RELATÓRIO DO PROJETO 2

ANÁLISE E SÍNTESE DE ALGORITMOS • 2º SEMESTRE, 2016/2017
GRUPO 29: MIGUEL REGOUGA, 83530 • PEDRO CALDEIRA, 83539

INTRODUÇÃO

No âmbito do segundo projeto da cadeira de Análise e Síntese de Algoritmos, foi-nos apresentado um problema que consiste em realizar um investimento em infraestruturas que ligam cidades de um país, sendo que o custo deste investimento deve ser o menor possível. As infraestruturas a construir podem ser estradas ou aeroportos, tendo em conta que, preferencialmente, são construídas estradas.

Para modelar este problema recorremos à teoria de grafos. É criado um grafo não dirigido e pesado, em que um vértice corresponde a uma cidade do Bananadistão, cada arco entre dois vértices corresponde a uma possível estrada a construir entre duas cidades e o peso dos arcos corresponde ao custo de construção dessa estrada.

Implementámo-la na linguagem de programação C, sendo que recorremos ao algoritmo de determinação de árvores de menor custo, *Kruskal*, e também ao algoritmo de ordenação eficiente *Heapsort*, os quais tivemos de estudar *a priori* através dos slides disponibilizados pelo corpo docente.

DESCRIÇÃO DA SOLUÇÃO

Para a resolução do problema, o nosso programa irá inicialmente analisar a primeira linha de input, que corresponde ao número total de cidades. Posteriormente, é lido do input o número de potenciais aeroportos a construir, bem como o preço de cada um e em que cidades os podemos construir.

Na nossa solução, resolvemos criar um vértice adicional que se liga a todas as cidades que podem ter aeroportos. É de referir que este

vértice só é de facto criado caso o número de aeroportos seja maior do que zero.

De seguida, é criado um vetor auxiliar (***vetorAeroportos***) onde ficam guardados todas arestas que estiverem munidas de vértices (cidades) com capacidade de ter aeroportos.

Por último, é lido o número total de potenciais estradas a construir. Tendo os números totais de cidades e de possíveis estradas e aeroportos a construir, estamos em condições de criar o grafo que servirá de base para a execução do resto do programa. Este grafo, no fundo, é composto pelo número de arcos e de vértices, bem como um vetor constituído por todas as ligações entre as cidades.

As últimas linhas do input correspondem ao número de potenciais estradas a construir, bem como ao custo de cada uma e que cidades ligarão. No nosso projeto, é criado, para cada um dos casos, um novo arco dentro do grafo entre as duas cidades referidas no input, ou seja, é adicionado um novo arco ao vetor de arcos.

Como passo final para a inicialização da estrutura base, e caso o número de aeroportos seja maior que zero, as ligações entre o vértice adicional criado anteriormente e os vértices que poderão vir a ter aeroportos são acrescentadas.

Após a criação das estruturas básicas para funcionamento do nosso programa, a execução foca-se então na realização do algoritmo de *Kruskal*. Escolhemos este algoritmo pois queremos obter uma árvore de menor custo (*MST*) entre os vértices do grafo.

No contexto do nosso problema, queremos averiguar quais as estradas/aeroportos a serem construídos, e entre que cidades, de forma a que todas as cidades estejam ligadas com o menor investimento possível. A sua implementação relativamente simples em comparação com outros algoritmos que calculam *MSTs*, bem como a sua baixa complexidade assimptótica, O(E log V), foram também aspetos que tivemos em conta na escolha do algoritmo.

O algoritmo de *Kruskal* requer que as arestas estejam ordenadas pelo seu peso de forma crescente. Assim, antes de entrar na execução do algoritmo em si, é ordenado o nosso vetor de arcos do grafo, recorrendo ao algoritmo *Heapsort*, que tem complexidade O(n log n), onde n é o número de elementos a ordenar.

Por fim, o programa irá executar o algoritmo de *Kruskal* duas vezes: uma primeira em que o vértice adicional não existe, ou seja, é feita uma análise sem considerar os potenciais aeroportos a construir e uma segunda onde o vértice adicional existe e temos em conta os referidos aeroportos.

No final de cada execução do algoritmo de *Kruskal*, é devolvida uma estrutura (***kruskalResult***) que contém o custo final e mínimo da árvore de menor custo, o total de aeroportos a construir, o total de estradas a construir e uma *flag* que indica se é um caso de insuficiência ou não.

Para analisar o caso de insuficiência, recorremos a uma variável auxiliar (***minEdges***) que corresponde à diferença entre o número total de vértices e 2, caso estejamos na iteração que considera os aeroportos, e à diferença entre o número total de vértices e 1 caso contrário. Na nossa conceptualização, a variável corresponde ao número de arcos que tem de haver para que existam ligações entre todas as cidades do grafo.

Por último, esta variável é posteriormente analisada através da seguinte fórmula:

Número de arestas do grafo (corresponde à soma do número de estradas e aeroportos construídos) **!= minEdges + 1**.

Caso esta desigualdade seja verificada, o campo ***flagInsuficiente*** fica a 1 e é passado ao resultado.

As duas estruturas serão comparadas no final da execução do programa, e é neste momento que sabemos qual a MST mais economicamente viável e que corresponda ao pedido pelo projeto.

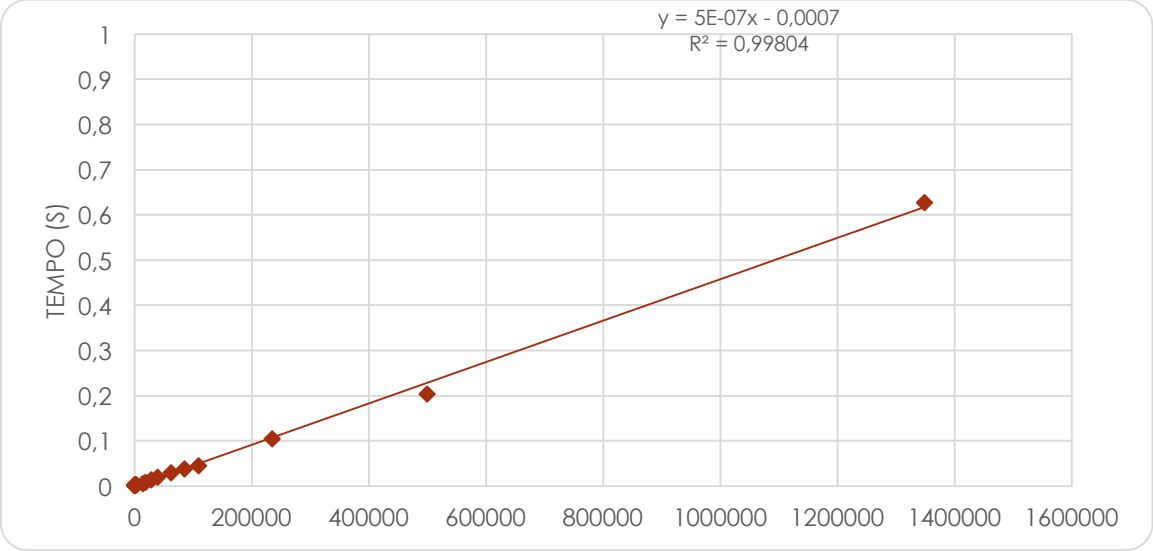
ANÁLISE TEÓRICA

O nosso programa, para a ordenação dos arcos, utiliza do *Heapsort*, que tem complexidade O(E log E), onde E corresponde ao número de arestas.

Em relação ao algoritmo de *Kruskal*, o mesmo tem complexidade O(V) para as operações de *Make-Set* e O(E) para as operações de *Find-Set* e *Union*.

Assim, podemos concluir que o algoritmo de *Kruskal* tem complexidade **O(E log V)**, e portanto o nosso programa também assegura essa mesma complexidade.

ANÁLISE EXPERIMENTAL



Variação do número de arestas e vértices (E log V) em função do tempo de execução do programa

Apoiados pelo gráfico acima, podemos verificar que a execução do algoritmo é linear Assim, podemos então comprovar que a complexidade final do nosso programa é **O(E log v)**, tal como esperado na avaliação teórica.