**Análise e Síntese de Algoritmos**

2º Projeto 2015/2016 | Grupo 022 | 78974 | 82448

# Descrição do Projeto

O problema abordado neste projeto corresponde a encontrar a localidade, num mapa com **N** localidades e **E** caminhos com um determinado valor de perda, onde uma empresa que tem **F** filiais sediadas em localidades distintas irá realizar o seu encontro. A empresa em questão faz transporte de mercadorias. Cada rota, com origem nas filiais, tem um determinado valor de perda, que resulta de subtrair a receita ao custo. Se a receita for maior que o custo, o valor de perda será negativo. A localidade escolhida será aquela com a menor soma dos valores de perda a partir das várias filiais.

Este problema tem as seguintes propriedades: o número de filiais é significativamente menor que o número de localidades; o número de pares de localidades entre os quais há um valor de perda é muito menor que o número de todos os pares possíveis; não existe um percurso entre localidades que forme um ciclo com valor de perda total negativo. Sabendo estas propriedades, o problema pode ser abordado em parte como um “All-Pair-Shortest-Path Problem”. Representa-se o problema com um grafo dirigido e pesado, em que as localidades são os vértices, os arcos são os caminhos de transporte, e os pesos de cada arco correspondem aos respectivos valores de perda.

# Implementação do Programa

O programa foi implementado na linguagem C, apenas com a utilização das bibliotecas <stdio.h>, <stdlib.h> e <limits.h>. Ao longo do programa, toma-se como infinito o valor SHRT\_MAX, uma vez que os pesos são implementados como short ints.

## Implementação da estrutura de dados

O problema acima descrito foi implementado com a utilização de grafos dirigidos pesados, representados por listas de adjacências. A implementação da lista de adjacências foi feita com base num vetor primário para representar cada um dos vértices, e uma lista simplesmente ligada para representar os arcos. São definidas as operações de inicialização do grafo, criação de arcos, e inserção dos arcos.

São também criadas estruturas de dados auxiliares à resolução do problema. Uma Min-Heap, onde cada nó tem uma chave (correspondente à identificação da localidade) e um peso (que corresponde ao valor de perda de um determinado vértice até ele). São definidas, para a Heap, operações de inicialização, inserção de um elemento, remoção do elemento mínimo, troca de pesos e, naturalmente, rearranjo da Heap.

É também criada uma FIFO com operações de inicialização, inserção e remoção de elementos (inteiros a representar localidades).

## Algoritmo de Johnson

Seja **V** o número de vértices (localidades) e **F** o número de filiais. A ideia base para a resolução do problema é a utilização do algoritmo de Johnson. Partimos do pressuposto que não existem ciclos negativos. Sabendo que o número de filiais é significativamente menor que o número de localidades e que o grafo é esparso, este algoritmo toma uma complexidade menor que, por exemplo, o algoritmo de Floyd-Warshall. O algoritmo de Johnson recorre aos algoritmos de Bellman-Ford e Dijkstra.

Começamos, então, por adicionar um vértice “dummy”, com ligação de peso 0 a todos os outros vértices. Ao corrermos o algoritmo de Bellman-Ford com base neste vértice e, depois, repesarmos cada arco do grafo com a função **w(u,v) = w(u,v)+h(u)-h(v)**, todos os valores de arcos passam agora a ser positivos. Neste momento, podemos correr o algoritmo de Dijkstra a partir de cada uma das filiais, preenchendo uma matriz de **V x F** com o valor de perda (modificado) das filiais a todas as localidades.

Após termos esta matriz, podemos voltar a repesar os arcos, invertendo a operação: **w(u,v) = w(u,v) - h(u) + h(v)**. Nestas condições, temos já ao nosso dispor o valor de perda real de cada uma das filiais a cada uma das localidades.

## Função main

A função main começa por ler do stdin o número de vértices, o número de filiais e o número de arestas, inicializando o grafo.

A identificação (localidade) das filiais corresponde aos valores que vão ser lidos na segunda linha de input. Estes valores são guardados num vetor de dimensão **F + 1**, ficando o número total de filiais na posição 0. Já a definição das arestas corresponde às restantes linhas de input que serão lidas. Cada uma destas linhas é então transformada numa aresta e adicionada ao grafo. Terminado este passo, inicializam-se as estruturas de dados auxiliares e corre-se o algoritmo de Johnson.

Após o retorno do algoritmo, temos uma matriz de **V x F** preenchida com o menor dos valores de perda de cada uma das filiais a cada uma das localidades. Tendo esses valores, procede-se à identificação da localidade com menor perda total:

* + Percorremos cada uma das colunas da matriz, onde cada coluna corresponde a uma localidade, somando todos os valores das suas linhas (filiais). Este valor corresponde ao valor de perda total de se realizar o encontro nesta localidade. Se houver algum valor infinito, quer dizer que não existe nenhum caminho da filial para essa localidade, e passa para a próxima localidade. Comparando as somas, encontra-se a soma mínima, ou seja, o valor mínimo de perda total, e a respectiva localidade é o ponto de encontro.
  + Inicializa-se a infinito o valor da soma mínima. Portanto, se no final do ciclo este valor for infinito, então não há nenhuma localidade que ligue todas as filiais. Caso não o seja, devolvemos o valor da soma, a localidade onde acontece e, de seguida, cada um dos valores de perda parcial das filiais em relação a essa localidade.

Note-se que os valores enviados para a estrutura de dados para os valores das arestas **(u,v)** são **u-1** e **v-1**. Isto deve-se ao facto de o programa esperar como input **N** vértices de valores de **1** a **N**, e a estrutura de dados interna funcionar com **N** vértices de valores **0** a **N-1**. Isto permite-nos a fácil identificação dos vértices através do índice do vetor primário.

# Análise Teórica

Foi escolhida como estrutura de dados para a implementação deste problema um grafo representado como lista de adjacências. Tomando **V** como o número de vértices e **E** como o número de arcos, numa estrutura deste tipo, a complexidade de inicializar um grafo vazio é de **O(|V|)**, sendo que a posterior inserção de todas as arestas tem um custo de **O(|E|)**. Assim sendo, a complexidade total da construção de um grafo dado como input é de **O(|V|+|E|)**.

Os passos principais no algoritmo são a chamada a Bellman-Ford, que tem uma complexidade estudada de **O(|V||E|)**, e F chamadas ao algoritmo de Dijkstra, que tem uma complexidade, devido à implementação com uma Min-Heap, de **O(|V|log|V| + |E|)**.

Sendo que o número de vértices corresponde ao número de localidades e **F** é o número de filiais, a posterior inicialização da matriz que armazena os valores de perda parciais é de **O(|F||V|)**.

Assim, a complexidade total do algoritmo é de **O(VE + F(Vlog(V) + E))**, ou seja, **O(VE + FVlog(V))**.

# Avaliação Experimental dos Resultados

Para proceder à avaliação da eficiência do programa foram criadas três estruturas diferentes de casos de teste de input com tamanhos definidos.

Na construção do Gráfico 1, foi-se incrementando o número de vértices dado como input ao programa. O primeiro input foi gerado com um valor inicial de 1000 vértices, e os inputs subsequentes incrementaram na ordem dos milhares até se gerar um input final de 10000 vértices (**1\*V** até **10\*V** ), sempre com um valor constante de 200 arestas e 100 filiais. O Gráfico 1 relaciona os casos de teste descritos com o tempo de execução do programa, medido no terminal com o programa *time*. Como se pode constatar pela observação do gráfico, a linha de tendência é polinomial, sendo uma aproximação da complexidade **O(VE + FVlog(V))**do algoritmo criado, que com **E** e **F** constantes fica **O(V + Vlog(V)).**

Para a construção do Gráfico 2, fixou-se o número de vértices (1000) e filiais (100) e incrementou-se apenas o número de arestas, na ordem das centenas, obtendo grafos progressivamente mais densos. O número de arestas, à semelhança do primeiro caso, variou de **1\*E** até **10\*E**, contabilizando este último um total de 1000 arestas. A linha de tendência neste caso é linear, correspondendo à complexidade linear do algoritmo criado quando existe apenas variação do número de arestas: **O(VE + FVlog(V))** → **O(E)**.

Finalmente, o Gráfico 3 representa a variação do número de filiais, mantendo-se o número de vértices e arestas fixo, 1000 e 100 respectivamente. Neste caso, incrementou-se as filiais na ordem das dezenas, obtendo-se inputs de 10 até 100 filiais. Á semelhança do número de arestas, a linha de tendência corresponde a uma complexidade linear, devido ao valor fixo de vértices.

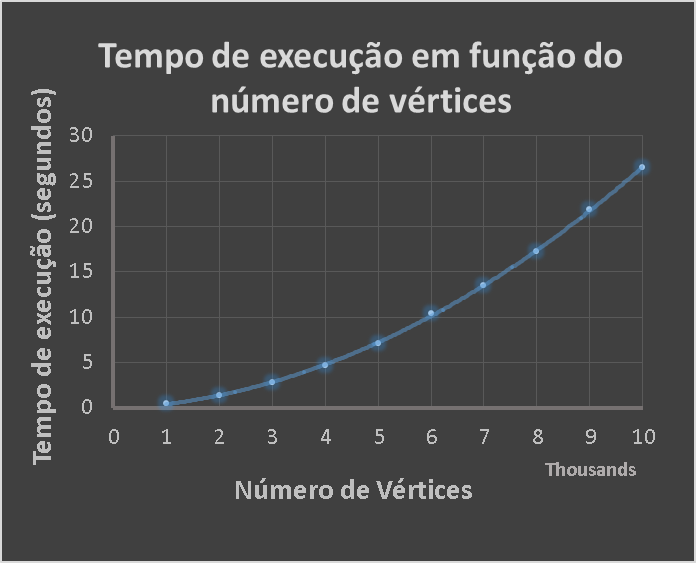


Gráfico 1 - Estrutura de teste 1

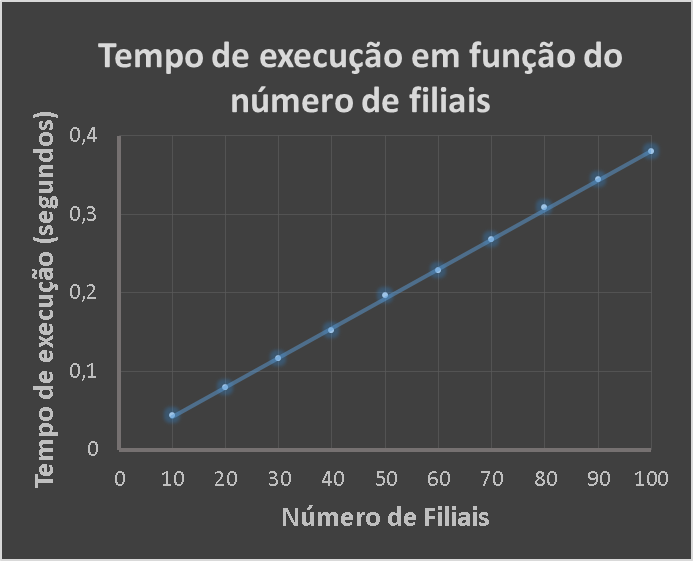
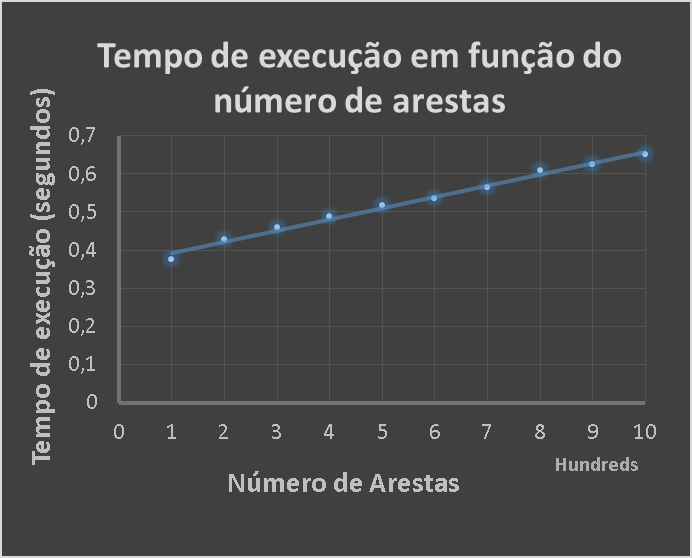


Gráfico 2 - Estrutura de teste 2

Gráfico 3 - Estrutura de teste 3

# Referências bibliográficas

SEDGEWICK, Robert; **Algorithms in C;** 1997; Addison-Wesley Publishing Company