# Relatório do 2º Projeto de ASA

## Grupo al065

**Filipe Azevedo – nº82468**

**Martim Zanatti – nº82517**

## Introdução

O objetivo deste projeto consiste em encontrar um algoritmo eficiente para encontrar um ponto de encontro adequado para celebrar o sucesso de uma empresa de transporte de mercadorias que possui várias filiais em localidades diferentes e que o transporte de mercadorias de uma localidade para outra tem um custo (ou receita se o custo for negativo). De forma a ser mais eficiente a empresa decidiu arranjar uma maneira de maximizar as receitas fazendo com que cada filial entregue mercadorias quando estiver a ir para o tal ponto de encontro.

Podemos representar o problema como um grafo dirigido e pesado, em que cada arco representa o caminho de uma localidade para outra, cada vértice representa uma localidade, e o peso de cada arco representa o custo, positivo ou negativo, de transportar mercadorias entre localidades.

Para realizar este algoritmo é necessário guardar os dados fornecidos à entrada numa estrutura. Estes dados contêm informações sobre o número de localidades onde a empresa opera, o número de filiais da empresa, o número de ligações entre localidades, as localidades onde estão sediadas as filiais, e as ligações entre localidades e o seu respetivo custo (ou receita se o custo for negativo).

De seguida apresentaremos a nossa solução para o problema assim como a sua complexidade, e também as dificuldades que tivemos ao implementar a nossa solução. O código do nosso grupo foi desenvolvido na linguagem de programação C.

## Descrição da Solução

Para este projeto foram criadas três estruturas: a estrutura vértice, que representa um vértice e contém os atributos fundamentais para a resolução do algoritmo, uma estrutura chamada struct LISTAnode que guarda um vértice e um ponteiro para outro elemento dessa estrutura, e a estrutura do tipo lista de

adjacências, que guarda a informação da rede e permite uma gestão eficiente da mesma.

No início, o programa lê do standard input três inteiros que representam o número de localidades, o número de filiais da empresa e o número de ligações entre localidades, respetivamente. É criada uma lista de struct LISTAnode alocada dinamicamente com a dimensão do número de localidades. De seguida é lida uma linha que contém informação sobre quais localidades onde estão sediadas as filiais da empresa. E por fim várias linhas que contêm informação sobre as ligações entre localidades e os respetivos custos para a empresa. À medida que lê do standard input as ligações entre localidades e o seu custo, vai guardando esta informação na lista de adjacências.

O algoritmo utilizado para a resolução eficiente deste problema foi o algoritmo de Bellman-Ford. Este algoritmo é mais lento que o algoritmo de Djikstra mas mais versátil, uma vez que permite a resolução do problema para grafos com pesos negativos, o que acontece neste projeto.

Dado um vértice inicial, o algoritmo coloca a distância de todos os vértices do grafo em relação a este a infinito e a sua própria distância a zero. De seguida vai percorrendo todos os arcos do grafo V-1 vezes (em que V é o número de vértices do grafo, e neste caso o número de localidades) relaxando-os. Isto quer dizer que para cada arco do grafo o algoritmo atualiza a distância do vértice de destino em relação ao vértice inicial se a distância do vértice inicial do arco mais o peso do mesmo for menor que a distância atual do vértice de destino em relação ao vértice inicial.

No fim do algoritmo este retorna a distância mais curta (com menor custo) do vértice inicial a todos os outros vértices do grafo.

No caso do nosso problema vamos aplicar o algoritmo de Bellman-Ford para cada filial assim descobrindo as distâncias dessa filial a cada localidade. No fim somamos todas as distâncias de todas as localidades a todas as filiais descobertas pelo algoritmo e a localidade que tiver a soma de distâncias menor é o ponto de encontro. Se as distâncias de todas as localidades forem infinito então é porque é impossível determinar um ponto de encontro.

Quando for descoberto o ponto de encontro aplicamos o algoritmo ao grafo transposto com o vértice correspondente ao ponto de encontro como vértice inicial descobrindo assim a distância de cada filial a essa localidade.

O output será uma linha com a identificação da localidade que servirá de ponto de encontro e o custo total do transporte de todas as filiais para essa localidade, e outra linha que corresponde ao custo do transporte de cada filial para lá. Se não for possível calcular um ponto de encontro o output deverá ser apenas uma linha com a letra “N”.

# Análise Teórica

A complexidade estará intrinsecamente ligada á complexidade da função *main* do programa. Relembrar que: *V* é o número de vértices, *E* o número de arcos entre estes, e F o número de filiais da empresa.

A função que cria a lista de adjacências tem complexidade O(1). Depois recorremos a um ciclo *while* para guardar as localidades das filiais e também a ordem como elas são dadas no input. Logo, esta parte do programa tem complexidade O(F). Por fim fazemos um ciclo *for* para guardar os arcos com complexidade O(E).

Depois de guardar os dados do input nas nossas estruturas, utilizamos o algoritmo Bellman-Ford para calcular o ponto de encontro. Aplicamos o algoritmo a todas as filiais, ou seja, aplicamos o algoritmo para cada filial com cada filial como vértice inicial e vamos somando a distância calculada em cada iteração às distâncias calculadas anteriormente com complexidade O(VF). Assim, aplicamos Bellman-Ford F vezes, e sabendo que o algoritmo tem complexidade O(VE) então todo este processo tem complexidade O(FVE + VF). Então podemos simplificar a expressão ficando O(FVE + VF) = O(VF(E + 1)) = O(FVE).

A seguir, quando já temos a soma das distâncias para de cada filial a cada localidade podemos calcular o ponto de encontro com menor distância para as filiais, ou seja, menor custo para a empresa. Isto é feito através de um ciclo *for* que percorre a lista de adjacências e vai verificando a distância total de cada vértice. Isto corre em O(V).

De seguida, quando já tivermos obtido o ponto de encontro fazemos o algoritmo Bellman-Ford ao grafo transposto com o ponto de encontro como vértice inicial, calculando assim as distâncias de cada filial ao ponto de encontro em O(VE) para dar o output.

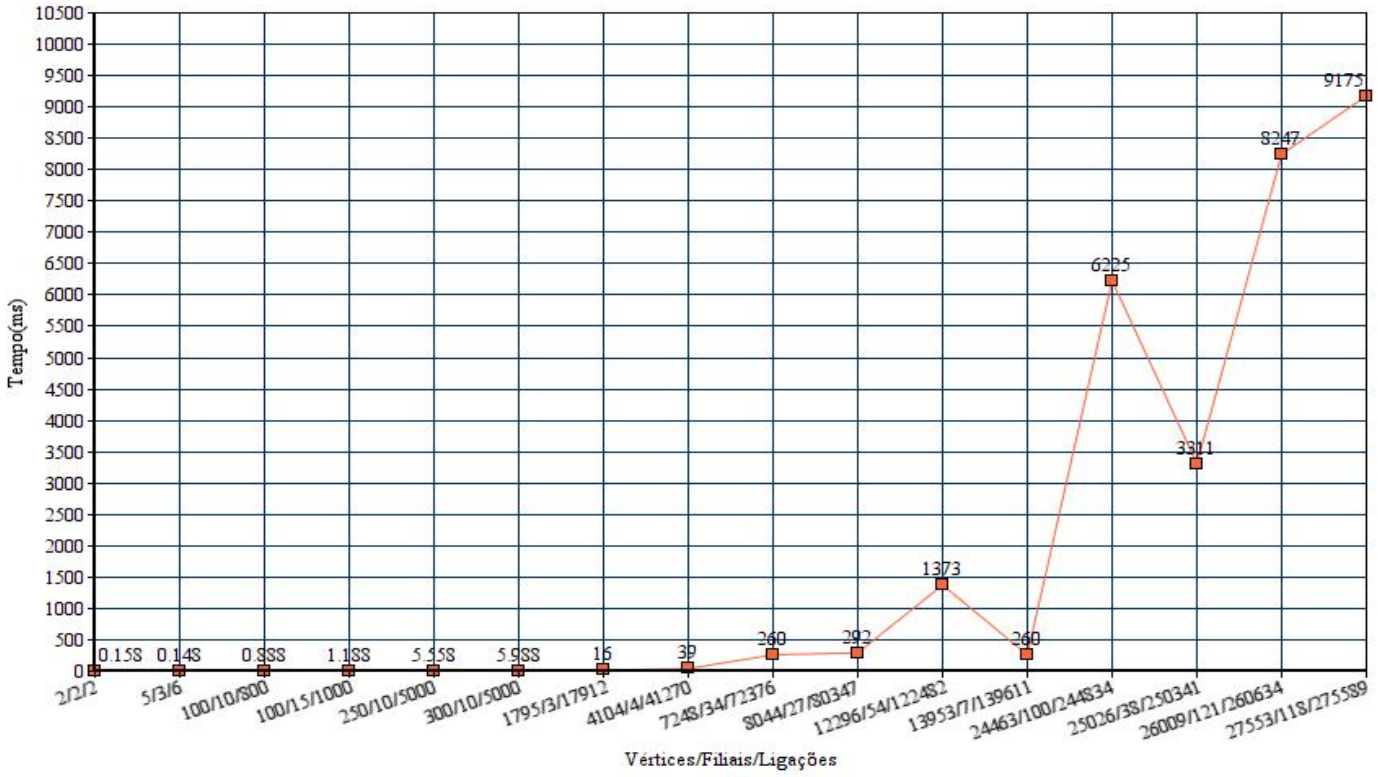
Por último basta fazer *free* das nossas estruturas, com complexidade O(E).

Como podemos observar, a nossa função *main*, e portanto o nosso programa, tem complexidade O(1) + O(F) + O(E) + O(FVE) + O(V) + O(VE) + O(E) = O(FVE).

# Análise Experimental

Para esta avaliação, foram utilizados os testes disponibilizados pelos responsáveis da disciplina e também alguns testes criados por nós para testar o desempenho do programa em diferentes situações.

Cada teste foi realizado 5 vezes, sendo que os valores presentes no gráfico abaixo representam as médias desses testes. Os tempos apresentados correspondem à componente *real* dos tempos obtidos. O gráfico está ordenado pelo número de vértices e pelo número de arcos do grafo testado.




Assim, verifica-se que o tempo de execução do programa aumenta quanto mais vértices e ligações entre eles houver mas dependendo também do número de filiais existentes pois este número vai determinar quantas vezes se aplica o algoritmo Bellman-Ford.

Era exatamente este tipo de comportamento que era esperado de um programa com complexidade O(FVE).

## Referências


https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://en.wikipedia.org/wiki/Johnson%27s_algorithm