Análise e Síntese de Algoritmos

Grupo 32 : André Fonseca (84698), Leonor Loureiro (84736)

Este projeto foi desenvolvido no âmbito da cadeira de Análise e Síntese de Algoritmos do curso de Engenharia Informática e Computadores (Taguspark).

O projeto tinha por base a integração de um conjunto de cidades numa rede de ligações aéreas e rodoviárias. O programa a implementar deveria decidir quais as estradas e aeroportos a construir de forma a minimizar o custo total das obras.

A implementação da nossa solução foi concretizada em C++.

O programa recebe como **input:**

- uma linha com o número *N* de cidades;
- uma linha com o número *A* de aeroportos;
- uma sequência de *A* linhas, em que cada linha consiste em dois inteiros *a* e *c*, onde *c* denota o custo de construir um aeroporto na cidade *a*;
- uma linha com o número *E* de potenciais estradas a construir;
- uma sequência de linhas *E*, em que cada linha consiste em três inteiros *a*, *b* e *c*, onde *c* denota o custo de construir uma estrada entre as cidade *a* e *b*.

Como **output**, o programa devolve:

- Caso não seja possível construir a rede, o output apresentado é “Insuficiente”;
- Caso seja possível construir a rede, o output corresponde a uma linha com o custo total e uma linha com dois inteiros indicando o número de aeroportos e o número de estradas a construir.

Descrição da Solução:

Modelamos o problema através dois grafos pesados não dirigidos. Para os ligações do primeiro grafo contribuem apenas as estradas:

- os vértices correspondem às *N* cidades;
- existe um arco com peso *c* do vértice *a* para o vértice *b* se é possível construir uma estrada que ligue a cidade a à cidade *b* com custo *c*;

Para o segundo grafo, para além das estradas, temos também em conta os potenciais aeroportos:

- existe um vértices adicional que corresponde ao espaço aéreo;
- existe um arco de peso *c* de cada vértice *a* para o vértice do espaço aéreo se é possível construir um aeroporto na cidade *a* com custo *c*;

A estrutura implementada que representa uma arco contém:

- dois inteiros que identificam os vértices que são ligados pelo arco;
- um inteiro corresponde ao peso do arco;
- e um inteiro que identifica o tipo da ligação: uma estrada ou um aeroporto.

A estrutura implementada que representa um grafo contém:

- um inteiro com o número de vértices do grafo;
- um vetor com todos os arcos do grafo.

Uma vez o objetivo era decidir o número de estradas e aeroportos a construir, minimizando o custo total das obras e priorizando a solução que tem menos aeroportos, aplicámos a ambos os grafos o algoritmo *Kruskal* que permite determinar a árvore abrangente de menor custo, mantendo o registo de ligações de estradas e de aeroportos que compõe a árvore.

<u>O algoritmo de Kruskal segue os seguintes passos:</u>

1. Ordena todos os arcos por ordem não decrescente segundo os seus pesos. No caso do grafo com estradas e aeroportos, as ligações das estradas são consideradas como tendo menor custo. A ordenação é feita pelo *sort* do C++ (**Ref:** http://www.cplusplus.com/reference/algorithm/sort/).

2. Seleciona o arco de menor peso e verifica se este forma um ciclo com a árvore abrangente formada até este ponto. Caso não forme um ciclo, então é adicionado à árvore abrangente e é incrementado o contador de estradas ou o contador de aeroportos construídos de acordo com o tipo da ligação, ou seja, se esta representa uma estrada ou um aeroporto. Caso contrário, o arco é descartado.

3. Repete o segundo passo até que todos os arcos do grafo tenham sido considerados.

Antes de aplicar o algoritmo *Kruskal* a qualquer um dos grafos, verificamos se para alguma cidade, o número de estradas que a esta estão ligadas é zero e não é possível construir um aeroporto na cidade. Caso se verifique, então não é possível construir a rede, pelo que o output do programa é “Insuficiente”.

Após aplicarmos o algoritmo *Kruskal* a ambos os grafos, verificamos se para nenhum dos grafos foi possível determinar a árvore abrangente de menor custo, significando que não é possível construir a rede, pelo que o output do programa é “Insuficiente”.

Caso contrário, comparamos o custo da àrvore abrangente determinada para os dois grafos. Se o custo for igual para os dois, então no output, o número de estradas construídas corresponde ao valor calculado pelo Kruskal para o grafo que contém somente estradas, e o número de aeroportos construídos é zero. O custo da rede é o custo da árvore.

Se os custo da árvore abrangente determinado para ambos os grafos diferir então o output corresponde ao número de estradas e de aeroportos construídos calculados pelo *Kruskal* para o grafo que cujo custo da árvore abrangente é inferior.

Análise Teórica:

Consideremos a aplicação do programa para N cidades, com E potenciais estradas e A aeroportos possíveis.

A leitura do input é O(E+A). A inicialização dos *arrays* com o número de estradas e o custo do aeroporto de cada cidade é O(V).

O tempo de inserção de cada arco no vetor do grafo é O(1). Para o grafo que contém somente as ligações formadas por estradas, a inicialização do grafo é O(E). Para o grafo que tem em conta também os aeroportos, a inicialização do grafo é O(E+A). Logo, a inicialização do grafo é O(E+A).

O ciclo que verifica se existe alguma cidade à não está ligada nenhuma potencial estrada nem é possível construir nela um aeroporto é O(N).

A função sort usada na ordenação dos arcos, de uma biblioteca do c++, não têm uma implementação específica, pelo que esta pode variar. No entanto a sua complexidade é sempre O(C log C) para a comparação de C elementos (**Ref:** https://en.wikipedia.org/wiki/Sort_%28C%2B%2B%29). Assim a ordenação dos arcos é O(E log E) para o grafo que contém somente as ligações formadas por estradas, e O((E+A) log (E+A)) para o grafo que tem em conta também os aeroportos. Logo, a ordenação dos arcos é O((E+A) log (E+A)).

A operação Make-Set é O(N) para ambos os grafos.

O ciclo principal do algoritmo Kruskal, percorre todos os arcos e aplica as operações Find-Set e Union. Estas operaçãoes são O(log V) para ambos os grafos. Assim, o ciclo é O(E log V) o grafo que contém somente as ligações das estrada e O((E+A) log V) para o grafo que tem em consideração também os aeroportos. Logo, o ciclo é O((E+A) log V).

Concluímos assim, que o tempo de execução da solução é O(E+A) ou O(N) se existir uma cidade que não está ligada a nenhuma potencial estrada e na qual não é possível construir um aeroporto. Para os restantes casos, o tempo de execução da solução é O((E+A) log (E+A)) ou O((E+A) log V).

$E < V^2 \rightarrow O(E) = O(V^2)$. Assim temos $O(\log E) = O(\log V^2) = O(2\log V) = O(\log V)$. Podemos então garantir O(E log V). Do mesmo modo, V = O(E), pelo que **podemos reduzir a complexidade a O (E log E)**.
(**Ref:** http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Tremblay/L22-MinSpanB.htm)

Análise Experimental

Usando o programa disponibilizado pela página da cadeira, gerámos testes com E = Aeroportos+Estradas diferenciando 2 situações: sem erros e insuficiência.
A análise experimental nestas duas situações permite concluir que:

- Na maioria, os casos de insuficiência são mais rápidos.
- A relação entre o input E com o tempo de execução resulta numa função E*log(E) multiplicado por uma constante.

Sem Erros:

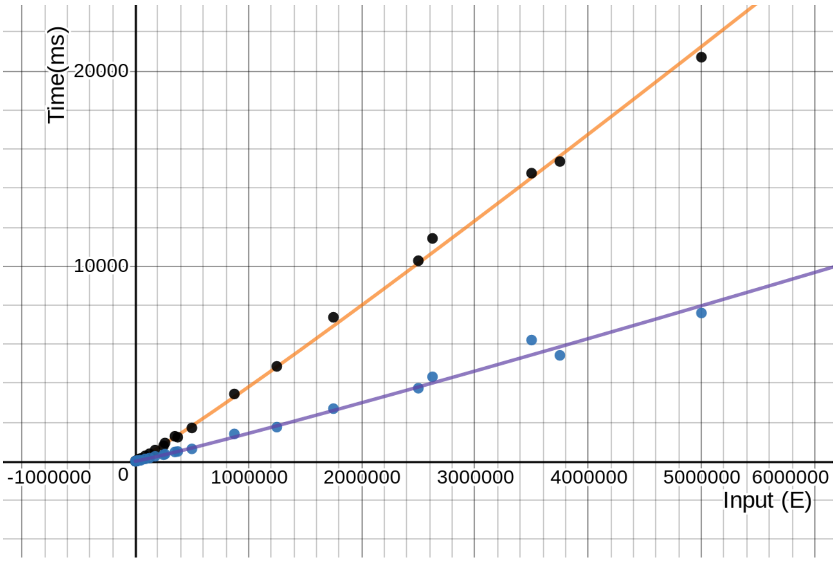
Input(E) k=1000	1750	25k	125k	350k	875k	2500k	5000k
Tempo(ms)	4	72	384	1280	3448	10280	20716

Insuficiencia:

Input(E) k=1000	1750	25k	125k	350k	875k	2500k	5000k
Tempo(ms)	0	32	160	476	1400	3744	7600

Gráfico com a comparação dos 2 casos gerados com regressão da função m*E*log(E) :

- A **laranja** a função dos resultados sem erro (R^2 =0.9986 , m= 0.00019)
- A **roxo** a função dos resultados insuficientes (R^2 =0.9915 , m= 0.00007)



Outras referências:

- https://fenix.tecnico.ulisboa.pt/downloadFile/1970943312285705/aula10.pdf
- https://fenix.tecnico.ulisboa.pt/downloadFile/1970943312285706/aula11.pdf