Relatório 2º Projeto ASA 2022/2023

Grupo: AL011
Alunos: Raquel Braunschweig (102624), Madalena Cabrita (103595)

Descrição do Problema e da Solução

O problema apresentado consiste em conseguir determinar o valor máximo de trocas comerciais entre regiões, usando o número mínimo de infraestruturas possível. Este problema traduz-se para uma representação em grafo, na qual as regiões correspondem aos vértices(V), os troços entre estas (infraestruturas) correspondem aos arcos(E) e o valor das trocas comerciais é dado pelo peso dos arcos.

De forma a resolver este problema, decidiu-se criar uma adaptação do algoritmo de Kruskal. O algoritmo de Kruskal tem como objetivo obter a árvore que contém todos os vértices do grafo e cujo peso total (soma do peso das sub-arestas) é o mínimo possível. Desse modo, escolhem-se primeiro os arcos com menor peso e constrói-se a árvore partindo destes para os de maior peso. No entanto, como este projeto pede o cálculo do valor máximo de trocas comerciais, o "nosso" algoritmo de Kruskal constrói a árvore começando pelos arcos de maior peso, conseguindo determinar assim o maior peso total usando a menor quantidade de arcos.

Análise Teórica

Seja V o número de vértices e E o número de arcos do grafo representativo do problema.

Leitura do número de vértices e leitura do número de arcos: simples leitura de input **Θ(1)**;

Inicializar vetor que contém todos os arcos pairs[]: linear com o número de arcos, **O(E)**;

Leitura e inserção dos arcos no vetor pairs[]: linear com o número de arcos, **O(E);**

Calcular peso máximo:

```
max_work<-0
inicializar vetor paths[] que conterá árvores(representadas por vértice inicial, vértice final e profundidade)
for i=0 to i=V do
  parent of paths[i]<- -1
  rank of paths[i]<-0
sort(pairs[])
j<-0
for each edge (u,v) in graph do
  if find(u)!=find(v)
      max_work+= edge's weight
      unite(parent of u, parent of v)
      j++
      if j==V-1
         break
      endif
   endif
endfor
return max_work
```

O(V)

Ordenar arcos por ordem decrescente de peso, usando quicksort, logo **O(ElogE)**

O(E), depende do número de arcos

O(ElogV), depende do loop exterior e da complexidade de find()

O(1), função baseada em comparações e atribuições de valor, operações constantes

find(vertice): O(log V)

```
if parent of paths[vertice]==-1
```

Procura recursivamente o pai da árvore onde o vértice se encontra. Como o unite() faz com que a raiz da menor árvore passe a apontar para a raiz da maior, então a complexidade de find torna-se O(logV)

return vertice

parent of paths[vertice]<- find(parent of paths[vertice])

return parent of paths[vertice]

Apresentação de resultado final: O(1);

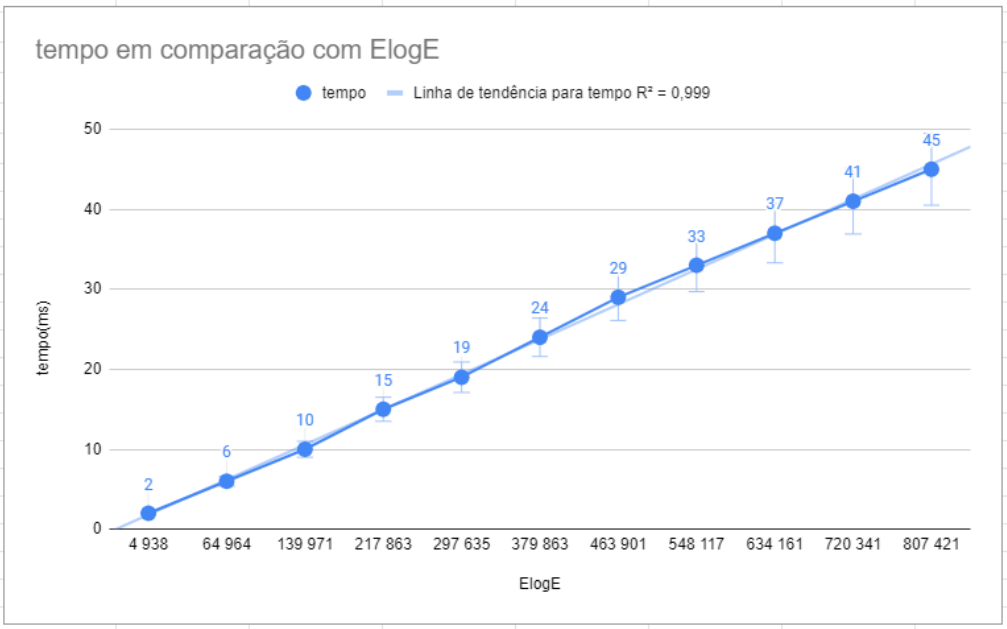
Complexidade total da solução: corresponde à maior complexidade apresentada anteriormente que é **O(ElogE)**, pois, no pior caso, existem mais arcos do que vértices**.**

Avaliação Experimental dos Resultados

Usando como exemplos 11 grafos de Delauney, gerados pelo gerador dgg com semente aleatória 15 e coordenada máxima igual a V*10, obteve-se o seguinte gráfico.

(os valores de ElogE estão arredondados às unidades)

vértices	arcos	ElogE	tempo (ms)
1000	1548	4 938	2
10000	15503	64 964	6
20000	31150	139 971	10
30000	46662	217 863	15
40000	62097	297 635	19
50000	77677	379 863	24
60000	93339	463 901	29
70000	108824	548 117	33
80000	124466	634 161	37
90000	139979	720 341	41
100000	155519	807 421	45



Observa-se que o gráfico tende a ser aproximadamente linear, podendo-se assim concluir que a nossa implementação está de acordo com a análise teórica.