Report for Programming Problem 2 - ARChitecture

Team:

Student ID: 2018293871 **Name** Sofia Margarida Ribeiro da Silva
Student ID: 2018296218 **Name** Sofia Meireles Fonseca Costa

1. Algorithm description

1.1. Circuit identification

De maneira a identificar os circuitos, recorremos ao algoritmo de *Tarjan*. Escolhemos este algoritmo uma vez que este é um algoritmo bastante eficiente na procura de SCC, e ao facto de correr em tempo linear.

O objetivo deste algoritmo é formar uma subárvore na DFS do grafo em que os nós que estão fortemente conectados, ou seja, o circuito.

1.2. Selection the streets for the bike lanes

Para escolhermos os circuitos utilizamos o algoritmo de *Kruskal*. Este algoritmo permite escolher o percurso cuja distância será menor, uma vez que encontra a *minimum spanning tree* para um grafo com pesos. Recorremos também ao algoritmo Union-find, para detectar ciclos.

2. Data structures

Relativamente às estruturas de dados, utilizámos três vectores de vectores de inteiros: *grafo*, *edges* e *circuitos*. O vector *grafo*, foi usado para guardar os POIs e a distância entre eles, em forma de matriz. O vector *edges*, foi usado para guardar as distâncias entre as edges de todos os circuitos, no formato {*pontoA*, *pontoB*, *distancia*}. O vector *circuitos* guardava todos os circuitos.

O array de booleanos *staCk*, a stack de inteiros *S* e os vetores de inteiros *dfs* e *low* foram usados como auxiliares do algoritmo tarjan.

3. Correctness

De forma a alcançar os 200 pontos, começámos por transformar o pseudo-código fornecido pelo professor no slides das aulas em código funcional. Para isso, foi necessário realizar alguns ajustes de forma a ficar 100% funcional. Após a

realização desses ajustes, passámos dos 0 (com erro de execução, uma vez que tínhamos atribuído um valor demasiado pequeno ao array *staCk*) para os 200, após atribuição de maior tamanho ao array.

4. Algorithm Analysis

Relativamente à complexidade, o algoritmo Tarjan tem uma complexidade temporal linear de O(|V|+|E|), sendo V os vértices e E as arestas, uma vez que cada nó é visitado apenas uma vez.
O algoritmo Kruskal tem complexidade temporal de O(E log V).

5. References

Slides fornecidos pelo professor.