Report for Programming Problem 3 – Bike Lanes

Team:

Student ID: 2018285164 Name: Eduardo Cruz

Student ID: 2018298209 Name: Rodrigo Sobral

1. Algorithm description

1.1. Circuit identification

Ao lermos e interpretarmos o enunciado do problema percebemos que os circuitos constituíam *Strongly Connected Components* (SCCs) com dois ou mais vértices. Daí, para solucionarmos o problema da identificação de circuitos, utilizámos o conhecido algoritmo de Tarjan lecionado nas aulas, que permite identificar SCCs. Este algoritmo permite, através de uma stack, identificar e armazenar os SCCs existentes num grafo. Ao detetarmos um SCC, este era retirado da stack utilizada e armazenado numa matriz, em que a primeira coluna de cada linha contém o número de vértices do SCC que se segue.

1.2. Selection the streets for the bike lanes

Este problema consistia em, a partir dos circuitos identificados no passo anterior, identificar, para cada um deles a minimum spanning tree. Para isso recorremos ao algoritmo, greedy, de Kurskal. Para tornar-mos este algoritmo mais eficiente aplicámos as Union-Find operations lecionadas e descritas nos slides das aulas teóricas. Visto que a nossa implementação foi realizada em linguagem C, e que uma das fases do algoritmo de Kurskal passa pela ordenação das arestas, por ordem decrescente dos seus pesos, neste caso distâncias, utilizámos o algoritmo de Quick Sort da stdlib.h através da função qsort. Dado que apenas possuíamos a identificação dos vértices pertencentes ao circuito, foi necessário, para este efeito, começarmos por calcular e armazenar as arestas existentes (criámos uma struct aresta com o identificador dos dois vértices e com o peso[distância] entre eles), para serem ordenadas. O algoritmo de Kurskal era aplicado apenas quando fossem requisitadas as respostas às duas últimas questões do enunciado.

2. Data structures

Tendo em consideração questões de eficiência dos algoritmos, utilizámos uma lista de adjacência para armazenar o grafo lido do input e que foi necessária no cálculo dos circuitos existentes utilizando o algoritmo de Tarjan, dado que esta estrutura de dados permite uma leitura mais rápida dos vizinhos de um determinado vértice, comparativamente a, por exemplo, uma matriz de adjacência, visto que numa lista de adjacência é possível percorrer todos os vizinhos de um vértice com uma complexidade temporal linear, o que é bastante vantajoso na aplicação do algoritmo de Tarjan. Para identificar SCCs, no algoritmo de Tarjan, é necessária a utilização de uma stack, daí termos implementado uma stack de tamanho fixo com um counter a servir de ponteiro para o último ‘objeto’ na stack. Ainda para o algoritmo de Tarjan foi necessário

utilizar um array para armazenar o valor dos lows e a DFS tree. Como já foi mencionado, utilizámos também uma matriz para armazenar os SCCs identificados.

Dada a implementação do algoritmo de Kurskal, considerámos que seria vantajoso utilizarmos um género de matriz de adjacência mas com os pesos (distâncias) entre vértices do grafo, tendo em conta a necessidade de calcular e armazenar as arestas existentes em cada entre cada vértice do circuito. Esta matriz permite aceder em tempo constante ao valor da distância entre dois vértices, conhecendo esses dois vértices. No algoritmo de Kurskal foi necessário percorrer o array com vértices, representativo de um circuito, n*n vezes, em que n é o número de vértices desse circuito, e aceder ao valor da distância entre esses vértices para detetar ligações existentes, o facto do acesso a estas distâncias poder ser realizado em tempo contante é um fator crucial para a eficiência do algoritmo.

3. Correctness

Tivemos em consideração vários aspetos na escolha das estruturas utilizadas, para garantir a eficiência dos dois algoritmos. Por exemplo, ao nível do algoritmo de Tarjan, utilizámos uma estrutura auxiliar “on_stack”, para podermos verificar se um elemento se encontrava na stack, com uma complexidade temporal constante. O algoritmo de sorting utilizado costuma apresentar boa complexidade temporal, dependendo da ordem inicial dos ‘objetos’ antes do ordenamento. Procurámos também evitar ciclos desnecessários que adicionassem complexidade não desejada ao programa.

4. Algorithm Analysis

A complexidade temporal associada ao Algoritmo de Kurskal aplicado, para um grafo G=(V,E), em que V é a quantidade de vértices e E é a quantidade de arestas, e considerando que a complexidade média do algoritmo de ordenamento usado, Quick Sort, é de O(E*log2(E)), temos O(E*log2(V))+O(V)+O(E*V)+O(V), para a operação de sorting, make_set, find_set e union, respetivamente, o que resulta em O(E*V) para o calculo de cada minimum spanning tree. Sendo o número de circuitos existentes igual a n, temos uma complexidade de O(n*E*V). Acrescentando a esta complexidade a complexidade associada à identificação de circuitos pelo algoritmo de Tarjan, que é igual a $O(V_{Total}+E_{Total})$, complexidade linear, resulta a complexidade final de $O(n*E*V) + O(V_{Total}+E_{Total})$ que é igual a O(n*E*V). A complexidade espacial resultante destes dois algoritmos, tendo em conta a descrição feita no ponto *2.Data structures* , é de $O(V^2)$.

5. References

Slides teóricos das aulas, week10, week11 e slides week1-Intro to Mooshak and Data Structures