NTJ
UDESC

Eric Grochowicz, Enzo de Almeida Rodrigues e João Marcos de Oliveira

8 de Março de 2023

Índice

1 Estruturas

1.1 Fenwick Tree

```
// Processas queries de operacao com inverso
// Build: O(n)
// Query: O(log(n))
// Update: O(log(n))

typedef long long ll;

struct fenwick {
    vector<ll> bit;
    fenwick(int n) { bit.assign(n+1, 0); }
    fenwick(vector<ll>& v) {
        int n = v.size();
        bit.assign(n+1, 0);
        for(int i = 1; i <= n; i++) bit[i] = v[i-1];
        for(int i = 1, j = 2; i <= n; i++, j = i + (i &
            -i)) if(j <= n) {
            bit[j] += bit[i];
        }
    }
    ll query(int i){
        ll res = 0;
        for(; i; i -= (i & -i))
            res += bit[i];
        return res;
    }
    ll query(int l, int r){
        return query(r) - query(l-1);
    }
    void update(int i, ll d){
        for(; i && i < (int)bit.size(); i += (i & -i))
            bit[i] += d;
    }
};
```

2 Grafos

2.1 Bridges e Edge Biconnected Components

```cpp
// Acha todas as pontes em O(n)
// Tambem constroi a arvore condensada, mantendo
// so as pontes como arestas e o resto comprimindo
// em nodos

const int maxn = 4e5;
int n, m;
bool vis[maxn];
int dp[maxn], dep[maxn];
vector<int> adj[maxn];
vector<ii> bridges;

void dfs_dp(int u, int p = -1, int d = 0){
    dp[u] = 0, dep[u] = d, vis[u] = 1;
    for(auto v : adj[u]) if(v != p) {
        if(vis[v]){
            if(dep[v] < dep[u]) dp[v]--, dp[u]++;
        } else {
            dfs_dp(v, u, d+1);
            dp[u] += dp[v];
        }
    }
    if(dp[u] == 0 && p != -1){ // edge {u, p} eh uma
       ponte
        bridges.emplace_back(u, p);
    }
}

void find_bridges(){
    memset(vis, 0, n+1);
    for(int i = 1; i <= n; i++){
        if(!vis[i]) dfs_dp(i);
    }
}

// EDGE BICONNECTED COMPONENTS (requer todo codigo acima)
int ebcc[maxn], ncc = 1;
vector<int> adjbcc[maxn];

void dfs_ebcc(int u, int p = -1, int cc = 1){
    vis[u] = 1;
    if(dp[u] == 0 && p != -1){
        cc = ++ncc;
    }
    ebcc[u] = cc;
    for(auto v : adj[u]) if(!vis[v]) {
        dfs_ebcc(v, u, cc);
    }
}

void build_ebcc_graph(){
    find_bridges();
    memset(vis, 0, n+1);
    for(int i = 1; i <= n; i++){
        if(!vis[i]) dfs_ebcc(i);
    }
    // Opcao 1 - constroi o grafo condensado passando
       por todas as edges
    for(int u = 1; u <= n; u++){
        for(auto v : adj[u]){
            if(ebcc[u] != ebcc[v]){
                adjbcc[ebcc[u]].emplace_back(ebcc[v]);
            } else {
                // faz algo
            }
        }
    }
    // Opcao 2 - constroi o grafo condensado passando so
       pelas pontes
    for(auto [u,v] : bridges){
        adjbcc[ebcc[u]].emplace_back(ebcc[v]);
        adjbcc[ebcc[v]].emplace_back(ebcc[u]);
    }
}
```

3 Extra

3.1 template.cpp

```cpp
// Templace C++

#include <bits/stdc++.h>

using namespace std;

void solve(){

}

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    solve();
}
```