Competitive Programming Library

Too bad to be Accepted

Contents

1 Dynamic Programming

2 Bit Manipulation

3 Algorithms

3.1 MO

MO Algorithm

```
// MO      -> O(N+Q SQRT(N)) <= 10^5

const int N = 1e5+5, M = 1e5+5;
int n, m;
int nums[N], q_ans[M];

struct query {
    int idx, block_idx, l, r;

    query() = default;
    query(int _l, int _r, int _idx) {
        idx = _idx;
        r = _r - 1;
        l = _l - 1;
        block_idx = _l / sqrt(n);
    }

    bool operator <(const query & y) const {
        if(y.block_idx == block_idx) return r < y.r;
        return block_idx < y.block_idx;
    }
};

int freq[N], ans;

void add(int idx) {
    freq[nums[idx]]++;
    if (freq[nums[idx]] == 2) ans++;
}

void remove(int idx) {
    freq[nums[idx]]--;
    if (freq[nums[idx]] == 1) ans--;
}

cin >> n >> m;
for (int i = 0; i < n; ++i) cin >> nums[i];
vector<query> Query(m);
for (int i = 0; i < m; ++i) {
    int l, r; cin >> l >> r;
    Query[i] = query(l, r, i);
}

sort(Query.begin(), Query.end());
int l0 = 1, r0 = 0;
for (int i = 0; i < m; ++i) {
    while (l0 < Query[i].l) remove(l0++);
    while (l0 > Query[i].l) add(--l0);
    while (r0 < Query[i].r) add(++r0);
    while (r0 > Query[i].r) remove(r0--);
    q_ans[Query[i].idx] = ans;
}
for (int i = 0; i < m; ++i) {
    cout << q_ans[i] << '\n';
}
```

4 Data Structures

4.1 Strings

4.1.1 Trie (Prefix Tree)

Basic Implementation

```
#define MAX_CHAR 26

struct TrieNode {
    TrieNode *pTrieNode[MAX_CHAR]{};
    bool isWord;

    TrieNode() {
        isWord = false;
        fill(pTrieNode, pTrieNode + 26, (TrieNode *) NULL);
    }

    virtual ~TrieNode() = default;
};

class Trie {
private:
    TrieNode *root;
public:
    Trie() {
        root = new TrieNode();
    }

    virtual ~Trie() = default;

    TrieNode *getTrieNode() {
        return this->root;
    }

    void insert(const string &word) {
        TrieNode *current = root;
        for (char c: word) {
            int i = c - 'a';
            if (current->pTrieNode[i] == nullptr)
                current->pTrieNode[i] = new TrieNode();
            current = current->pTrieNode[i];
        }
        current->isWord = true;
    }

    bool search(const string &word) {
        TrieNode *current = root;
        int ch = 0;
        for (char c: word) {
            ch = c - 'a';
            if (current->pTrieNode[ch] == nullptr)
                return false;
            current = current->pTrieNode[ch];
        }
        return current->isWord;
    }

    bool startsWith(const string &prefix) {
        TrieNode *current = root;
        int ch = 0;
        for (char c: prefix) {
            ch = c - 'a';
            if (current->pTrieNode[ch] == nullptr)
                return false;
            current = current->pTrieNode[ch];
        }
        return true;
    }
};
```

4.2 Range Queries

4.2.1 Segment Tree

Basic Implementation

```
struct Node {
    long long val;
};

struct SegTree {
private:
    const Node NEUTRAL = {INT_MIN};

    static Node merge(const Node& x1, const Node& x2) {
        return {x1.val + x2.val};
```

```
    }

    void set(const int& idx, const int& val, int x, int lx, int rx) {
        if (rx - lx == 1) return void(values[x].val = val);

        int mid = (rx + lx) / 2;

        if (idx < mid)
            set(idx, val, 2 * x + 1, lx, mid);
        else
            set(idx, val, 2 * x + 2, mid, rx);

        values[x] = merge(values[2 * x + 1], values[2 * x + 2]);
    }

    Node query(const int& l, const int& r, int x, int lx, int rx) {
        if (lx >= r || l >= rx) return NEUTRAL;
        if (lx >= l && rx <= r) return values[x];

        int mid = (rx + lx) / 2;

        return merge(query(l, r, 2 * x + 1, lx, mid), query(l, r, 2 * x +
            2, mid, rx));
    }
    void build(vector<int> &a, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < a.size()) {
                values[x] = single(a[lx]);
            }
            return;
        }
        int m = (lx + rx) / 2;
        build(a, 2 * x + 1, lx, m);
        build(a, 2 * x + 2, m, rx);
        values[x] = merge(values[2 * x + 1], values[2 * x + 2]);
    }

public:
    int size{};
    vector<Node> values;

    void build(vector<int> &a) {
        build(a, 0, 0, size);
    }

    void init(int _size) {
        size = 1;
        while (size < _size) size *= 2;
        values.assign(2 * size, NEUTRAL);
    }

    void set(int idx, int val) {
        set(idx, val, 0, 0, size);
    }

    Node query(const int& l, const int& r) {
        return query(l, r, 0, 0, size);
    }
};
```

4.2.2 Fenwick Tree

Fenwick Tree

```
struct Fenwick {
    // One Based
    vector<int> tree;

    explicit Fenwick(int n) {tree.assign(n + 5, {});}

    // Computes the prefix sum from [1, i], O(log(n))
    int query(int i) {
        int res = 0;
        while (i > 0) {
            res += tree[i];
            i &= ~(i & -i);
        }
        return res;
    }

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

    // Get the value at index i
    int get(int i) {
        return query(i, i);
    }
```

```
    // Add 'v' to index 'i', O(log(n))
    void update(int i, int v) {
        while (i < tree.size()) {
            tree[i] += v;
            i += (i & -i);
        }
    }
};
```

4.2.3 Sparse Table

4.3 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void erase_set(ordered_set &os, int v) {
    // Number of elements less than v
    int rank = os.order_of_key(v);

    auto it = os.find_by_order(rank);
    os.erase(it);
}
```

Ordered Set

5 Counting Principles

5.1 nCr

$$C(n,k) = \frac{n!}{(n-k)!k!} = \frac{n*(n-1)*(n-2)*\ldots*(n-k+1)}{k!}$$

5.1.1 Fast nCr

$$C(n,k) = \frac{n*(n-1)*(n-2)*\ldots*(n-k+1)}{1*2*3*\ldots*k} = \prod_{i=0}^{k-1}\frac{n-i}{i+1} = \prod_{i=0}^{k-1}(n-i)(i+1)^{-1}$$

Fast nCr

```
int nCr(const int& n, const int& r) {
    double res = 1;
    for (int i = 1; i <= r; ++i)
        res = res * (n - r + i) / i;
    return (int)(res + 0.01);
}
```

6 Graph Theory

6.1 Dijkstra Algorithm

```
#define INF (1e18) // for int defined as ll

int n, m;
vector<vector<pair<int, int>>> adj;
vector<int> cost;
vector<int> parent;

void dijkstra(int startNode = 1) {
    priority_queue<pair<ll, int>, vector<pair<ll, int>>,
   greater<>> pq;

    cost[startNode] = 0;
    pq.emplace(0, startNode);

    while (!pq.empty()) {
        int u = pq.top().second;
        ll d = pq.top().first;
        pq.pop();

        if (d > cost[u]) continue;

        for (auto &p: adj[u]) {
            int v = p.first;
            int w = p.second;
            if (cost[v] > cost[u] + w) {
                cost[v] = cost[u] + w;
                parent[v] = u;
                pq.emplace(cost[v], v);
            }
        }
    }
}

void run_test_case(int testNum) {
    cin >> n >> m;

    adj.assign(n + 1, {});
    cost.assign(n + 1, INF);
    parent.assign(n + 1, -1);

    while (m--) {
        // Read Edges
    }

    dijkstra();

    if (cost[n] == INF) {
        cout << -1 << el; // not connected {Depends on you
   use case}
        return;
    }

    stack<int> ans;
    for (int v = n; v != -1; v = parent[v]) ans.push(v);

    while (!ans.empty()) { // printing the path
        cout << ans.top() << ' ';
        ans.pop();
    }
    cout << el;
}
```

Dijkstra Implementation

7 Techniques

7.1 Coordinate Compression

```
void coordinate_compress(vector<int> &x, int start=0, int
   step=1) {
 set unique(x.begin(), x.end());
 map<int, int> valPos;

 int idx=0;
 for (auto i: unique) {
  valPos[i] = start + idx * step;
  ++idx;
 }
 for(auto &i: x) i = valPos[i];

}
```

Coordinate Compression

8 Templates

8.1 MOD Template

```
constexpr int MOD = 1e9+7; // must be a prime number

int add(int a, int b) {
    int res = a+b;
    if(res >= MOD) return res -= MOD;
}

int sub(int a, int b) {
    int res = a-b;
    if(res < 0) return res += MOD;
}

int power(int a, int e) {
    int res = 1;
    while(e) {if(e & 1) res = res * a % MOD; a = a * a % MOD;
    e >>= 1;}
    return res;
}

int inverse(int a) {
    return power(a, MOD-2);
}

int div(int a, int b) {
    return a * inverse(b) % MOD;
}
```

MOD Template