Relatório 2º projecto ASA 2022/2023

Grupo: TP053

Aluno(s): Rui Amaral (103155)

Descrição do Problema e da Solução

Para resolvermos o problema, temos que descobrir o valor máximo de trocas entre o maior número de regiões possível recorrendo ao número mínimo de troços de ferrovia. Se transformarmos o problema num grafo em que cada vértice representa uma região e cada aresta, um troço de ferrovia, o problema simplifica-se.

Em cada grafo, queremos descobrir um aglomerado de arestas que ligue o maior número de pontos do grafo (**V**), e que contenha o menor número de arestas possível (no máximo terá **V - 1** arestas), finalmente, se houver mais que um grupo de arestas que respeite essas condições, escolhemos o que tem o valor de trocas total mais elevado, valor este que é a solução para cada problema.

A solução de cada situação é no fundo a soma dos pesos de uma *Maximum Spanning Tree* do grafo. Em aula, falámos de algoritmos para descobrir *Minimum Spanning Trees* mas, com algumas adaptações conseguimos usar esses mesmos algoritmos para resolver este problema. Neste caso, foi escolhido o algoritmo de Kruskal para tal.

O algoritmo de Kruskal ordena todas as arestas pelo seu peso por ordem crescente e depois itera sobre elas, escolhendo assim as arestas com o menor peso possível para a *spanning tree*. No entanto, ao alterar a ordenação das arestas para uma ordem decrescente, consegue-se que o algoritmo obtenha uma *Maximum Spanning Tree*.

Análise Teórica

Seja **V** o número de vértices e **E** o número de arestas de um grafo:

Durante a execução do programa temos as seguintes fases:

- Leitura dos dados de entrada: simples leitura do input, com ciclo(s) a depender linearmente de **E**. Logo, **O(E)**.
- Inicialização dos vetores do *rank* e dos pais de cada vértice a depender linearmente de **V**. Logo, **O(V)**.
- Chamada de função que implementa o algoritmo de Kruskal:
    - Ordenação das arestas por ordem decrescente (recorrendo à função `std::sort`). Logo, **O(E*log(E))**.
    - **O(E)** operações das funções **Union** e **Find-Set**.
- Sabemos que, com a implementação de mecanismos de compressão de caminhos e união por *rank*, o algoritmo de Kruskal tem uma complexidade temporal de **O(E*log(E))**.
- Apresentação do resultado em tempo constante. Logo, **O(1)**.

Conseguimos assegurar uma complexidade global da solução: **O(E*log(E))**

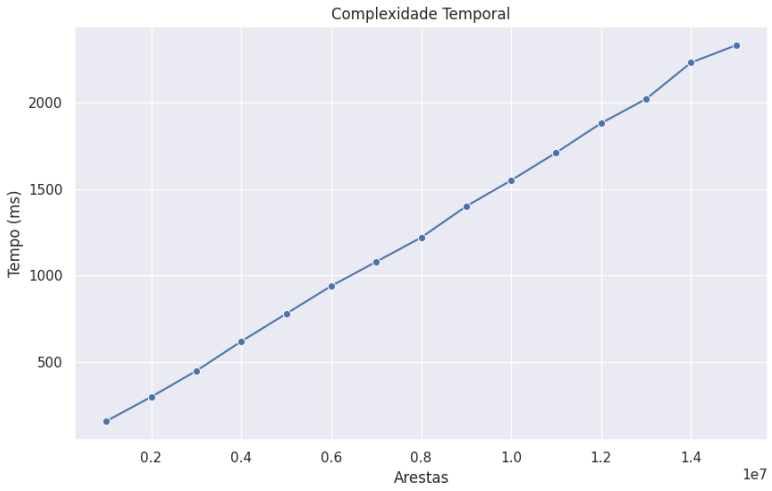
Relatório 2º projecto ASA 2022/2023

Grupo: TP053

Aluno(s): Rui Amaral (103155)

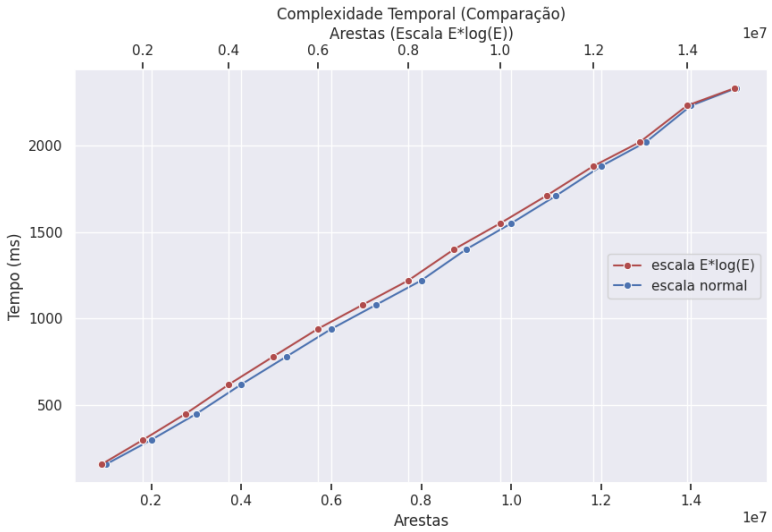
Avaliação Experimental dos Resultados

Foi feito o registo do tempo que o programa demora a resolver problemas com grafos densos. Seja **E** o número de arestas de cada grafo denso. Ao fazer este registo para 15 valores de **E** (de 1000000 a 15000000), obtemos o seguinte gráfico:



À primeira vista, pode parecer que o resultado é uma complexidade temporal linear.

No entanto, ao mudar a escala do eixo **x** para **E*log(E)** para confirmar a complexidade calculada obtemos uma segunda linha:



Que, quando comparada com o gráfico anterior, revela que a linha do primeiro gráfico apresenta uma subtil curva, enquanto que a linha com a escala ajustada se aproxima mais a uma reta. Como ao mudar a escala para **E*log(E)** obtemos uma reta, conseguimos provar que a complexidade prática da solução é equivalente à complexidade teórica.