Instituto Superior Técnico

Relatório 2º Projecto de ASA

23 de Abril de 2016

Grupo AL009
André Mendonça (82304)
Gonçalo Ribeiro (82303)

1 - Introdução:

1.1 - O problema:

Uma empresa de transporte de mercadorias necessita de realizar um encontro entre as várias filiais da empresa numa localidade da maneira mais económica (lucrativa) possível e para isso é necessário criar um algoritmo eficaz para a resolução do problema.

É sabido que as rotas consistem numa sequência de localidades, em que cada par de localidades tem um custo e uma receita associado, para simplificar, é subtraida a receita ao custo e se o resultado for negativo significa que a empresa lucra e fica prejudicada caso contrário. A estratégia da empresa passa obviamente por escolher os resultados mais baixos destas subtrações para cada par de localidades.

Após este cálculo o objectivo é estabelecer uma localidade para realizar o encontro em que a perda total da empresa é minimizada, tendo em conta todas a filiais e as localidades que podem explorar até chegar ao ponto de encontro.

1.2 - O Input e Output:

O input consiste numa primeira linha com o número de localidades **N** (existem pelo menos duas localidades), o número de filiais ligadas à empresa **F** (existem pelo menos duas filiais) e o número de ligações **C**; Uma segunda linha com **F** números entre 1 e N que identifica as filiais ligadas à empresa; E por fim uma sequência de C linhas, em que cada linha contém três inteiros **u**,**v** e **w**, que indicam que o custo de deslocação de **u** para **v** é **w**.

Cada localidade está representada por um inteiro entre **1** e **N**.

O output gerado é constituido por uma primeira linha contendo o número que identifica o ponto de encontro e a respetiva perda total; Uma segunda contendo

os valores de perda minima de uma filial, compreendida entre **1** e **F** inclusive, até ao ponto de encontro definido pela empresa. Caso seja impossível encontrar um ponto de encontro a linha deve conter apenas o carácter "**N**" (caso em que a filial está "isolada" das restantes).

2 - Descrição da Solução:

2.1 - Linguagem de programação:

Para a implementação do projecto foi escolhida a linguagem C porque das três linguagens disponíveis para a realização do problema esta possui várias estruturas de dados já implementadas de raíz facilitando assim a criação de outras estruturas necessárias à implementação do problema, tendo em conta também que é uma linguagem que estamos mais familiarizados, o que torna mais fácil a programação para um problema complexo como este.

2.2 - Estruturas de dados:

A estrutura que melhor modela o problema proposto é um grafo, sendo assim necessária a sua implementação. Visto que as rotas dependem da direcção da visita (i.e, u para v é diferente de v para u), o grafo implementado será dirigido, e pesado, existindo um arco entre dois vértices cada vez que possa ser feito um transporte entre essas localidades. O peso de um arco, é o respetivo valor de perda.

2.4 - Algoritmo:

Decidimos usar o algoritmo Bellman-Ford para caminhos mais curtos de uma fonte única para todos os outros vértices. O caminho mais curto é, neste caso, a rota que resulta no menor prejuízo possível (i.e, no maior ganho possível).
No entanto, embora este algoritmo indique se existem ciclos de peso negativo, não nos ajuda a identificá-los. Foi necessário, para isso, extender a solução para além do Bellman-Ford: quando é identificado um vértice que faz parte do ciclo negativo, é feita uma procura para identificar quais são os outros vértices que também fazem parte do ciclo.

O programa começa por ler o input (atráves da função **scanf**) e criar o grafo **g** a partir dessa informação recolhida: por exemplo, uma linha "u v w" (**AdjL addEdge(u,v,w, nodes[])**) corresponde a criar um arco entre os dois vértices, **u** e **v**, com peso **w**. De seguida, é executado o algoritmo Bellman-Ford sobre o grafo, com uma source **firstNode**, através da função **void BellmanFord.**
É então inicializado:

- Todos os vertices com todas as distâncias a infinito, onde vão ser guardadas as distâncias entre o vértice **secondNode** e a source **firstNode**, na posição [**secondNode**];
- O primeiro vértice é inicializado com o tempo de descoberta 0.

De seguida tendo em conta o número de filiais (**n_filiais**), é executado o ciclo principal do algoritmo que corre **n_ligacoes** vezes ou até que não sejam feitas mais operações de relaxamento; este caso implica que não existem ciclos de peso negativo, e portanto o algoritmo termina.
O ciclo principal consiste numa sequência de operações de **relaxamento** a todos os arcos do grafo, segundo uma ordem arbitrária, e actualizando as distâncias no vector correspondente.
Por fim determinamos o ponto de encontro mais favorável tendo em conta as **n_localidades**, verificamos qual dos vértices contem o tempo total de descoberta (**totalDisc**) mais pequeno e atribuimos à variável **ponto_encontro** o valor da posição da localidade pretendida (**y**).

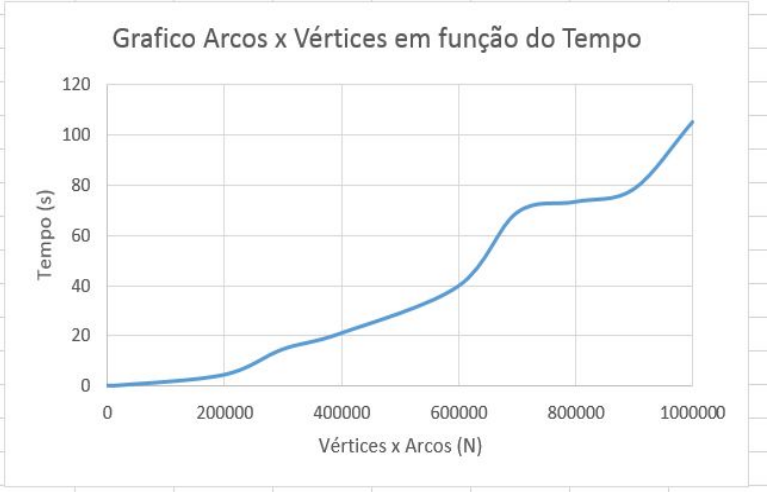
3 - Análise Teórica:

A complexidade do nosso programa é **O(V*E)**, pois utilizamos o algoritmo de Bellman-Ford, que é **O(VE)**, na sua essencia o que nos leva a concluir que a complexidade do programa tem de ser igual à do algoritmo. Se investigar-mos melhor o programa, ou seja, realçando os detalhes, podemos dar conta que a leitura do input é **O(E)** e a escrita do output é **O(V)**. A identificação do ciclo negativo (caso exista) é **O(E)**.

4 - Avaliação Experimental:

Através do gerador de grafos e avaliação dos tempos foi gerado o seguinte gráfico:

Vértices x Arcos	Tempo
20	0,003
200	0,004
2000	0,007
20000	0,113
200000	4,344
300000	14,509
400000	20,922
600000	39,663
700000	68,962
800000	73,277
900000	78,372
1000000	105,093



Os testes gerados foram feitos com valores entre (20,20), (200,200), … , (1 000 000, 1 000 000), podemos concluir que o tempo do programa aumenta considerávelmente à medida que vamos aumentando o número de vértices e arcos do input, isto deve-se à complexidade exponencial do nosso algoritmo.