ACM-ICPC Team Reference Document

Vilnius University (Šimoliūnaitė, Strakšys, Strimaitis)

Contents

1 Data Structures

1.1 Disjoin Set Union

```
struct DSU {
    vector<int> par;
    vector<int> sz;

    DSU(int n) {
        FOR(i, 0, n) {
            par.pb(i);
            sz.pb(1);
        }
    }

    int find(int a) {
        return par[a] = par[a] == a ? a : find(par[a]);
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);
        if(sz[a] > sz[b]) swap(a, b);
        sz[b] += sz[a];
        par[a] = b;
    }
};
```

1.2 Fenwick Tree Point Update And Range Query

```
struct Fenwick {
    vector<ll> tree;
    int n;
    Fenwick(){}
    Fenwick(int _n) {
        n = _n;
        tree = vector<ll>(n+1, 0);
    }
    void add(int i, ll val) { // arr[i] += val
        for(; i <= n; i += i&(-i)) tree[i] += val;
    }
    ll get(int i) { // arr[i]
        return sum(i, i);
    }
    ll sum(int i) { // arr[1]+...+arr[i]
        ll ans = 0;
        for(; i > 0; i -= i&(-i)) ans += tree[i];
        return ans;
    }
    ll sum(int l, int r) {// arr[l]+...+arr[r]
        return sum(r) - sum(l-1);
    }
};
```

1.3 Fenwick Tree Range Update And Point Query

```
struct Fenwick {
    vector<ll> tree;
    vector<ll> arr;
    int n;
    Fenwick(vector<ll> _arr) {
        n = _arr.size();
        arr = _arr;
        tree = vector<ll>(n+2, 0);
    }
    void add(int i, ll val) { // arr[i] += val
        for(; i <= n; i += i&(-i)) tree[i] += val;
    }
    void add(int l, int r, ll val) {// arr[l..r] += val
        add(l, val);
        add(r+1, -val);
    }
    ll get(int i) { // arr[i]
        ll sum = arr[i-1]; // zero based
        for(; i > 0; i -= i&(-i)) sum += tree[i];
        return sum; // zero based
    }
};
```

1.4 Fenwick Tree Range Update And Range Query

```
struct RangedFenwick {
    Fenwick F1, F2; // support range query and point update
    RangedFenwick(int _n) {
        F1 = Fenwick(_n+1);
        F2 = Fenwick(_n+1);
    }
    void add(int l, int r, ll v) { // arr[l..r] += v
        F1.add(l, v);
        F1.add(r+1, -v);
        F2.add(l, v*(l-1));
        F2.add(r+1, -v*r);
    }
    ll sum(int i) { // arr[1..i]
        return F1.sum(i)*i-F2.sum(i);
    }
    ll sum(int l, int r) { // arr[l..r]
        return sum(r)-sum(l-1);
    }
};
```

1.5 Implicit Treap

```
namespace ImplicitTreap {
    template <typename T>
    struct Node {
        Node* l, *r;
        ll prio, size, sum;
        T val;
        bool rev;
        Node() {}
        Node(T val) : l(nullptr), r(nullptr), val(val), size(1), sum(val), rev(false) {
            prio = rand() ^ (rand() << 15);
        }
    };
    template <typename T>
    using NodePtr = Node<T>*;

    template <typename T>
    int sz(NodePtr<T> n) {
        return n ? n->size : 0;
    }
    template <typename T>
    ll getSum(NodePtr<T> n) {
        return n ? n->sum : 0;
    }

    template <typename T>
    void push(NodePtr<T> n) {
        if (n && n->rev) {
            n->rev = false;
            swap(n->l, n->r);
            if (n->l) n->l->rev ^= 1;
            if (n->r) n->r->rev ^= 1;
        }
    }

    template <typename T>
    void recalc(NodePtr<T> n) {
        if (!n) return;
        n->size = sz(n->l) + 1 + sz(n->r);
        n->sum = getSum(n->l) + n->val + getSum(n->r);
    }

    template <typename T>
    void split(NodePtr<T> tree, ll key, NodePtr<T>& l, NodePtr<T>& r) {
        push(tree);
        if (!tree) {
            l = r = nullptr;
        }
        else if (key <= sz(tree->l)) {
            split(tree->l, key, l, tree->l);
            r = tree;
        }
        else {
            split(tree->r, key-sz(tree->l)-1, tree->r, r);
            l = tree;
        }
        recalc(tree);
    }

    template <typename T>
    void merge(NodePtr<T>& tree, NodePtr<T> l, NodePtr<T> r) {
        push(l); push(r);
        if (!l || !r) {
            tree = l ? l : r;
        }
        else if (l->prio > r->prio) {
            merge(l->r, l->r, r);
            tree = l;
        }
        else {
            merge(r->l, l, r->l);
            tree = r;
        }
        recalc(tree);
    }

    template <typename T>
    void insert(NodePtr<T>& tree, T val, int pos) {
        if (!tree) {
            tree = new Node<T>(val);
            return;
        }
        NodePtr<T> L, R;
        split(tree, pos, L, R);
        merge(L, L, new Node<T>(val));
        merge(tree, L, R);
        recalc(tree);
    }

    template <typename T>
    void reverse(NodePtr<T> tree, int l, int r) {
        NodePtr<T> t1, t2, t3;
        split(tree, l, t1, t2);
        split(t2, r - l + 1, t2, t3);
        if(t2) t2->rev = true;
        merge(t2, t1, t2);
        merge(tree, t2, t3);
    }

    template <typename T>
    void print(NodePtr<T> t, bool newline = true) {
        push(t);
        if (!t) return;
        print(t->l, false);
        cout << t->val << ” ”;
        print(t->r, false);
        if (newline) cout << endl;
    }

    template <typename T>
    NodePtr<T> fromArray(vector<T> v) {
        NodePtr<T> t = nullptr;
        FOR(i, 0, v.size()) {
            insert(t, v[i], i);
        }
        return t;
    }

    template <typename T>
    ll calcSum(NodePtr<T> t, int l, int r) {
        NodePtr<T> L, R;
        split(t, l, L, R);
        NodePtr<T> good;
        split(R, r - l + 1, good, L);
        return getSum(good);
    }
}
```

1.6 Treap

```
namespace Treap {

    struct Node {
        Node *l, *r;
        ll key, prio, size;
        Node() {}
        Node(ll key) : key(key), l(nullptr), r(nullptr), size(1) {
            prio = rand() ^ (rand() << 15);
        }
    };

    typedef Node* NodePtr;

    int sz(NodePtr n) {
        return n ? n->size : 0;
    }

    void recalc(NodePtr n) {
        if (!n) return;
        n->size = sz(n->l) + 1 + sz(n->r);
    }

    void split(NodePtr tree, ll key, NodePtr& l, NodePtr& r) {
        if (!tree) {
            l = r = nullptr;
        }
        else if (key < tree->key) {
            split(tree->l, key, l, tree->l);
            r = tree;
        }
        else {
            split(tree->r, key, tree->r, r);
            l = tree;
        }
        recalc(tree);
    }

    void merge(NodePtr& tree, NodePtr l, NodePtr r) {
        if (!l || !r) {
            tree = l ? l : r;
        }
        else if (l->prio > r->prio) {
            merge(l->r, l->r, r);
            tree = l;
        }
        else {
            merge(r->l, l, r->l);
            tree = r;
        }
        recalc(tree);
    }

    void insert(NodePtr& tree, NodePtr node) {
        if (!tree) {
            tree = node;
        }
        else if (node->prio > tree->prio) {
            split(tree, node->key, node->l, node->r);
            tree = node;
        }
        else {
            insert(node->key < tree->key ? tree->l : tree->r, node);
        }
        recalc(tree);
    }

    void erase(NodePtr tree, ll key) {
        if (!tree) return;
        if (tree->key == key) {
            merge(tree, tree->l, tree->r);
        }
        else {
            erase(key < tree->key ? tree->l : tree->r, key);
        }
        recalc(tree);
    }

    void print(NodePtr t, bool newline = true) {
        if (!t) return;
        print(t->l, false);
        cout << t->key << ” ”;
        print(t->r, false);
        if (newline) cout << endl;
    }
}
```

2 General

2.1 Automatic Test

```
# Linux Bash
# gen, main and stupid have to be compiled beforehand
for((i=1;;++i)); do
    echo $i;
    ./gen $i > genIn;
    diff <(./main < genIn) <(./stupid < genIn) || break;
done

# Windows CMD
@echo off
FOR /L %%I IN (1,1,2147483647) DO (
    echo %%I
    gen.exe %%I > genIn
    main.exe < genIn > mainOut
    stupid.exe < genIn > stupidOut
    FC mainOut stupidOut || goto :eof
)
```

2.2 C++ Template

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp> // gp_hash_table<int, int> == hash
      map
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef pair<double, double> pdd;
template <typename T> using min_heap = priority_queue<T, vector<T>, greater<
      T>>;
template <typename T> using max_heap = priority_queue<T, vector<T>, less<T
      >>;
template <typename T> using ordered_set = tree<T, null_type, less<T>,
      rb_tree_tag, tree_order_statistics_node_update>;
template <typename K, typename V> using hashmap = gp_hash_table<K, V>;

template<typename A, typename B> ostream& operator<<(ostream& out, pair<A, B
      > p) { out << ”(” << p.first << ”, ” << p.second << ”)”; return out;}
template<typename T> ostream& operator<<(ostream& out, vector<T> v) { out
      << ”[”; for(auto& x : v) out << x << ”, ”; out << ”]”;return out;}
template<typename T> ostream& operator<<(ostream& out, set<T> v) { out << ”
      {”; for(auto& x : v) out << x << ”, ”; out << ”}”; return out; }
template<typename K, typename V> ostream& operator<<(ostream& out, map<K,
      V> m) { out << ”{”; for(auto& e : m) out << e.first << ” -> ” << e.second
      << ”, ”; out << ”}”; return out; }
template<typename K, typename V> ostream& operator<<(ostream& out, hashmap
      <K, V> m) { out << ”{”; for(auto& e : m) out << e.first << ” -> ” << e.
      second << ”, ”; out << ”}”; return out; }

#define FAST_IO ios_base::sync_with_stdio(false); cin.tie(NULL)
#define TESTS(t) int NUMBER_OF_TESTS; cin >> NUMBER_OF_TESTS; for(
      int t = 1; t <= NUMBER_OF_TESTS; t++)
#define FOR(i, begin, end) for (int i = (begin) - ((begin) > (end)); i != (end) - ((
      begin) > (end)); i += 1 - 2 * ((begin) > (end)))
#define sgn(a) ((a) > eps ? 1 : ((a) < -eps ? -1 : 0))
#define precise(x) fixed << setprecision(x)
#define debug(x) cerr << ”> ” << #x << ” = ” << x << endl;
#define pb push_back
#define rnd(a, b) (uniform_int_distribution<int>((a), (b))(rng))
#ifndef LOCAL
    #define cerr if(0)cout
    #define endl ”\n”
#endif
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
clock_t __clock__;
```

```
void startTime() {___clock___ = clock();}
void timeit(string msg) {cerr << ”> ” << msg << ”: ” << precise(6) << ld(clock()-___clock___)/CLOCKS_PER_SEC << endl;}
const ld PI = asin(1) * 2;
const ld eps = 1e-14;
const int oo = 2e9;
const ll OO = 2e18;
const ll MOD = 1000000007;
const int MAXN = 1000000;

int main() {
    FAST_IO;
    startTime();

    timeit(”Finished”);
    return 0;
}
```

2.3 Compilation

```
# Simple compile
g++ -DLOCAL -O2 -o main.exe -std-c++17 -Wall -Wno-unused-result -Wshadow main.cpp
# Debug
g++ -DLOCAL -std=c++17 -Wshadow -Wall -o main.exe main.cpp -fsanitize=address -fsanitize=undefined -fuse-ld=gold -D_GLIBCXX_DEBUG -g
```