Instituto Superior Técnico

Relatório 2º Projecto de ASA

21 de Março de 2015

Grupo 096
Gonçalo Fialho (ist179112)
Pedro Santos (ist178328)

1 - Introdução:

1.1 - O problema:

Uma empresa de mercadorias necessita de fazer o planeamento das rotas da maneira mais económica (lucrativa) possível e é por isso 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, 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 para cada localidade o percurso com sede na empresa que maximize o lucro da empresa.

1.2 - O Input e Output:

O input consiste numa primeira linha com o número de localidades N (existem pelo menos duas localidades) e o número de custos conhecidos C; uma segunda linha com um número entre 1 e N que identifica a sede da empresa; e 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 uma sequência de N linhas em que a linha i, compreendida entre 1 e N inclusive, corresponde à perda mínima do ponto i. Caso seja impossível definir essa perda, a linha deve conter apenas o carácter "I"; caso não existam deslocamentos suficientes para determinar essa perda, a linha deve conter apenas o carácter "U".

2 - Descrição da Solução:

2.1 - Linguagem de programação:

Para a implementação do projecto foi escolhida a linguagem C++ porque das três linguagem 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.

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, visto que cada deslocação tem um custo.

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" (**Graph::addEdge(u,v,w)**) 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 **s**, através da função **Graph::bellmanFord(s)**.
É então inicializado:

- um vector **d[V+1]** com todas as distâncias a infinito, onde vão ser guardadas as distâncias entre o vértice **v** e a source **s**, na posição [v];
- um vector **cycle[V+1]**, onde vão ser marcados os vértices que fazem parte do ciclo negativo;
- é também criada uma *flag* **done**, que vai ser usada durante a iteração principal do algoritmo para perceber se foi feito alguma operação de relax.

De seguida, é executado o ciclo principal do algoritmo, que corre **V-1** vezes ou até que não sejam feitas mais operações de relax, cujo caso é verificado com a ajuda do valor da flag **done**; este caso implica que não existem ciclos de peso negativo, e portanto o algoritmo termina e é imprimido o output atráves da função **printResult**.
O ciclo principal consiste numa sequência de operações de **relax** a todos os arcos do grafo, segundo uma ordem arbitrária, e actualizando as distâncias no vector correspondente.
Após as V-1 iterações, é feita mais uma única iteração, onde são percorridos todos os arcos do grafo para verificar se é possível fazer mais algum relax; caso seja possível, então foi encontrado um ciclo de peso negativo e é necessária a sua identificação, que é feita do seguinte modo: é marcado o vértice de destino do arco onde foi achado o ciclo negativo; depois, são marcados todos os vértices adjacentes a esse vértice, i.e, todos os que estão ligados por um arco com origem nesse vértice. Isto é feito recursivamente a todos os vértices que são adjacentes aos vértices que são marcados como parte do ciclo.
Após a identificação do ciclo de peso negativo, o programa termina com a impressão do output.

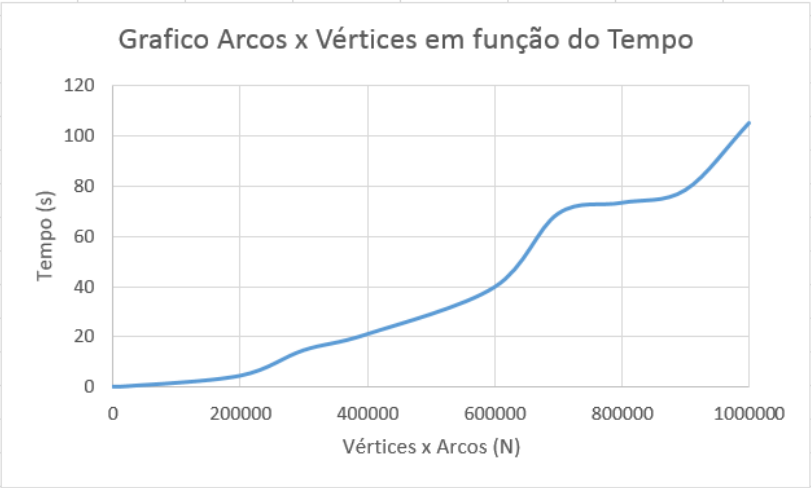
3 - Análise Teórica:

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 no seu todo 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 consideravelmente à medida que vamos aumentando o número de vértices e arcos do input, isto deve-se à complexidade exponencial do nosso algoritmo.