# Introdução

O principal objetivo deste projeto visa a concretização de um algoritmo que determine o menor custo da construção de uma rede de aeroportos e estradas de forma a ligar um determinado conjunto de cidades, bem como o número total de aeroportos e estradas a construir. O algoritmo deteta também situações em que não é possível a construção de uma rede que ligue todas as cidades. No caso de existirem várias configurações possíveis de menor custo, o algoritmo dá prioridade à que tiver menos aeroportos.

A solução abaixo descrita foi desenvolvida em C e demorou um total de 30 horas a finalizar.

# Solução

Foi utilizado um **grafo bidirecional pesado** para guardar a informação introduzida pelo utilizador. Cada cidade representa um vértice do grafo e cada estrada, uma aresta. O custo da aresta é o custo da estrada, introduzido pelo utilizador.

Relativamente aos aeroportos, foi criado um vértice auxiliar (um *Hub*[[1]](#footnote-2)). Para que haja uma ligação aérea entre duas cidades, ambas têm de ter uma aresta até ao vértice *Hub* de custo igual ao custo dos respetivos aeroportos. Para que o algoritmo possa dar prioridade às arestas do tipo estrada, foi acrescentado o valor de 0.1 ao custo das arestas relativas à construção de aeroportos.[[2]](#footnote-3)

Se este grafo não for ligado (à exceção do vértice *Hub*), é impossível construir uma rede que ligue todas as cidades.

A nossa solução consistiu na **adaptação do algoritmo Prim** ao problema apresentado.

Esta adaptação passou pela implementação do próprio algoritmo (baseado no pseudocódigo apresentado nas aulas), na implementação de um amontoado de custo mínimo (*min* *heap*) para resolução das prioridades (necessário ao funcionamento eficiente do Prim) e ainda de algumas funções auxiliares para tornar o código mais legível.

O programa começa por esperar pelo input do utilizador. À medida que o recebe, inicializa e constrói o grafo contendo a totalidade das arestas. As arestas e respetivos custos são guardados num *array* de listas de adjacências, onde a posição i do *array* contém a lista dos vértices que estão ligados ao vértice i por uma aresta. O vértice *Hub* encontra-se no final do *array* (de tamanho nº vértices total mais 1, para que o índice coincida com o número do vértice).

Foi utilizada uma *flag* para assinalar a presença ou ausência de aeroportos. Se o grafo inicial não contém aeroportos, esta inicializa com o valor *false*, caso contrário é inicializada a *true*.

Após construção do grafo (e inicialização da *flag*), é chamada a **função auxiliar *runPrim*** cujo objetivo é correr o algoritmo Prim sobre o grafo, através da chamada da função *prim*. Esta função devolve um *array* contendo o custo total da solução encontrada, o número total de aeroportos e o número total de estradas na solução.

Caso o grafo inicial não seja ligado (i.e., não tenha arestas suficientes para ligar todos os vértices), a função *prim* retorna *null* e *runPrim* escreve no *stdout* “Insuficiente”, retornando.

Caso contrário, se a MST (*minimum spanning tree*) encontrada não possuir aeroportos, *prim* retorna o referido *array*, cujos valores são impressos ao utilizador pela *runPrim*.

No caso da MST encontrada incluir aeroportos, e caso o número de arestas do tipo estrada seja suficiente (igual ou maior que o número de cidades - 1), então é de novo chamada a função *prim*, desta vez com o valor da *flag* a *false* de forma a que a função ignore todas as arestas relativas a aeroportos, assim como o vértice *Hub.* Caso seja possível criar esta MST sem aeroportos, se esta tiver igual custo à primeira MST encontrada, os valores correspondentes a esta nova solução serão os impressos ao utilizador.

Após terminar a função *runPrim*, a memória alocada para o grafo é libertada e o programa termina.

A **função *prim*** começa por criar dois *arrays*: (1) um referente à fila de prioridades, que tem em conta o menor custo da criação de uma aresta entre um vértice e o seu vértice pai, guardando um ponteiro para a informação relativa ao vértice que se encontra em cada posição de prioridade; (2) e outro contendo toda a informação relevante acerca de cada vértice, nomeadamente o custo da aresta entre ele e o seu vértice pai, qual o seu vértice pai, se já foi (ou não) visitado pelo algoritmo e qual a sua posição relativa na fila de prioridades.

Após criação dos *arrays* acima, a função inicia o algoritmo propriamente dito: Este começa por extrair o vértice com o menor custo associado, verificando se o custo de o ligar aos vértices adjacentes é menor que o custo atualmente associado a esses. Caso isso se verifique, o custo da ligação desses vértices à solução em construção é atualizado assim como o seu vértice pai. De seguida, a fila de prioridades é reordenada, “puxando” para a primeira posição o vértice com o menor custo associado. Este processo é repetido até que todos os vértices tenham sido visitados. Caso o vértice de custo mínimo extraído não tenha um pai associado (à exceção do vértice inicial), estamos perante um grafo não ligado, e a função *prim* termina de forma precoce, retornando vazio.

O cálculo do custo total, assim como a contagem do número total de aeroportos e de estradas numa solução, é feito pela **função *maxCost****,* chamada no final da função *prim* (antes desta libertar a memória alocada e retornar o valor devolvido pela função referida). Nesta função é percorrido o *array* com a informação dos vértices e, mediante a aresta associada ao custo seja um aeroporto ou uma estrada, esta atualiza os respetivos valores, assim como o custo total associado à solução. Finalmente retorna um *array* com estes três valores.

# Análise Teórica

Para determinar a complexidade do algoritmo, analisámos cada um dos seus procedimentos em separado:

* **Inicialização do grafo** dado um número V de vértices: Ѳ(V) (o vetor é percorrido depois de alocado para que todas as posições sejam inicializadas a NULL).
* **Adição de E arestas ao grafo inicializado**: Ѳ(E) (as arestas são adicionadas uma de cada vez à lista ligada do vértice origem e à do vértice destino).
* **Execução do Prim** no grafo construído: O(E log V) (graças ao uso de uma *min-heap)*.
  + **Cálculo do custo total**: Ѳ(V) (é percorrido um vetor com toda a informação acerca de cada vértice extraída da execução do Prim, com tamanho igual a V).
  + **Libertação de memória**: Ѳ(V + E) (necessário percorrer todo o grafo).

Em suma, a complexidade total do programa é O(E log V).

# Avaliação Experimental

Fez-se uma bateria de testes para **grafos ligados** (grafos com solução). Para cada número de vértices foram feitos 5 testes seguidos e medida a variação de tempo de relógio entre o início e o fim dos 5 testes. O número de vértices é crescente, a começar em 100 com incrementos de 100 vértices. O número de arestas foi variado de duas formas: **(1)** utilizando o número máximo de arestas obtido através do gerador de testes (o total de vértices utilizado é de 5500 dado o tamanho dos inputs gerados) e **(2)** utilizando o número mínimo de arestas (E = V – 1) (total de vértices utilizado é de 200000). Os gráficos abaixo retratam as duas situações acima descritas:

Gráfico 1 - Variação de tempo com número crescente de vértices (V) e o maior número possível de arestas (E).

Gráfico - Variação de tempo por número crescente de vértices (V) e o menor número possível de arestas (E) num grafo ligado.

É possível concluir, através da análise dos gráficos, que a complexidade experimental é idêntica à complexidade teórica esperada, sendo assim possível afirmar que a solução desenvolvida é eficiente na resolução do problema proposto.

1. <https://en.wikipedia.org/wiki/Airline_hub> [↑](#footnote-ref-2)
2. Como o trabalho com *floats* em C nem sempre corresponde ao esperado, os custos de estradas e aeroportos inseridos pelo utilizador foram multiplicados por 10 e de seguida adicionou-se 1 às arestas do tipo aeroporto, dividindo o custo total final por 10 antes deste valor ser impresso ao utilizador. [↑](#footnote-ref-3)