Instituto Superior Técnico
Análise e Síntese de Algoritmos – 2º Semestre – 2016/2017
Grupo 18T – Francisco Barros 85069 e Rafael Ribeiro 84758

1. Introdução

O objetivo deste projeto é calcular o custo total mínimo de contruir uma rede de estradas e aeroportos que interligue todas as cidades do "Bananadistão", em que cidades com aeroportos conseguem ligar-se a qualquer outra cidade com aeroporto e cidades sem aeroportos precisam de estradas para estarem ligadas. O programa deve apresentar o custo total mínimo de construção e o número de aeroportos e estradas construídos, ou, em caso de impossibilidade, imprimir para o *standart output* tal .

2. Descrição da solução

Para encontrar a solução do problema, em primeiro lugar, tentou-se transformar os *inputs* em estruturas de dados ***pair <<Connection>, <Cost>>***[1], que representam a ligação da cidade A para a cidade B e o respetivo custo. Esta estrutura representa uma aresta (u,v) e respetivo peso, de um grafo pesado, não dirigido, G=(V,E). Por motivos de coerência é adicionado a G um vértice hipotético denominado ***skyCity***, ao qual todas as cidades com aeroporto se ligam e cujo o peso é o custo de construção do aeroporto na cidade de partida, u. Permitindo assim tratar as arestas, rodoviárias e aéreas, de forma homogénea. Sempre que uma linha do *standart input* é transformada numa aresta, esta é colocada no respetivo(s) ***vector***. As arestas rodoviárias são colocadas em ***roadsVector*** e ***airwaysVector*** ao passo que as arestas aéreas são colocadas apenas em ***airwaysVector***.

Para chegar a uma solução de custo mínimo que ligue todas as cidades são calculadas duas árvores. Ambas resultam da aplicação do algoritmo de ***Kruskal*** aos vetores de arestas de G. A primeira execução considera somente arestas em ***roadsVector*** e a segunda considera todas as arestas de G, guardadas em ***airwaysVector***. Uma árvore é considerada uma árvore abrangente[2] se: "Dado um grafo ligado, não dirigido, G=(V,E) e para o qual cada aresta (u,v)∈E está bem definida a função de peso w(u,v), existe um subconjunto T⊆E que liga todos os vértices de G, tal que |T|=|V|-1 e cujo o peso é mínimo."[3,4] Desta afirmação retira-se ainda o caso particular que a árvore gerada por Kruskal sobre airwaysVector só será MST se |T|=|V| e se e só forem utilizados pelo menos dois aeroportos, uma vez que o vértice skyCity é hipotético e para todos os efeitos não pertence a G e a sua utilização implica o incremento do número de arestas na MST em uma unidade, qualquer que seja o número de aeroportos usado superior a zero e que a construção de um único aeroporto não gera ligações.

---

[1] Um *pair <connection*, *cost>* é uma estrutura de dados que contem uma *Connection*, que é também uma estrutura do tipo *pair<cityA, cityB>*, e um *cost*. *CityA*, *cityB* e *cost* são inteiros. E representam arestas do grafo G
[2] MST – Minimum Spanning Tree – Árvore abrangente de menor custo
[3] Cormen et al. (2009). *Introduction to Algorithms*, 3rd Edition. Massachussets, The MIT Press. (Chap.23.1, 624)
[4] Russo, Luís. (2017). *Aula10*. [Slide 12-13]. Em: https://fenix.tecnico.ulisboa.pt/downloadFile/1970943312285705/aula10.pdf

A motivação por trás da utilização de duas execuções de **Kruskal** deriva dos pré-requisitos do problema, em que se houver mais do que uma solução ótima, escolhe-se a que tiver o menor número de aeroportos[5]. Assim, calculam-se as árvores da rede de estradas (T_1) e a árvore da rede completa, com estradas e aeroportos, (T_2) e caso sejam ambas MST, escolhe-se a solução que tiver o menor custo. Caso sejam ambas MST e tenham o mesmo custo, escolhe-se a solução associada a T_1, para priorizar a rede com menor número de aeroportos. Caso apenas uma árvore seja MST, escolhe-se a solução associada a essa mesma MST. Caso contrário imprime-se para o *standart output* a *String* "Insuficiente".

Como já foi referido foi utilizado o algoritmo de **Kruskal** para obter as possíveis MST do grafo, **G**, que representa o "Bananadistão". No caso genérico, para que o algoritmo execute corretamente, começam-se por criar estruturas que guardem os predecessores e alturas de cada vértice **v∈V** de **G**[6], gerando assim |V| árvores cada um com um vértice de **G**. De seguida as arestas de **G**, representadas pelo par referido no início desta secção são organizados por ordem crescente de peso, esta ordenação permite que no ciclo que a sucede, o algoritmo seja capaz de adicionar sempre que possível à floresta **T** o conjunto cuja a aresta que os une tenha peso mínimo iterando sobre as arestas ordenadas. Esta adição a **T**, faz-se através das heurísticas de compressão e união de caminhos, apenas quando os predecessores dos vértices **u** e **v**, não forem os mesmo, isto é, se pertencem a árvores diferentes, uma vez que uma união nessas condições criaria um ciclo dentro da árvore.[7] A existência de tal ciclo faria com que a soma dos pesos dos arcos na floresta deixasse de ser minimizada e passaríamos a quebrar o teorema que: "Se um grafo ligado árvore então existe no máximo um caminho simples entre cada par de vértices."[8] Após analisadas todas as arestas ordenadas de **G**, a floresta **T** será constituída por uma única árvore, abrangente ou não, ou por um conjunto de várias árvores. Pelo que o *output* do programa dependerá das condições anteriormente enunciadas.

3. Análise Teórica da Solução Particular

Em qualquer notação assintótica referida nesta ou na próxima secção, assume-se que V corresponde ao número de cidades do *input* e E corresponde ao número de estradas mais o número de aeroportos dados. Abaixo encontram-se as complexidades temporais de cada etapa envolvida na solução desenvolvida pelo grupo para o problema proposto, implementada em C++. O pseudocódigo genérico do algoritmo pode ser consultado nas páginas 571 e 631 do livro *Introduction to Algorithms* de Thomas *et al*, já

---

[5] A priorização funciona uma vez que ***airwaysVector*** se encontra ordenado por peso da aresta, como é requerido pelo algoritmo de ***Kruskal***, mas dentro dos mesmos subintervalos de peso os caminhos aéreos vêm depois das estradas, pois têm um peso nulo acrescido, identificado pela cidade destino ser **skyCity**.

[6] Cormen et al. (2009). *Introduction to Algorithms*, 3rd Edition. Massachussets, The MIT Press. (Chap.21.3, 570-571)

[7] Cormen et al. (2009). *Introduction to Algorithms*, 3rd Edition. Massachussets, The MIT Press. (Chap.21.3, 631-632)

[8] Cormen et al. (2009). *Introduction to Algorithms*, 3rd Edition. Massachussets, The MIT Press. (Appendix, 1174-1175)

referido diversas vezes em nota de rodapé, note-se que a implementação genérica não se mapeia a cem porcento na implementação da solução do grupo, pelo que pode ser consultada apenas como referência.

- O programa começa por criar dois ***vector*** em Θ(1).
- A leitura de arestas do *standart input* e adição aos respetivos vetores. Corre em tempo O(KE)[9] com custo Θ(1), pela inserção ser feita no fim do vetor.
- A ordenação de cada vector ocorre em tempo O(N*LgN)[10], em que N é o subconjunto de arestas de **G** inseridas no dito vector. Este tempo encontra-se de acordo com tempo esperado de ordenação para a implementação assintoticamente mais rápida de **Kruskal** do caso genérico que é O(E*LgE).[11]
- As operações ***roadsMakeSet*** e ***airwaysMakeSet*** têm ambas custo O(|V|). A inicialização dos quatro ***array*** de ***int***, como a atribuição de predecessores e altura a um vértice têm custo Θ(1). Cada ciclo faz este conjunto de tarefas |V| vezes.[10]
- Ao longo da execução do programa, nomeadamente durante a execução de ***Kruskal***, as chamadas aos métodos ***FindSet*** ocorre no pior caso O(N), vezes por cada execução de ***Kruskal***, em que N o número de arestas em cada um dos vectores. Caso a união de árvores ocorra, a operação ***UniteSet*** tem custo Θ(1) pois trata-se meramente da mudança de valores de variáveis e acessos a posições dos ***array***, tudo em tempo constante.[10]
- Como assumimos que o grafo **G** representativo do "Bananadistão" é ligado então sabemos que a #E ≥ #V-1, o que segundo *Thomas et al*[10] é suficiente para provar, com as estruturas de dados usadas (Conjuntos Disjuntos e Heurísticas de União e Compressão de caminhos), que as operações de manipulação da floresta **T** desta parte do algoritmo correm em O(E*LgE). Como não podemos ter mais do que V^2 arestas, então $V-1 \leq E \leq V^2$, pelo que cada ***Kruskal*** deverá correr em O(E*LgV).

Resumidamente:

- Criação de dois vetores e inserção de arestas nos mesmos: Θ (1);
- Leitura, criação e adição das ligações à estrutura: O(E);
- Execução do algoritmo de *Kruskal* duas vezes: O(2ELogV) = O(E*LgV);
- **Logo, o nosso programa corre em tempo O(E*LgV)**. Mas que à medida que o grafo se torna cada vez mais denso este tempo piora de forma crescente, mas não abrupta, pois como tivemos oportunidade de observar ***Kruskal*** depende maioritariamente do número de arestas de **G**.

---

[9] A constante K∈ℕ depende da relação de aeroportos e estradas dados no *standart input*.
[10]AnonymousAuthor. (2000-2017). *List::sort – C++ Reference.* Em: http://www.cplusplus.com/reference/list/list/sort/
[11] Custo encontra-se em conformidade com a análise algoritmítica encontrada em Cormen et al. (2009). *Introduction to Algorithms*, 3rd Edition. Massachussets, The MIT Press. (Chap.23.2, 633).

4. Análise experimental dos resultados da solução implementada

Para conferir que a análise teórica por escrita acima, baseada nas várias referências bibliográficas consultadas, testamos o nosso programa com *inputs* válidos. Executaram-se 49 testes com *inputs* válidos diferentes. O número de vértices variou de 10 a 1 milhão, em que o número de máximo de arestas do grafo equivale ao dobro do número de vértices do mesmo.

Os tempos de execução cada um dos testes encontram-se no gráfico abaixo e representam a linha inferior do gráfico. A linha superior é a função do limite assimptótico superior teórico: E*lgV.

O eixo das abcissas representa o número de vértices do *input*. O eixo das ordenadas representa o tempo de execução (para a linha de baixo) e o resultado da função E*lgV para a linha de cima.

Na análise teórica chegou-se à conclusão teórica que o programa corria em O(E*lgV). No gráfico abaixo verifica-se essa análise teórica devido ao facto de a linha que representa o limite assimptótico superior ter a mesma forma que a linha que liga os tempos de execução do programa.

Teoriza-se que a linha de cima tem valores muito superiores pelo facto de que uma instrução não demorar 1 segundo a correr, e ter-se de multiplicar por uma constante na ordem de $1*10^{-4}$ para que as linhas se alinhem.

Conclui-se assim que a análise teórica está correta e que a complexidade do programa é O(E*lgV).

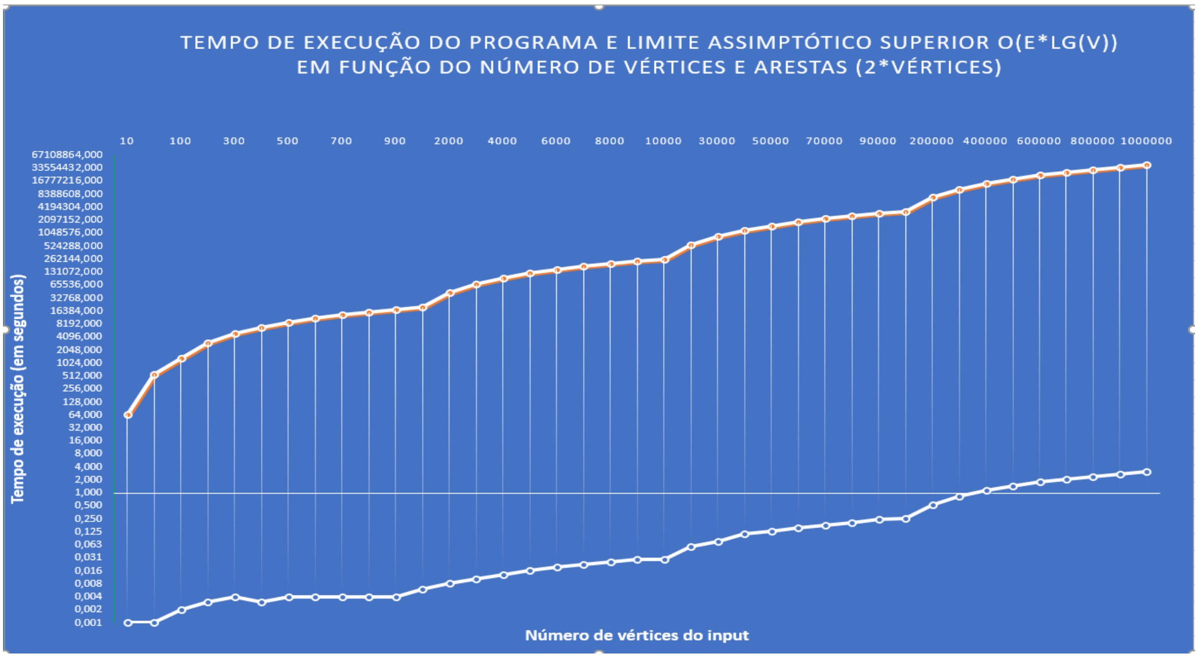

*Fig1: Tempo de execução e limite assimptótico superior teórico em função d número de vértices (V) e arestas (2*V) inseridos*