NTJ
UDESC

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

19 de agosto de 2023

Índice

1 Problemas

1.1 Kth digito na string infinita de digitos

```
// Retorna qual o numero e qual o algarismo do Kth digito
// na string infinita dos numeros naturais
   (12345678910111213...)
// Complexidade: O(log_10(k))

pair<ll,ll> kthdig(ll k){
    ll qtd = 1, num_alg = 1, base = 1;
    while(1){
        ll add = (9 * base) * num_alg;
        if(qtd + add < k){
            qtd += add;
        } else break;
        base *= 10, num_alg++;
    }
    ll algarismo = (k - qtd) % num_alg;
    ll numero = (k - qtd) / num_alg + base;
    return {numero, algarismo};
}
```

2 Estruturas

2.1 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; i <= n; i++) {
            int 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;
    }
};
```

3 Grafos

3.1 Binary Lifting

```
// Binary Lifting (em nodos)
//
// Computa LCA e tambem resolve queries de operacoes
// associativas e comutativas em caminhos. Para operacoes
// nao comutativas, modificar funcao query ou usar HLD.
//
// Build(): O(n log(n))
// Query(): O(log(n))
// Lca(): O(log(n))
//
// up[u][i] = (2 ^ i)-esimo pai do u
// st[u][i] = query ate (2 ^ i)-esimo pai do u (NAO
   INCLUI O U)

const int maxn = 1e5 + 5, LG = 20;
vector<int> adj[maxn];

struct BinaryLifting {
    int up[maxn][LG], st[maxn][LG], val[maxn], t = 1;
    int tin[maxn], tout[maxn];

    const int neutral = 0;
    int merge(int l, int r) { return l + r; }

    void build(int u, int p = -1) {
        tin[u] = t++;
        for (int i = 0; i < LG - 1; i++) {
            up[u][i + 1] = up[up[u][i]][i];
            st[u][i + 1] = merge(st[u][i],
               st[up[u][i]][i]);
        }
        for (int v : adj[u]) if (v != p) {
            up[v][0] = u, st[v][0] = val[u];
            build(v, u);
        }
        tout[u] = t++;
    }

    void build(int root, vector<int> &v) {
        int N = size(v);
        for (int i = 0; i < N; i++) val[i] = v[i];
        build(root);
    }

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

    int query2(int u, int v, bool include_lca) {
        if (ancestor(u, v)) return include_lca ? val[u]
           : neutral;
        int ans = val[u];
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                ans = merge(ans, st[u][i]);
                u = up[u][i];
            }
        }
        return include_lca ? merge(ans, st[u][0]) : ans;
    }

    int query(int u, int v) {
        if (u == v) return val[u];
        return merge(query2(u, v, 1), query2(v, u, 0));
    }

    int lca(int u, int v) {
        if (ancestor(u, v)) return u;
        if (ancestor(v, u)) return v;
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                u = up[u][i];
            }
        }
        return up[u][0];
    }

} bl;

// Binary Lifting (em arestas)
```

```
//
// up[u][i] = (2 ^ i)-esimo pai do u
// st[u][i] = query ate (2 ^ i)-esimo pai do u

const int maxn = 1e5 + 5, LG = 20;
vector<pair<int, int>> adj[maxn];

struct BinaryLifting {
    int up[maxn][LG], st[maxn][LG], t = 1;
    int tin[maxn], tout[maxn];

    const int neutral = 0;
    int merge(int l, int r) { return l + r; }

    void build(int u, int p = -1) {
        tin[u] = t++;
        for (int i = 0; i < LG - 1; i++) {
            up[u][i + 1] = up[up[u][i]][i];
            st[u][i + 1] = merge(st[u][i],
                st[up[u][i]][i]);
        }
        for (auto [w, v] : adj[u]) if (v != p) {
            up[v][0] = u, st[v][0] = w;
            build(v, u);
        }
        tout[u] = t++;
    }

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

    int query2(int u, int v) {
        if (ancestor(u, v)) return neutral;
        int ans = neutral;
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                ans = merge(ans, st[u][i]);
                u = up[u][i];
            }
        }
        return merge(ans, st[u][0]);
    }

    int query(int u, int v) {
        if (u == v) return neutral;
#warning TRATAR ESSE CASO ACIMA
        return merge(query2(u, v), query2(v, u));
    }

} bl;
```

3.2 Bridges e Edge Biconnected Components

```
// Acha todas as pontes em O(n)
// Tambem constroi a arvore condensada, mantendo
// so as pontes como arestas e o resto comprimindo
// em nodos
//
// Salva no vetor bridges os pares {u, v} cujas arestas
   sao pontes

typedef pair<int, int> ii;
const int maxn = 2e5 + 5;
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);
    for (int i = 0; i < n; i++) if (!vis[i]) {
        dfs_dp(i);
    }
}

// Edge Biconnected Components (requer todo codigo acima)

int ebcc[maxn], ncc = 0;
vector<int> adjbcc[maxn];

void dfs_ebcc(int u, int p, int cc) {
    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);
    for (int i = 0; i < n; i++) if (!vis[i]) {
        dfs_ebcc(i, -1, ncc);
        ++ncc;
    }
    // Opcao 1 - constroi o grafo condensado passando
       por todas as edges
    for (int u = 0; 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]);
    }
}
```

4 Extra

4.1 Config do Vim

```
// .vimrc

set nu
set ai
set ts=4
set sw=4
filetype plugin indent on
inoremap {} {}<Left><Return><Up><End><Return>

set nohls
set belloff=all
syntax on
set expandtab
set noshiftround
set showmode
set showcmd
```

4.2 Gerador aleatorio de inteiros em [l, r]

```
mt19937 rng(chrono::steady_clock::now()
    .time_since_epoch().count());

ll uniform(ll l, ll r){
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}
```

4.3 Rand C++

```
mt19937 rng(chrono::steady_clock::now()
    .time_since_epoch().count());
```

4.4 Script de stress test

```
set -e
g++ -O2 code.cpp -o code
g++ -O2 brute.cpp -o brute
g++ -O2 gen.cpp -o gen

for((i = 1; ; ++i)); do
    ./gen > input_file
    ./code < input_file > myAnswer
    ./brute < input_file > correctAnswer
    diff myAnswer correctAnswer > /dev/null || break
    echo "Passed test: "  $i
done

echo "WA on the following test:"
cat input_file
echo "Your answer is:"
cat myAnswer
echo "Correct answer is:"
cat correctAnswer
```

4.5 Script pra rodar C++

```
// chmod +x run
// ./run A.cpp

#!/bin/bash
g++ --std=c++20 -Wall -O2 -fsanitize=address,undefined
    $1 && ./a.out
```

4.6 Template C++

```
#include <bits/stdc++.h>
#define endl '\n'

using namespace std;
typedef long long ll;
```

```
void solve(){

}

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

4.7 Template de debug simples

```
void _print() { }
template<typename T, typename... U> void _print(T a,
   U... b) {
    if(sizeof... (b)){
        cerr << a << ", ";
        _print(b...);
    } else cerr << a;
}
#define debug(x...) cerr << "[" << #x << "] = [",
   _print(x), cerr << "]" << endl

// #define debug(...)
```