() << endl;
    reverse(ans.begin(), ans.end());
    for (auto i : ans)
        cout << i << ' ';
}
```

2.3 Euler Path

```cpp
// All vertices in graph have even degree for Euler cycle
set<int> adj[100005];
vector<int> ans;

void dfs_euler(int u)
{
  for(auto v:adj[u])
  {
    adj[v].erase(u);
    adj[u].erase(v);
    dfs_euler(v);
  }
  ans.push_back(u);
}
```

2.4 Topological Sort

```
vector<int> store; // store result in reverse order

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

2.5 Joints and Bridges

```
bool visited[100005];
vector<int> adj[100005];
int num[100005], low[100005];
int timeDFS = 0;
vector<pair<int, int>> bridges;
set<int> joints;

void dfs_bridges(int u, int father)
{
    num[u] = low[u] = ++timeDFS;
    for (int v : adj[u])
    {
        if (v == father)
            continue;
        else if (num[v] > 0)
            low[u] = min(low[u], num[v]);
        else
        {
            dfs_bridges(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > num[u])
                bridges.emplace_back(u, v);
        }
    }
}

// Find articulation points
void dfs_joints(int u, int father)
{
    num[u] = low[u] = ++timeDFS;
    int children = 0;
    for (int v : adj[u])
    {
        if (v == father)
            continue;
        else if (num[v] > 0)
            low[u] = min(low[u], num[v]);
        else
        {
            dfs_joints(v, u);
            children++;
            low[u] = min(low[u], low[v]);
            if (father == -1 && children >= 2)
                joints.insert(u);
            if (father != -1 && low[v] >= num[u])
                joints.insert(u);
        }
    }
}
```

2.6 Strongly Connected Components (SCC)

```
// Strongly Connected Components using Tarjan's algorithm
int low[100005], num[100005], cntDfs = 0, ans = 0;
bool is_deleted[100005];
int root[100005];
stack<int> store;

void dfs(int u)
{
    low[u] = num[u] = ++cntDfs;
    store.push(u);
    for (int v : adj[u])
    {
        if (is_deleted[v])
            continue;
        if (num[v] > 0)
            low[u] = min(low[u], num[v]);
        else
        {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
    }
    if (low[u] == num[u])
    {
        ans++; // Count number of SCCs
        while (true)
        {
            int v = store.top();
            store.pop();
            is_deleted[v] = true;
            root[v] = u; // Store root if needed
            if (v == u)
                break;
        }
    }
}
```

2.7 BFS

```
queue<int> q;
int dist[1'000'005];
void bfs()
{
    for (int i = 0; i <= m; i++)
        dist[i] = -1;
    dist[s0] = 0;
    q.push(s0);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (auto v:adj[u])
        {
            if (dist[v] == -1)
            {
                dist[v] = dist[u] + 1;
                if (v == 0)
                {
                    cout << dist[v];
                    exit(0);
                }
                q.push(v);
            }
        }
    }
}
```

2.8 Dijkstra

```
typedef pair<int, int> pii;
priority_queue<pii, vector<pii>, greater<pii>> pq;
int dist[100005];

void dijkstra(int start)
{
    pq.emplace(0, start);
    while (!pq.empty())
    {
        auto [d, u] = pq.top();
        pq.pop();
        if (d > dist[u])
            continue;
        for (auto [v, w] : adj[u])
            if (dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                pq.emplace(dist[v], v);
            }
    }
}
```

2.9 Disjoint Set Union and Kruskal

```
// Disjoint Set Union
int findRoot(int u) {
    return (root[u] = (root[u] == u) ? u : findRoot(root[u]));
}

void unionSet(int u, int v)
{
    int rootu = findRoot(u);
    int rootv = findRoot(v);
    if (rootu != rootv)
    {
```

```
            cnt_union--;
            root[rootu] = rootv;
        }
    }
```

3 Math

3.1 Modular Arithmetic

```
// Addition
ll addMod(ll a, ll b) {
    return (a % M + b % M) % M;
}

// Subtraction
ll subMod(ll a, ll b) {
    return (a % M - b % M + M) % M;
}

// Multiplication
ll mulMod(ll a, ll b) {
    return (a % M) * (b % M) % M;
}

// Division (when M is prime)
ll Division(ll a, ll b) {
    return mulMod(a, powMod(b, M-2));
}

// Exponentiation: use powMod function
ll powMod(ll a, ll b) {
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            (res *= a) %= M;
        (a *= a) %= M;
        b >>= 1;
    }
    return res;
}
```

3.2 Combinatorics

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

// Stars and Bars Problem (Euler's Distribution)
// N candies and M children. How many ways to distribute all N candies
// to M children such that each child gets at least one candy?
// Number of ways: C(N-1, M-1)
// Fast computation of C(n,k)

typedef long long ll;
const int MAXN = 2000005;
const int MOD = 1000000007;

ll fact[MAXN], inv_fact[MAXN];

ll powMod(ll a, ll b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}

void pre_calculator() {
    fact[0] = 1;
    for (int i = 1; i < MAXN; i++) {
        fact[i] = (fact[i - 1] * i) % MOD;
    }

    inv_fact[MAXN - 1] = powMod(fact[MAXN - 1], MOD - 2);
    for (int i = MAXN - 2; i >= 0; i--) {
        inv_fact[i] = (inv_fact[i + 1] * (i + 1)) % MOD;
    }
}

ll C(ll n, ll k) {
    if (k > n || k < 0) return 0;
    return (((fact[n] * inv_fact[k]) % MOD) * inv_fact[n - k]) % MOD;
}
```

4 Advanced Data Structures

4.1 Segment Tree

```
// Given a[1..n], for query(l,r) find min(a[1..r])

#include <iostream>
#include <math.h>
using namespace std;
const int inf = 1000000007;
int a[200005];
int it[4 * 200005];
int n, numQueries;

void build(int index, int L, int R)
{
    if (L == R)
    {
        it[index] = a[L];
        return;
    }
    int mid = (L + R) / 2;
    build(2 * index, L, mid);
    build(2 * index + 1, mid + 1, R);
    it[index] = min(it[2 * index], it[2 * index + 1]);
}

int get(int index, int L, int R, int l, int r)
{
    if (l > R || L > r)
    {
        return inf;
    }
    if (l <= L && R <= r)
    {
        return it[index];
    }
    int mid = (L + R) / 2;
    int vLeft = get(2 * index, L, mid, l, r);
    int vRight = get(2 * index + 1, mid + 1, R, l, r);
    return min(vLeft, vRight);
}

int main()
{
    cin >> n >> numQueries;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
    }
    build(1, 1, n);
    for (int query = 1; query <= numQueries; query++)
    {
        int l, r;
        cin >> l >> r;
        cout << get(1, 1, n, l, r) << endl;
    }
    return 0;
}
```

4.2 Lazy Propagation

```
#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; i++)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; i--)
using namespace std;
typedef pair<int, int> pii;
struct Node
{
    int lazyValue;
    int maxValue;
};
Node tree[maxP];
void pushDown(int index, int l, int r)
{
```

```
    if (tree[index].lazyValue > 0 && l < r)
    {
        tree[2 * index].lazyValue += tree[index].lazyValue;
        tree[2 * index].maxValue += tree[index].lazyValue;
        tree[2 * index + 1].lazyValue += tree[index].lazyValue;
        tree[2 * index + 1].maxValue += tree[index].lazyValue;
        tree[index].lazyValue = 0;
    }
}
void update(int index, int L, int R, int x, int y, int k)
{
    pushDown(index, L, R);

    if (x > R || L > y)
    {
        return;
    }

    if (x <= L && R <= y)
    {
        tree[index].lazyValue += k;
        tree[index].maxValue += k;
        pushDown(index, L, R);
        return;
    }

    int mid = (L + R) / 2;
    update(2 * index, L, mid, x, y, k);
    update(2 * index + 1, mid + 1, R, x, y, k);
    tree[index].maxValue = max(tree[2 * index].maxValue, tree[2 * index + 1].maxValue);
}
int get(int index, int L, int R, int x, int y)
{
    pushDown(index, L, R);

    if (x > R || L > y)
    {
        return -1;
    }

    if (x <= L && R <= y)
    {
        return tree[index].maxValue;
    }

    int mid = (L + R) / 2;
    int vLeft = get(2 * index, L, mid, x, y);
    int vRight = get(2 * index + 1, mid + 1, R, x, y);
    return max(vLeft, vRight);
}
int main()
{
    int n, m, x, y, k;
    cin >> n >> m;
    while (m--)
    {
        int type;
        cin >> type;
        if (type == 0)
        {
            cin >> x >> y >> k;
            update(1, 1, n, x, y, k);
        }
        else
        {
            cin >> x >> y;
            cout << get(1, 1, n, x, y) << endl;
        }
    }
}
```

4.3 Persistent Segment Tree

```
/*
CSES Problem Set
Range Queries and Copies:

Given a[1..n].
    query 1: change value a[i] in array k to x.
    query 2: sum(a[l..r]) in array k
    query 3: add new version of array k.
*/
#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; i++)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; i--)
using namespace std;
typedef long long ll;
int n, numQueries;
int a[200005];

struct Node
{
    Node *left;
    Node *right;
    ll sum;
    Node() : sum(0), left(nullptr), right(nullptr) {};
};
vector<Node *> version;

void BuildTree(Node *&root, int L, int R)
{
    root = new Node();
    if (L == R)
    {
        root->sum = a[L];
        return;
    }
    int mid = (L + R) >> 1;
    BuildTree(root->left, L, mid);
    BuildTree(root->right, mid + 1, R);
    root->sum = root->left->sum + root->right->sum;
}

void update(Node *&root, int L, int R, int pos, int val)
{
    root = new Node(*root);
    if (R < pos || L > pos)
        return;
    if (L == R)
    {
        root->sum = val;
        return;
    }
    int mid = (L + R) >> 1;
    if (pos <= mid)
        update(root->left, L, mid, pos, val);
    else
        update(root->right, mid + 1, R, pos, val);
    root->sum = root->left->sum + root->right->sum;
}

ll get(Node *&root, int L, int R, int l, int r)
{
    if (R < l || r < L)
        return 0;
    if (L >= l && r >= R)
        return root->sum;
    int mid = (L + R) >> 1;
    ll sum_L = get(root->left, L, mid, l, r);
    ll sum_R = get(root->right, mid + 1, R, l, r);
    return sum_L + sum_R;
}

void initial()
{
    cin >> n >> numQueries;
    FOR(i, 1, n)
    cin >> a[i];
    // Create seg tree for ver0
    Node *ver0 = new Node();
    version.push_back(ver0);
    BuildTree(version[0], 1, n);
}

int main()
{
    initial();
    while (numQueries--)
    {
        int type, k, val, pos, b, l, r;
        cin >> type >> k;
        k--;
        if (type == 1)
        {
            cin >> pos >> val;
            update(version[k], 1, n, pos, val);
        }
        else if (type == 2)
        {
            cin >> l >> r;
            cout << get(version[k], 1, n, l, r) << endl;
        }
        else if (type == 3)
        {
            Node *newVer = new Node(*version[k]);
            version.push_back(newVer);
        }
    }
}
```

4.4 Fenwick Tree

```
/*
Fenwick Tree:
    query 1: increase f[x] by k units.
    query 2: calculate sum(f[1->i]);

ICPC 2021 Southern Vietnam - I: Inversion Number
Choose number a[i]
    numbers < a[i] move to the left.
    numbers > a[i] move to the right.
    count inversions
=> return a[i] to minimize inversions.
*/

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; i++)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; i--)
#define ALL(v) (v).begin(), (v).end()

int n;
int a[1000005], fw[1000005], smaller[1000005], bigger[1000005];
ll sumSmaller[1000005], rsumBigger[1000005];

void update(int index, int vals)
{
    for (; index <= 1000000; index += index & -index)
        fw[index] += vals;
}

ll get(int index)
{
    ll res = 0;
    for (; index > 0; index -= index & -index)
        res += fw[index];
    return res;
}

int main()
{
    cin >> n;
    FOR(i, 1, n) { cin >> a[i]; }
    FOR(i, 1, n)
    {
        smaller[a[i]] = (a[i] - 1) - (get(a[i] - 1));
        bigger[a[i]] = (i - 1) - (get(a[i] - 1));
        // update fw[a[i]] = 1
        update(a[i], 1);
    }
    FOR(i, 1, n) { sumSmaller[i] = sumSmaller[i - 1] + smaller[i]; }
    FORD(i, n, 1) { rsumBigger[i] = rsumBigger[i + 1] + bigger[i]; }
    ll minVal = 1000000000000;
    FOR(i, 1, n)
    {
        minVal = min(minVal, sumSmaller[i - 1] + rsumBigger[i + 1]);
    }
    cout << minVal;
}

/*
smaller[x]: number of elements to the right < x.
bigger[x]: number of elements to the left > x.

inversions = sum(smaller[2..x] + bigger[x..n-1])
--------
for i,1,n
    fw: count 1->a[i]: how many numbers processed.
    smaller[a[i]] = (total < a[i]) - numbers <a[i] on left.
    smaller[a[i]] = (a[i] - 1) - get(a[i] - 1)
    ---
    bigger[a[i]] = left numbers - (smaller numbers on left)
    bigger[a[i]] = i - 1 - get(a[i] - 1)
*/
```

4.5 Trie

```
/*
Search Engine - hackerearth.

Given N strings, string i has power value p[i]
Given Q queries, each query has string st, among strings that have st as prefix,
which string has maximum power?
*/

#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= (_b); i++)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= (_a); i--)
#define FORE(i, a) for (auto i : a)
using namespace std;

bool maximum(int &X, const int &Y)
{
    if (X < Y)
    {
        X = Y;
        return true;
    }
    return false;
}

typedef long long ll;
struct TrieNode
{
    TrieNode *child[26];
    int maxWeight;
    TrieNode(int __maxWeight)
    {
        FOR(i, 0, 25) { child[i] = NULL; }
        maxWeight = __maxWeight;
    }
};

void addString(string &s, int w, TrieNode *&root)
{
    TrieNode *p = root;
    for (char c : s)
    {
        if (p->child[c - 'a'] != NULL)
            maximum(p->child[c - 'a']->maxWeight, w);
        if (p->child[c - 'a'] == NULL)
            p->child[c - 'a'] = new TrieNode(w);
        p = p->child[c - 'a'];
    }
}

int checkString(string &s, TrieNode *root)
{
    TrieNode *p = root;
    int w = -1;
    for (char c : s)
    {
        if (p->child[c - 'a'] == NULL)
            return -1;
        w = p->child[c - 'a']->maxWeight;
        p = p->child[c - 'a'];
    }
    return w;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, q, w;
    string s;
    cin >> n >> q;
    TrieNode *root = new TrieNode(0);
    FOR(i, 1, n)
    {
        cin >> s >> w;
        addString(s, w, root);
    }
    FOR(i, 1, q)
    {
        cin >> s;
        cout << checkString(s, root) << endl;
    }
}
```