

# 1 Descrição do Problema e da Solução

O problema proposto foi o de ajudar um governo num país imaginário, Caracolândia, a encontrar o número máximo de trocas comerciais entre regiões vizinhas passíveis de serem transportadas para uma rede ferroviária, minimizando os custos de financiamento dos troços ferroviários.

Para resolver o **problema** recorreu-se a uma implementação modificada do algoritmo de Kruskal para minimizar a quantidade de arestas num grafo, maximizando o seu peso. O problema é modelado por um **grafo de pesos não direcionado** e por isso podemos aplicar o algoritmo mencionado. Um **vértice** no grafo representa uma **região** no país imaginário. Uma **aresta** representa uma **conexão entre duas regiões**, entre as quais existem **trocas comerciais**, que representam o **peso** dessa aresta.

Assim, na modelação da solução, a informação das arestas recebida como input é representada por uma *struct* que guarda os identificadores dos dois vértices e o peso dessa aresta. O algoritmo começa por ordenar as arestas do grafo pelo seu peso em ordem decrescente, pois assim permite ao algoritmo de Kruskal escolher sempre, das arestas que faltam, a que tem o maior número de trocas comerciais. Em seguida, percorre o vetor ordenado e adiciona vértices a uma estrutura de dados designada por **conjunto disjunto**. O algoritmo utiliza a estrutura de dados para garantir que apenas liga à árvore as arestas que não criam ciclos. Faz isto, utilizando a operação **findSet** para verificar se os dois vértices de uma aresta já se encontram no mesmo conjunto, não os unindo nesse caso. Caso contrário, a operação **merge** une os conjuntos de cada vértice e o peso da aresta que liga esses vértices é adicionado ao peso total. Deste modo, garantimos que todos os vértices que estavam ligados ainda têm um caminho entre eles e por isso mantemos apenas as ligações necessárias, sem ciclos, minimizando o número de arestas.

No final do algoritmo, o peso máximo das arestas da árvore é retornado, que no contexto do problema representa o número máximo de trocas comerciais passíveis de serem passadas para a rede ferroviária.

## 2 Análise Teórica

Seja  $V$  o número de vértices e  $E$  o número de arestas.

- Simples leitura do input, guardando a informação das arestas em estruturas, adicionando-as a um vetor. Logo, assumindo input correto,  $\Theta(E)$ .
- A operação de inicializar os vetores nos conjuntos disjuntos, **makeSets**, é realizada em tempo linear. Logo,  $\Theta(V)$ .
- Para ordenar as arestas por ordem decrescente de peso é utilizado um algoritmo de ordenação eficiente do tipo *comparison sort*. Logo,  $O(E \log E)$ .
- São realizadas  $O(E)$  operações de **findSet** e **merge**.
- Através do uso de conjuntos disjuntos, com as heurísticas de união por categoria e compressão de caminhos, podemos concluir que a complexidade do *for loop* é  $O(Ea(V))$ .
- A função  $a(V)$  corresponde à inversa da função de Ackermann[1], sendo sub-logarítmica e, portanto, de crescimento prolongado.
- Em qualquer aplicação concebível de um conjunto disjunto,  $a(V) \leq 4$ . Assim, podemos tomar a complexidade do *for loop* como linear em  $E$ . Estritamente falando, porém, na realidade é pior que linear.
- A apresentação do resultado é feita em tempo constante. Logo,  $\Theta(1)$ .

Assim, no pior caso tem-se uma complexidade temporal  $O(E \log E)$ , pois é o termo dominante de todas as complexidades analisadas.

A solução tem complexidade espacial  $\Theta(E + V)$ , visto que se utilizou um vetor, de tamanho  $E$ , para todas as arestas do grafo e para os conjuntos disjuntos foram necessários mais dois vetores de tamanho  $V$ , um para os vértices representantes de cada vértice e outro para as categorias de cada vértice representante.

### 3 Avaliação Experimental dos Resultados

Foram gerados vários grafos com um número de arestas entre 10 e 10 000 000, todos diferentes entre si. Para cada ordem de grandeza de arestas foram gerados pelo menos 100 grafos, tendo um total de 541 grafos. O programa foi executado pelo menos 100 vezes para cada grafo, recorrendo ao programa de *benchmarking* *hyperfine*.

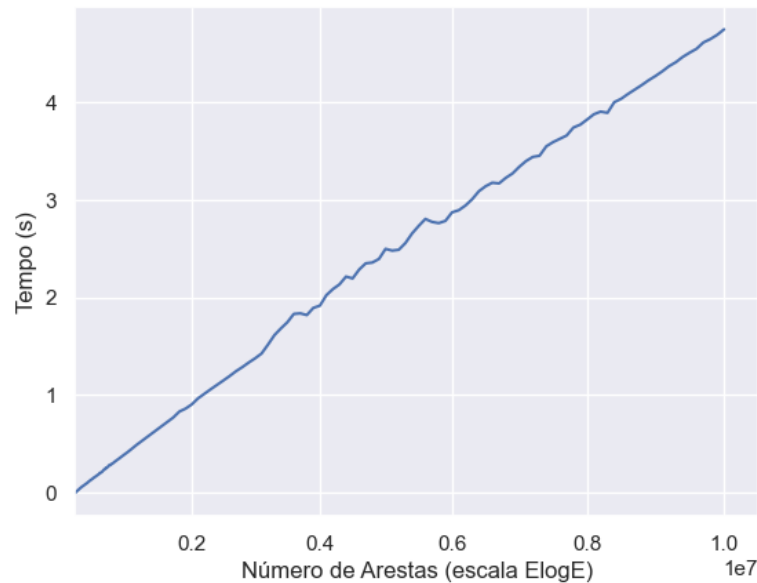


Figure 1: Complexidade temporal do problema proposto

O gráfico apresentado tem o número de arestas no eixo dos  $xx$  com uma escala  $E \log E$  e o tempo em segundos no eixo dos  $yy$ . Os dados revelam uma tendência linear, comprovando que a complexidade temporal do problema é, num caso geral,  $O(E \log E)$ , tal como concluído na análise teórica.

### Referência

[1] Wikipedia contributors. *Ackermann function* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2022]. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Ackermann\\_function&oldid=1123526697](https://en.wikipedia.org/w/index.php?title=Ackermann_function&oldid=1123526697).