Análise e Síntese de Algoritmos

2016/2017

Relatório do Segundo Projeto

Grupo 127

81186 - Stéphane Duarte

INTRODUÇÃO

No âmbito da cadeira de Análise e Síntese de Algoritmos, foi-nos proposto um projeto cujo objetivo era desenvolver um sistema que ajudasse a decidir que aeroportos e estradas construir para interligar todas as cidades de pequena e média dimensão numa rede de ligações aéres e rodoviárias.

Deste modo, o problema vai ser encarado como um grafo não dirigido no qual terá de ser aplicado um algoritmo de custo mínimo, recorrendo a uma adaptação ao algoritmo Kruskal.

DESCRIÇÃO DA SOLUÇÃO

Este problema é encarado como um grafo não dirigido, em que cada vértice representa uma cidade e cada aresta representa uma ligação entre cidades. É ainda criado um vértice auxiliar que serve como vértice de ligação de cidades que possuem aeroportos. Neste último caso, o valor das arestas que ligam a cidade ao vértice auxiliar é o valor do custo de construção do aeroporto.

A solução foi implementada em Java para facilitar a implementação de ADTs, usando as bibliotecas List, ArrayList e Collections.

Como input, é recebido:

1. Uma linha com o número N de cidades existentes (N ≥ 2), ou seja, o número de vértices do grafo. Este valor é utilizado para a criação do grafo.
2. Uma linha com o número A de aeroportos existentes (0 ≥ A ≥ N). Este número vai ser utilizado no ponto 3.
3. A linhas com dois elementos numéricos, a e c, separados por um espaço, em que c representa o custo de construir um aeroporto na cidade a. Estes valores vão ser utilizados para criar a aresta que liga a ao vértice auxiliar e cujo valor é c.
4. Uma linha com o número E de potenciais estradas a construir. Este número vai ser utilizado no ponto 5.
5. E linhas com três elementos númericos, a b e c, separados por um espaço, em que c representa o custo de construir uma estrada que ligue a a b. Estes valores vão ser utilizados para criar a aresta que liga a a b e cujo valor é c.

Posteriormente, é feita a ordenação das arestas, utilizando o sort da API Collections, que ordena em primeira instância por custo e, em caso de empate, coloca as ligações rodoviárias antes das ligações aéreas.

A este ponto, o programa está pronto a executar a parte seguinte à ordenação do algoritmo de Kruskal. À exceção da ordenação, este algoritmo é executado duas vezes. Primeiramente, apenas com as arestas que representam estradas. De seguida, com todas as arestas (estradas e ligações aéreas). Há necessidade da dupla execução para excluir potenciais falhas de construção de aeroportos desnecessários. Foi também adaptado ao problema o algoritmo union-find para a deteção de ciclos. O objeto cidade tem um atributo para este algoritmo que representa a cidade que o liga ao restante grafo.

Após a dupla execução, os resultados necessitam de ser comparados. A escolha vai recair sobre o resultado com menor custo. Se a execução só com estradas apresentar menor ou igual custo, então essa é a escolha predileta. Se, pelo contrário, o segundo resultado apresentar menor custo, então será essa a escolha, o que significa que existirão, pelo menos, dois aeroportos a construir. Outra alternativa a estes resultados é não ser encontrada nenhuma forma de ligar todas as cidades.

Como output, obtém-se:

1. Uma linha com um inteiro c que representa o melhor custo de ligação de todas as cidades.
2. Uma linha com dois inteiros, A e E, separados por um espaço, que representam o número A de aeroportos a construir e o número E de estradas a construir.

NOTA: No caso de não existir uma solução para o problema, o output gerado será “insuficiente”.

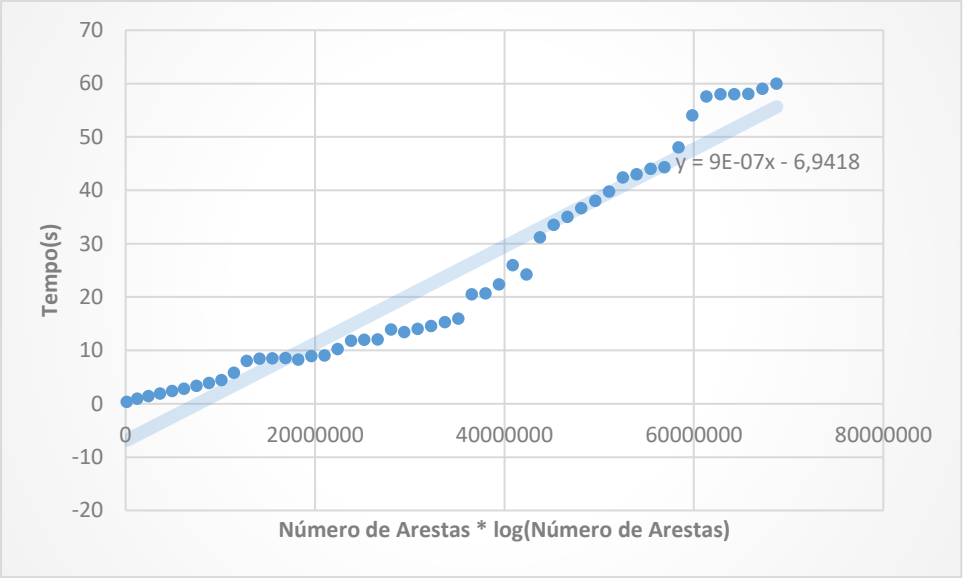
ANÁLISE TEÓRICA

Após cuidada análise ao código, é possível determinar o tempo de execução de cada ciclo e, consequentemente, do programa. Considere-se N o número de vértices do grafo e E o número de arestas.

1. Inicialização do grafo: O(N);
2. Ciclo para gerar as arestas: O(E);
3. Ordenação das arestas: O (E log(E));
4. Kruskal: O (E log(V));

Tendo em conta as complexidades apresentadas anteriormente, concluímos que a complexidade total do programa será: N + E + E log(E) + E log(V) = E log(E).

AVALIAÇÃO EXPERIMENTAL DOS RESULTADOS



Tal como visto na análise teórica, a complexidade do programa é O(E log(E)). Foram então realizados testes experimentais para comprovar este valor.

Após 50 testes, com valores de E entre 0 e 10 milhões, foi verificado que, com o aumento da função E log(E), o tempo de execução aumentava linearmente, como se pode ver no gráfico acima.

Deste modo, verifica-se que a complexidade do programa é, de facto, O(E log(E)).

REFERÊNCIAS

https://pt.wikipedia.org/wiki/Algoritmo_de_Kruskal

http://www.geeksforgeeks.org/greedy-algorithms-set-2-kruskals-minimum-spanning-tree-mst/

http://stackoverflow.com/questions/4254122/what-is-the-time-complexity-of-this-sort-method