Report for Programming Problem 3 – Bike Lanes

Team:

Student ID: 2018288464 Name: João Alexandre Santos Cruz

Student ID:2018277921 Name: André Cristóvão Ferreira da Silva

1. Algorithm description

1.1. **Circuit identification**

Para a identificação de circuitos foi usado o algoritmo de *Tarjan.* Foi enviada uma lista de adjacências onde para além de ser posta o ponto de chegada é também guardada a distância entre os pontos. O algoritmo vai ser chamado para todos os pontos que não tenham sido visitados durante a chamada anterior.

O algoritmo começa por um nó qualquer, que ainda não tenha sido visitado, e vai percorrer os nós da sua lista de adjacência e altera-se o seu *low* caso o do seu pai seja menor. Caso o nó ainda não tenha sido visitado adiciona-se ao circuito. Caso o ponto já esteja na *stack* tem de se alterar o *low* para o menor entre o seu *dfs* e o *low do pai.* Após estarem todos os nós do circuito na *stack*, vai-se criar o *SCC* e vai-se adicionar a um *vector* o ponto atual, todos os pontos com o qual tem ligação e a sua distância, de maneira a poder usar-se no algoritmo seguinte.

1.2. **Selection the streets for the bike lanes**

Para a seleção dos caminhos de circuitos de bicicletas foi usaso o *Kruskal*. Este algoritmo é apenas chamado caso o número de perguntas seja maior ou igual a 3, de maneira a otimizar a rapidez.

No início cria-se um set, um grupo, para cada nó, é dado *sort* pela distância e vai-se tentando juntar os nós. Caso 2 nós tenham um *set_id* diferente então dão *merge* no mesmo grupo, o resultado final vai dar a *minimum spanning tree*. Durante o cálculo de cada caminho vai-se somando a distância total e comparando com os anteriores para obter o maior caminho de todos.

2. Data structures

As estruturas de dados utilizadas foram:

- Deque: que adiciona os pontos a que pertencem ao caminho nas chamadas recursivas do *Tarjan*.
- Vectors: dois deles bidimensionais, um que guarda a lista de adjacências com o par (vizinho,distância) num tuplo e outro que guarda os pontos que pertencem a cada caminho. Um vetor tridimensional que guarda todas as

ligações dos caminhos gerados no Tarjan em vectors {ponto1, ponto2, distância}. Por fim, 5 vetores que:

- 1 que indica o grau de um nó na árvore de procura em profundidade.
- 1 que guarda o grau mínimo de cada nós na árvore de procura em profundidade
- 1 que indica se um nó já foi adicionado à deque
- 2 que são usados para o union no algoritmo Krustal

3. Correctness

Foi selecionado o algoritmo de *Tarjan* visto que era preciso calcular as componentes fortemente conectadas para formar um circuito. Foi usado o algoritmo de *Kruskal* de maneira a poder calcular-se a *minimum spanning tree* de cada circuito obtido anteriormente, uma vez que se pretende obter apenas um caminho a ligar a todos os nós com a menor distância.

Inicialmente foi obtido *Time Limit,* uma vez que se estava a fazer bastantes cópias dos vetores, para combater isto foi diminuído o número de vetores. Após este problema estar resolvido foi obtido *Wrong Answer*, uma vez que no vector que se usa no *Kruskal* devia ser bidirecional. Com este erro corrigido foi possível obter *Accepted Answer.*

4. Algorithm Analysis

A complexidade temporal irá depender principalmente dos dois algoritmos principais. Com um grafo G = (V, A) com vértices V e arestas A:

- Tarjan: tem uma complexidade O (|A| + |V|);
- Kruskal: tem uma complexidade O (log(|A|) * |V|):

Sendo assim a complexidade temporal final dos algoritmos será O (log(|A|) * |V|).

Quanto à complexidade espacial com um grafo G = (V, A), a matriz tridimensional tem uma dimensão de tamanho fixo e, portanto, esta que consta de nº circuitos*tamanho de cada circuito*3 (ponto inicial, ponto final, distância), terá sempre menor tamanho e relevância na complexidade que a lista de adjacências que alberga O número de pontos por número de vizinhos de cada ponto. Desta forma, a lista de adjacências tem no pior caso O(n^2) de complexidade espacial e, portanto, será essa a nossa complexidade final.

5. References

Slides aula 10 e 11

https://www.thealgorists.com/Algo/GraphTheory/Tarjan/SCC