Projeto 2 ASA

93075 Gonçalo Azevedo 99205 Diogo Vieira

January 5, 2023

1 Descrição do problema

Procura-se obter o **maior valor** possível de trocas comerciais minimizando o número de troços necessários numa rede de regiões da Caracolândia ligadas por ferrovia. O problema trata-se assim de encontrar uma **MST** para o grafo $G = (V, E)$ não dirigido onde cada V simboliza uma região e E simboliza um conjunto de troços definidos como próximos após uma triangulação de **Delauny** com um valor de trocas comerciais associado. Neste caso, o objetivo passa por calcular o valor de uma **Maximum Spanning Tree**, cujo processo é semalhante a qualquer algoritmo abordado em aula de cálculo de uma **Minimum Spanning Tree**. Uma vez que há a probabilidade de existirem subcomponentes disconectas no grafo o algoritmo de **Prim** não será um bom candidato para este problema, logo, iremos recorrer ao algoritmo de **Kruskal** com **Disjoint Sets**. Foi então feita a implementação de uma estrutura que nos permita utilizar as propriedades **Union** e **Find** de um **Disjoint Set** e, no fim, foi aplicado o algoritmo de **Kruskal** ao conjunto de arestas (E) ordenadas por ordem **decrescente** do seu peso. Será utilizado um **Disjoint Set** na forma de "árvore" com **union by rank** e **path compression**.

2 Análise Teórica

- **Input**: A leitura do input é caracterizada por um simples loop que é executado $|E|$ vezes para ler todas as arestas do grafo, onde cada aresta é definida pelos vértices que conecta ($v1$ e $v2$) e pelo seu peso (w).

  Let $Edges$ be a new array
  for $i \leftarrow 1$ to E **do**
  read Edge into $Edges[i]$
  end for

  A complexidade desta operação é assim $\mathbf{\Theta(E)}$.

- **Inicialização do Disjoint Set**: Após a leitura dos dados, é necessário a inicialização da estrutura **Disjoint Set** para a utilização do algoritmo de Kruskal. Esta estrutura guarda em si dois vectores, um que mantém informação sobre o **representante** de cada vértice e outro sobre o **rank** de cada vértice (estimativa de tamanho da árvore originada pelo vértice). Assim, é necessário inicializar o vetor P (de *parent*), sinalizando cada vértice como o seu próprio representante.

  Let P be a new array
  Let R be a new array of 0's
  for $i \leftarrow 1$ to V **do**
  $P[i] \leftarrow i$
  end for

  Podemos assim concluir que a complexidade desta operação é de $\mathbf{\Theta(V)}$.

- **Ordenação de Edges por peso**: Para finalizar, antes da aplicação do algoritmo de Kruskal, iremos proceder à ordenação do vector de arestas por ordem **decrescente** do valor de peso, uma vez que optámos por não utilizar uma MaxHeap na leitura das arestas. É utilizada a função *sort* incluida na **STL** do C++. Este algoritmo apresenta uma compleidade de $\mathbf{O(Elog(E))}$, onde E simboliza o número de arestas.

- **Algorimo de Kruskal**: Por fim será aplicado o algoritmo de **Kruskal** para se obter o peso total de uma **Maximum Spanning Tree** do grafo G.

  $total = 0$
  for all $(u, v) \in Edges$ **do**
  if $Find(u) \neq Find(v)$ **then**
  $Union(u, v)$
  $total += weight((u, v))$
  end if
  end for

Para o algoritmo foi utilizado um **Disjoint Set** recorrendo a **union by rank** e **path compression**, assim, como se encontra demonstrado na Secção 21.4 do livro CRLS, temos que o número de m operações sobre n elementos é $O(m \cdot \alpha(n))$, onde α é uma função sub logaritmica. Podemos então concluir que o loop do pseudo-código apresentado se resume à complexidade $O(E \cdot \alpha(V))$. Para efeitos práticos consideramos $\alpha(V) \leq 4$ e concluimos que a complexidade total desta secção será $\mathbf{O(E)}$.

Assim, temos que a complexidade final deste algoritmo será $\mathbf{O(V + Elog(E))}$.

3 Avaliação experimental dos resultados

Decidimos então testar múltiplos grafos igualmente densos (gerados pelo programa contido em *dgg.c*) com número de vértices até 20000 (ficheiros de 3 GB!). Com a variação dos eixos x definida para $V + Elog(E)$ obtivemos o seguinte gráfico que comprova a complexidade prevista na análise teórica. De notar que, fazer variar o x com $Elog(V)$ ou $Elog(E)$ produziram resultados semelhantes de gráficos com crescimento linear, algo que corrobora a complexidade do algoritmo de Kruskal abordado em aula.