Code Template for ACM-ICPC

kesarPista

March 29, 2023

Contents

1 Template

1.1 base.cpp

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define f first
#define s second
#define all(x) (x).begin(), (x).end()
#define sz(x) (int)(x).size()

template<typename A, typename B> ostream&
    operator<<(ostream &os, const pair<A, B> &p) {
    return os << '(' << p.f << ", " << p.s << ')'; }
template<typename T_container, typename T = typename
    enable_if<!is_same<T_container, string>::value,
    typename T_container::value_type>::type> ostream&
    operator<<(ostream &os, const T_container &v) { os
    << '{'; string sep; for (const T &x : v) os << sep
    << x, sep = ", "; return os << '}'; }
void dbg_out() { cerr << endl; }
template<typename Head, typename... Tail> void
    dbg_out(Head H, Tail... T) { cerr << ' ' << H;
    dbg_out(T...); }

#ifdef LOCAL
#define dbg(...) cerr << "(" << #__VA_ARGS__ << "):",
    dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
#ifdef LOCAL
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
#endif
    return 0;
}
```

1.2 cprun.sh

```
# alias cprun="/home/cprun.sh"
PROG_NAME=$1
OUT_NAME="${PROG_NAME%.*}"
g++ -std=c++17 -DLOCAL $PROG_NAME -o $OUT_NAME.out
if [[ $? == 0 ]]; then
    ./$OUT_NAME.out
fi
```

1.3 .vimrc

```
set nocp
filetype plugin indent on
set number
set relativenumber
syntax on
colorscheme ron
let mapleader = " "
set tabstop=4
set shiftwidth=4
set softtabstop=4
set autoindent
set expandtab
if has('persistent_undo')
    set undodir=$HOME/.vim/undo
    set undofile
endif
set ignorecase
set smartcase
set incsearch
nnoremap <leader>t :term<CR>
inoremap { {}<Esc>ha
inoremap ( ()<Esc>ha
inoremap [ []<Esc>ha
inoremap " ""<Esc>ha
inoremap ' ''<Esc>ha
inoremap ` ``<Esc>ha
```

2 Number Theory

3 Range-Query

3.1 segtree.cpp

```
template<typename Info>
class SegTree {
public:
    int n;
    vector<Info> info;

    SegTree(int _n) {
        n = _n;
        info = vector<Info>(4 * n + 1);
    }

    SegTree(const vector<Info> &init) :
         SegTree(init.size()) {
        function<void(int, int, int)> build = [&](int
            p, int l, int r) -> void {
            if(l == r) {
                info[p] = init[l];
                return;
            }
            int m = l + (r - l) / 2;
            build(2 * p + 1, l, m);
            build(2 * p + 2, m + 1, r);
        };
        build(0, 0, n - 1);
    }

    void pull(int p) {
        info[p] = info[2 * p + 1] + info[2 * p + 2];
    }

    void modify(int p, int l, int r, int x, const Info&
         v) {
        if(r == l) {
            info[p] = v;
            return;
        }

        int m = l + (r - l) / 2;
        if(x <= m) {
            modify(2 * p + 1, l, m, x, v);
        } else {
            modify(2 * p + 2, m + 1, r, x, v);
        }
        pull(p);
    }

    void modify(int p, const Info& v) {
        modify(0, 0, n - 1, p, v);
    }

    Info rangeQuery(int p, int l, int r, int x, int y) {
        if(y < l or r < x) {
            return Info();
        }

        if(x <= l and r <= y) {
            return info[p];
        }

        int m = l + (r - l) / 2;
        return rangeQuery(2 * p + 1, l, m, x, y) +
             rangeQuery(2 * p + 2, m + 1, r, x, y);
    }

    Info rangeQuery(int l, int r) {
        return rangeQuery(0, 0, n - 1, l, r);
    }
};

class Sum {
public:
    long long x = 0;
    int index = -1;

    Sum() {}

    Sum(long long _x) {
        x = _x;
    }

    Sum(long long _x, int _index) {
        x = _x;
        index = _index;
    }

};

Sum operator+(const Sum &lf, const Sum &rt) {
    return Sum(lf.x + rt.x);
}

class Solution {
public:
    long long numberOfPairs(vector<int>& nums1,
         vector<int>& nums2, int diff) {
        const int n = (int)nums1.size();
        SegTree<Sum> st(1e5 + 2);

        int shift = 1e4;
        long long ans = 0;
        for(int i = 0; i < n; i++) {
            int lf = nums1[i] - nums2[i] + 2 * shift;
            ans += st.rangeQuery(0, lf + diff).x;
            int val = st.rangeQuery(lf, lf).x;
            st.modify(lf, Sum(val + 1));
        }
        return ans;
    }
};
```

3.2 lazysegtree.cpp

```
template<typename Info, typename Tag>
class LazySegTree {
public:
    int n;
    vector<Info> info;
    vector<Tag> tag;

    LazySegTree(int _n) {
        n = _n;
        info = vector<Info>(4 * n + 1);
        tag = vector<Tag>(4 * n + 1);
    }

    LazySegTree(const vector<Info> &init) :
         LazySegTree(init.size()) {
        function<void(int, int, int)> build = [&](int
             p, int l, int r) -> void {
            if(r == l) {
                info[p] = init[l];
                return;
            }
            int m = l + (r - l) / 2;
            build(2 * p + 1, l, m);
            build(2 * p + 2, m + 1, r);
        };
        build(0, 0, n - 1);
    }

    void pull(int p) {
        info[p] = info[2 * p + 1] + info[2 * p + 2];
    }

    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }

    void push(int p) {
        apply(2 * p + 1, tag[p]);
        apply(2 * p + 2, tag[p]);
        tag[p] = Tag();
    }

    void modify(int p, int l, int r, int x, const Info
         &v) {
        if(r == l) {
            info[p] = v;
            return;
        }

        int m = l + (r - l) / 2;
        push(p);
        if(x <= m) {
            modify(2 * p + 1, l, m, x, v);
        } else {
            modify(2 * p + 2, m + 1, r, v);
        }
        pull(p);
    }

    void modify(int p, const Info &v) {
        modify(0, 0, n - 1, p, v);
    }

    Info rangeQuery(int p, int l, int r, int x, int y) {
        if(y < l or r < x) {
            return Info();
        }

        if(x <= l and r <= y) {
            return info[p];
        }

        int m = l + (r - l) / 2;
        push(p);
        return rangeQuery(2 * p + 1, l, m, x, y) +
             rangeQuery(2 * p + 2, m + 1, r, x, y);
    }

    Info rangeQuery(int l, int r) {
        return rangeQuery(0, 0, n - 1, l, r);
    }

    void rangeApply(int p, int l, int r, int x, int y,
         const Tag &v) {
        if(y < l or r < x) {
            return;
        }

        if(x <= l and r <= y) {
            apply(p, v);
            return;
        }

        int m = l + (r - l) / 2;
        push(p);
        rangeApply(2 * p + 1, l, m, x, y, v);
        rangeApply(2 * p + 2, m + 1, r, x, y, v);
        pull(p);
    }

    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(0, 0, n - 1, l, r, v);
    }
};

class Tag {
public:
    long long x = 0;

    Tag() {}

    Tag(long long _x) {
        x = _x;
    }

    void apply(const Tag &t) {
        x += t.x;
    }
};

class Sum {
public:
    long long x = 0;
    int index = -1;

    Sum() {}

    Sum(long long _x) {
        x = _x;
    }

    Sum(long long _x, int _index) {
        x = _x;
        index = _index;
    }

    void apply(const Tag &t) {
        x += t.x;
    }
};

Sum operator+(const Sum &lf, const Sum &rt) {
    return Sum(lf.x + rt.x);
}

class Solution {
public:
    string shiftingLetters(string s,
         vector<vector<int>>& shifts) {
        const int n = (int)s.size();
        LazySegTree<Sum, Tag> st(n);

        for(const auto& v: shifts) {
            st.rangeApply(v[0], v[1], Tag(v[2]? +1: -1));
        }
        for(int i = 0; i < n; i++) {
            int val = st.rangeQuery(i, i).x;
            s[i] = (((s[i] - ’a’ + val) % 26 + 26) % 26)
                 +’a’;
        }
        return s;
    }
};
```

3.3 bit.cpp

```
template<typename T>
class BIT {
public:
    vector<T> bit;
    int n;

    BIT() {
        n = 0;
    }

    BIT(int _n) {
        n = _n;
        bit.assign(n, 0);
    }

    void inc(int idx, T val) {
        assert(0 <= idx and idx < n);
        for(int i = idx + 1; i <= n; i += (i & -i))
            bit[i - 1] += val;
    }
    T query(int idx) {
        assert(0 <= idx and idx < n);
        T res = 0;
        for(int i = idx + 1; i > 0; i -= (i & -i))
            res += bit[i - 1];
        return res;
    }

    T at(int idx) {
        assert(0 <= idx and idx < n);
        return query(idx) - (idx - 1 >= 0? query(idx -
             1): 0);
    }

    T at(int l, int r) {
        assert(0 <= l and l <= r and r < n);
        return query(r) - (l - 1 >= 0? query(l - 1): 0);
    }
};

template<typename T>
class FT {
public:
    BIT<T> f1, f2;
    int n;

    FT() {
        n = 0;
    }

    FT(int _n) {
        n = _n;
        f1 = f2 = BIT<T>(_n + 1);
    }

    void inc(int idx, T val){
        assert(0 <= idx and idx < n);
        inc(idx, idx, val);
    }

    void inc(int l, int r, T val) {
        assert(0 <= l and l <= r and r < n);
        f1.inc(l, val);
        f1.inc(r + 1, -val);
        f2.inc(l, val * (l - 1));
        f2.inc(r + 1, -val * r);
    }

    T query(int idx) {
        assert(0 <= idx and idx < n);
        return f1.query(idx) * idx - f2.query(idx);
    }

    T at(int idx) {
        assert(0 <= idx and idx < n);
        return query(idx) - (idx - 1 >= 0? query(idx -
             1): 0);
    }

    T at(int l, int r) {
        assert(0 <= l and l <= r and r < n);
        return query(r) - (l - 1 >= 0? query(l - 1): 0);
    }
};
```

3.4 rmq.cpp

```
template<typename T>
class RMQ {
public:
    vector<vector<T>> st;
    function<T(T, T)> op;
    int n, m;

    RMQ(const vector<T>& a, function<T(T, T)> _op) {
        n = (int)a.size(); m = __lg(n) + 1; //
             ceil(log(r - l + 1))
        st = vector<vector<T>>(n, vector<T>(m));
        op = _op;
        int p = 1;
        for(int j = 0; j < m; j++) {
            for(int i = 0; i + p - 1 < n; i++) {
                if(j == 0) st[i][j] = a[i];
                else st[i][j] = op(st[i][j - 1], st[i +
                     (p >> 1)][j - 1]);
            }
            p *= 2;
        }
    }

    T query(int l, int r) {
        int sz = __lg(r - l + 1); // floor(log(r - l +
             1))
        return op(st[l][sz], st[r + 1 - (1 << sz)][sz]);
    }
}; // RMQ<int> rmq(arr, [](int x, int y) -> int {
    return min(x, y); });
```

4 String

4.1 trie.cpp

```
class Node {
public:
    static const int N = (1 << 8); // 256
    bool is_leaf;
    int next[N];
    int count;

    Node() {
        is_leaf = false;
        count = 0;
        memset(next, -1, sizeof next);
    }
};

class Trie {
public:
    vector<Node> root;

    Trie() {
        root.emplace_back(); // insert a object
    }

    void insert(string word) {
        int i = 0;
        for (char c: word) {
            if (root[i].next[c] == -1) {
                root[i].next[c] = root.size();
                root.emplace_back();
            }
            i = root[i].next[c];
        }
        root[i].is_leaf = true;
    }

    bool search(string word) {
        int i = 0;
        for (char c: word) {
            if (root[i].next[c] == -1) {
                return false;
            }
            i = root[i].next[c];
        }
        return root[i].is_leaf;
    }

    bool startsWith(string prefix) {
        int i = 0;
        for (char c: prefix) {
            if (root[i].next[c] == -1) {
                return false;
            }
            i = root[i].next[c];
        }
        return true;
    }
};
```

5 Graph

5.1 bridges.cpp

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

5.2 bridges-online.cpp

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}

int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] =
        find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] =
        find_cc(dsu_cc[v]);
}

void make_root(int v) {
    v = find_2ecc(v);
    int root = v;
    int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a;
                break;
            }
            last_visit[a] = lca_iteration;
            a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b;
                break;
                }
            last_visit[b] = lca_iteration;
            b = par[b];
        }

    }

    for (int v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
    for (int v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
}

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;

    int ca = find_cc(a);
    int cb = find_cc(b);

    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
```

5.3 articulation.cpp

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

5.4 scc.cpp

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;
    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);
    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...
    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }
    used.assign(n, false);
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());
    for (auto v : order)
        if (!used[v]) {
            dfs2 (v);
            // ... processing next component ...
            component.clear();
        }
}
```

5.5 binlift.cpp

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p) {
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

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

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

6 Miscellaneous

6.1 mat.cpp

```
template<typename T>
class Matrix {
public:
    vector<vector<T>> mat;

    Matrix() {}

    Matrix(int _n, int _m, T init) {
        mat = vector<vector<T>>(_n, vector<T>(_m,
            init));
    }

    Matrix(const vector<vector<T>>& a) {
        mat = a;
    }

    void set(T init) {
        for(int i = 0; i < rows(); i++) {
            for(int j = 0; j < cols(); j++) {
                mat[i][j] = init;
            }
        }
    }

    int rows() const {
        return mat.size();
    }

    int cols() const {
        return mat[0].size();
    }

    Matrix operator*(const Matrix<T>& obj) const {
        assert(cols() == obj.rows());
        vector<vector<T>> res(rows(),
            vector<T>(obj.cols()));
        for(int r = 0; r < rows(); r++) {
            for(int c = 0; c < obj.cols(); c++) {
                for(int k = 0; k < cols(); k++) {
                    res[r][c] += mat[r][k] *
                        obj.mat[k][c];
                }
            }
        }
        return res;
    }

    vector<T>& operator[](int index) {
        return mat[index];
    }

};

template<typename T>
ostream& operator<<(ostream& os, Matrix<T>& mat) {
    int n = mat.rows(), m = mat.cols();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            os << mat[i][j] << " ";
        }
        os << endl;
    }
    return os;
}

template<typename T>
Matrix<T> power(Matrix<T> res, Matrix<T> a, long long
    b) {
    while(b > 0) {
        if(b & 1) res = a * res;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```