UERJ - Universidade do Estado do Rio de Janeiro

Códigos de Maratona de Programação

UERJ++

(Jaime Lay, Vinicius Sathler, Paulo Victor)

coach
Paulo Eustáquio

September 21, 2019

Contents

Chapter 1

C++ Tricks e Template

1.1 Template

```
#include <bits/stdc++.h>

using namespace std;

#define DEBUG(x) cout << (#x) << " == " << x << endl;

const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const long long INFLL = 0x3f3f3f3f3f3f3f3fLL;
const long double EPS = 1e-9, PI = acos(-1.0);

int main(void){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

1.2 Tricks

```
// Acha o menor valor em um array de tamanho n.
int menor = *min_element(a,a+n);
// Conta quantos numeros 3 aparecem no vector v.
int qntd = count(v.begin(),v.end(),3);
// Calcula o MDC entre dois numeros. OBS: Tomar cuidado com long long.
int mdc = __gcd(a,b);
// Calcula o numero de 1's na representacao binaria do x.
// OBS: Usar __builtin_popcountll quando x for long long;
int qntdDeUm = __builtin_popcount(x);
// Soma todos os valores do vetor com o valorInicial
int soma = accumulate(vetor.begin(), vetor.end(), valorInicial);
// Preenche o vetor acumulando o valorInicial
iota(vetor.begin(), vetor.end(), valorInicial);
// Preenche o vetor com o valor
fill(vetor.begin(), vetor.end(), valor);
```

Chapter 2

Data Structures

2.1 Binary Indexed Tree

```
// Complexidade:- update -> O(logN)
//               - query  -> O(logN)

#define MAXN 1010

int a[MAXN], bit[MAXN], n;

void init(){
    for(int i = 1; i <= n; i++)
        update(i, a[i]);
}

void update(int x, int val){
    for(int i = x; i <= n; i += i & -i)
        bit[i] += val;
}

int query(int x){
    int sum = 0;
    for(int i = x; i > 0; i -= i & -i)
        sum += bit[i];
    return sum;
}

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

2.2 Binary Indexed Tree 2D

```
// Complexidade:- update -> O(logN)
//               - query  -> O(logN)

#define MAXN 1010

int bit[MAXN][MAXN], a[MAXN][MAXN], x, y;

void update(int idx, int idx2, int val){
    for(int i = idx; i <= x; i += i & -i){
        for(int j = idx2; j <= y; j += j & -j){
            bit[i][j] += val;
        }
    }
}

void query(int idx, int idx2){
    int sum = 0;
    for(int i = idx; i > 0; i -= i & -i){
        for(int j = idx2; j > 0; j -= j & -j){
            sum += bit[i][j];
        }
    }
    return sum;
}

void query(int xmin, int ymin, int xmax, int ymax){
    if(xmin > xmax) swap(xmin, xmax);
    if(ymin > ymax) swap(ymin, ymax);
```

```
    return query(xmax, ymax) - query(xmax, ymin - 1) - query(xmin - 1, ymax) + query(xmin - 1, ymin -
        1);
}
```

2.3 Inversion Count w/ Merge Sort

```
int invCountMS(vector<int> &v){
    int inv = 0;

    if(v.size() == 1) return 0;

    vector<int> m1, m2;
    for(int i = 0; i < v.size()/2; i++) m1.push_back(v[i]);
    for(int i = v.size()/2; i < v.size(); i++) m2.push_back(v[i]);

    inv += countInvMS(m1);
    inv += countInvMS(m2);

    m1.push_back(INT_MAX);
    m2.push_back(INT_MAX);

    int idx = 0, idx2 = 0;
    for(int i = 0; i < v.size(); i++){
        if(m1[idx] <= m2[idx2]){
            v[i] = m1[idx++];
        }else{
            v[i] = m2[idx2++];
            inv += m1.size() - idx - 1;
        }
    }
    return inv;
}
```

2.4 Ordered Set

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
template <class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

Tree<int> ord_s;
ord_s.insert(2);
ord_s.insert(5);
ord_s.insert(7);
ord_s.insert(9);

// find_by_order returns an iterator to the element at a given position
auto x = ord_s.find_by_order(2);
cout << *x << "\n"; // 7

// order_of_key returns the position of a given element
cout << ord_s.order_of_key(7) << "\n"; // 2

// If the element does not appear in the set, we get the position that the element would have in the
    set
cout << ord_s.order_of_key(6) << "\n"; // 2
cout << ord_s.order_of_key(8) << "\n"; // 3
```

2.5 Segment Tree

```
const int MAXN = 112345;

int st[4*MAXN], a[MAXN];

void build(int node, int left, int right){
    if(left == right){
        st[node] = a[left];
    }else{
        int mid = (left + right)/2;
        build(2*node, left, mid);
        build(2*node + 1, mid + 1, right);

        st[node] = st[2*node] * st[2*node + 1];
    }
```

```
}

void update(int node, int left, int right, int idx, int val){
    if(idx > right || idx < left){
        return;
    }else if(left == right){
        a[idx] = val;
        st[node] = val;
    }else{
        int mid = (left + right)/2;
        update(2*node, left, mid, idx, val);
        update(2*node + 1, mid + 1, right, idx, val);

        st[node] = st[2*node] * st[2*node + 1];
    }
}

int query(int node, int left, int right, int a, int b){
    if(b < left || a > right){
        return 1;
    }else if(left >= a && right <= b){
        return st[node];
    }else{
        int mid = (left + right)/2;
        int e = query(2*node, left, mid, a, b);
        int d = query(2*node + 1, mid + 1, right, a, b);
        return e * d;
    }
}
```

2.6 Segment Tree

```
#define MAXN 112345

int st[4*MAXN], lazy[4*MAXN], a[MAXN];

void build(int node, int left, int right){
    if(left == right){
        st[node] = a[left];
    }else{
        int mid = (left + right)/2;
        build(2*node, left, mid);
        build(2*node + 1, mid + 1, right);

        st[node] = st[2*node] + st[2*node + 1];
    }
}

void propagation(int node, int left, int right){
    if(lazy[node] != -1){
        st[node] = lazy[node];
        if(left != right){
            lazy[2*node] = lazy[node];
            lazy[2*node + 1] = lazy[node];
        }
        lazy[node] = -1;
    }
}

void update(int node, int left, int right, int a, int b, int val){
    propagation(node, left, right);
    if(a < left || b > right){
        return;
    }else if(left == right){
        lazy[node] = val;
        propagation(node, left, right);
    }else{
        int mid = (left + right)/2;
        update(2*node, left, mid, a, b, val);
        update(2*node + 1, mid + 1, right, a, b, val);

        st[node] = st[2*node] + st[2*node + 1];
    }
}

int query(int node, int left, int right, int a, int b){
    propagation(node, left, right);
    if(a < left || b > right){
        return 0;
    }else if(left == right){
```

```
        return st[node];
    }else{
        int mid = (left + right)/2;
        return query(2*node, left , mid, a, b) + query(2*node + 1, mid + 1, right , a, b);
    }
}
```

2.7 Union-find set

```
int pai[1000001], n;

void init(){
    for(int i = 1; i <= n; i++)
        pai[i] = i;
}

int find(int v){
    if(pai[v] == v)
        return v;
    return pai[v] = find(pai[v]);
}

bool unions(int u, int v){
    u = find(u); v = find(v);
    if(u == v) return false;
    pai[u] = v;                          // Faz com que o 'v' seja pai de 'u'
}
```

Chapter 3

Graphs

3.1 Breadth-First Search

```
vector<int> adj[n];
int dist[n];

void BFS(){
    memset(dist, -1, sizeof(dist));
    queue<int> q;
    dist[1] = 0;
    q.push(1);

    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(int v : adj[u]){
            if(dist[v] == -1 || dist[v] > dist[u] + 1){
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

3.2 Depth-First Search

```
vector<int> g[n];
int vis[n];

int tam = 0;

void DFS(int v){
    vis[v] = 1;
    for(auto u : g[v])
        if(!vis[v]) DFS(u);
}

// Cobertura Minima
// 0 = Nao visitado, 1 = Visitado, 2 = Vertice da cobertura
void DFS(int v){
    vis[v] = 1;
    for(auto u : g[v]){
        if(vis[u] == 0){
            DFS(u);
            if(vis[u] == 1) vis[v] = 2;
        }
    }
}

// O vertice u esta conectado com dest?
bool isConnect(int u, int dest){
    vis[u] = true;
    if(u == dest) return true;
    for(auto v : adj[u])
        if(connect(v, dest)) return true;
    return false;
}
```

3.3 Articulation Points

```
int vis[MAXN], low[MAXN], ap[MAXN], cont;

void articulation_point(int p, int v){
    vis[v] = low[v] = ++cont;
    for(auto u : g[v]){
        if(!vis[u]){
            articulation_point(v, u);
            if(low[u] >= vis[v]) ap[v]++;
            low[v] = min(low[v], low[u]);
        }else if(u != p){
            low[v] = min(low[v], vis[u]);
        }
    }
}

vector<int> isArticulationPoint(){
    vector<int> points;
    for(int i = 1; i <= n; i++){
        if(!vis[i]) articulation_point(i, i);
    }

    for(int i = 1; i <= n; i++){
        if(i == 1 && ap[1] > 1) points.push_back(1);
        else if(i != 1 && ap[i] > 0) points.push_back(i);
    }

    return points;
}

void init(){
    cont = 0;
    memset(vis, 0, sizeof(vis));
    memset(low, 0, sizeof(low));
    memset(ap, 0, sizeof(ap));
    memset(g, 0, sizeof(g));
}
```

3.4 Topological Sort

```
int vis[MAXN], dist[MAXN], maior, end_point;
vector<int> g[MAXN];
stack<int> topoSort;

void topological_sort(int v){
    vis[v] = 1;
    for(auto u : g[v]){
        if(!vis[u]) topological_sort(u);
    }
    topoSort.push(v);
}

void longest_path(int src){
    dist[src] = 0;
    maior = 0;
    while(!topoSort.empty()){
        int v = topoSort.top();
        topoSort.pop();
        if(dist[v] != -1){
            for(auto u : g[v]){
                dist[u] = max(dist[u], dist[v] + 1);
            }
        }
        maior = max(maior, dist[v]);
    }
    for(int i = 1; i <= n; i++){
        if(maior == dist[i]){
            end_point = i;
            break;
        }
    }
}
```

Chapter 4

Strings

4.1 Z-Function

```
// Retorna um vector Z, onde cada Z[i] mostra a maior substring comecando do i que tambem eh prefixo da
    string.

vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    z[0] = s.size();
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

Chapter 5

Miscellaneous

5.1 Histogram Problem

```
// Funcao que retorna a maior area retangular de um histograma.

long long getMaxArea(vector<long long> &hist, long long n){
    stack<long long> s;

    long long max_area = 0;
    long long tp;
    long long area_with_top;

    long long i = 0;
    while(i < n){
        if(s.empty() || hist[s.top()] <= hist[i]){
            s.push(i++);
        }else{
            tp = s.top();
            s.pop();
            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
            if(max_area < area_with_top)
                max_area = area_with_top;
        }
    }

    while(!s.empty()){
        tp = s.top();
        s.pop();
        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
        if(max_area < area_with_top)
            max_area = area_with_top;
    }
    return max_area;
}
```