Notebook - Maratona de Programação

Cabo HDMI, VGA, USB

Contents

1 Graph

1.1 BFS

Complexity: O(N + M), N: Vertices, M: Arestas
Uma bfs simples

```cpp
// TITLE: BFS
// COMPLEXITY: O(N + M), N: Vertices, M: Arestas
// DESCRIPTION: Uma bfs simples

vector<int> adj[MAX];
bool visited[MAX];

void bfs(int start)
{
    queue<int> q;
    q.push(start);

    while(not q.empty()){
        auto a = q.top();
        q.pop();
        if (visited[a]) continue;
        visited[a] = true;
        for (auto b: adj[a]){
            q.push(b);
        }
    }
}

void solve(){
// Alguma coisa
}
```

1.2 DFS

Complexity: O(N + M), N: Vertices, M: Arestas
Uma dfs simples

```cpp
// TITLE: DFS
// COMPLEXITY: O(N + M), N: Vertices, M: Arestas
// DESCRIPTION: Uma dfs simples

vector<int> adj[MAX];
void dfs(int a)
{
    if (visited[a]) return;
    visited[a] = true;

    for (auto b: adj[a]) {
        dfs(b);
    }
}

void solve(){
// Alguma coisa
}
```

1.3 Topological Sort

Complexity: O(N + M), N: Vertices, M: Arestas
Retorna no do grapho em ordem topologica, se a quantidade de nos retornada nao for igual a quantidade de nos e impossivel

```cpp
// TITLE: Topological Sort
// COMPLEXITY: O(N + M), N: Vertices, M: Arestas
// DESCRIPTION: Retorna no do grapho em ordem
    topologica, se a quantidade de nos retornada nao
    for igual a quantidade de nos e impossivel

typedef vector<vector<int>> Adj_List;
typedef vector<int> Indegree_List; // How many nodes
    depend on him
typedef vector<int> Order_List;    // The order in
    which the nodes appears

Order_List kahn(Adj_List adj, Indegree_List indegree)
{
    queue<int> q;
    // priority_queue<int> q; // If you want in
    lexicografic order
    for (int i = 0; i < indegree.size(); i++) {
        if (indegree[i] == 0)
            q.push(i);
    }
    vector<int> order;

    while (not q.empty()) {
        auto a = q.front();
        q.pop();

        order.push_back(a);
        for (auto b: adj[a]) {
            indegree[b]--;
            if (indegree[b] == 0)
                q.push(b);
        }
    }
    return order;
}

int32_t main()
{

    Order_List = kahn(adj, indegree);
    if (Order_List.size() != N) {
        cout << "IMPOSSIBLE" << endl;
    }
    return 0;
}
```

2 String

2.1 Z function

Complexity: Z function complexity
z function

```cpp
// TITLE: Z function
// COMPLEXITY: Z function complexity
// DESCRIPTION: z function

void z_function(string& s)
{
    return;
}
```