**Descrição do Problema e da Solução**

Para resolver o problema foi utilizado o algoritmo de Kruskal com a modificação onde as arestas a ser processadas são ordenadas de forma decrescente de peso para assim ser possível ter o número máximo de trocas, com o mínimo custo. Foi utilizada a estrutura 'DisjointSets' para representar conjuntos disjuntos, onde inicialmente o pai de cada vértice é inicializado como o próprio, e os ranks inicializados a 0. Existem depois as funções 'find(x)' que coloca todos os vértices a apontar para a raiz do conjunto, realizando a compressão do caminho; e 'union(x, y)' que une as raízes dos conjuntos onde se encontram x e y, tendo em conta os ranks. É também implementada a estrutura 'Graph' que representa o grafo com V vértices e E arestas, e um vetor de arestas (definidas pelo vértice de ínicio, fim e peso) fornecidas no input. Aqui também se encontra a função principal do algoritmo de Kruskal que percorre as arestas e adiciona à MST se não for criado um ciclo retornando por fim a solução.

**Análise Teórica**

* Leitura dos dados de entrada: O(E)
* Guardar as ligações entre vértices: O(V)
* Função find: O(logV)
* Função unite: O(logV)
* Função addEdge: O(1)
* Kruskal: O(E logE)

Pseudocódigo da função principal:

Kruskal() {

sol = 0

sort(edges.begin(), edges.end(), [](a,b) {

return a.weight > b.weight})

DisjoinSets ds(V)

for e to edges do

u = ds.find(e.start)

v = ds.find(e.end)

if (u ≠ v) {

sol = sol + e.weight

ds.unite(u, v)

}

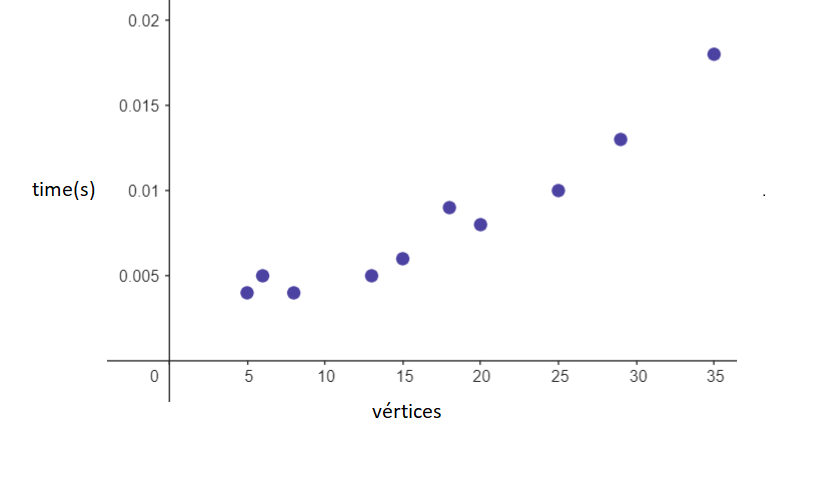
}

return sol

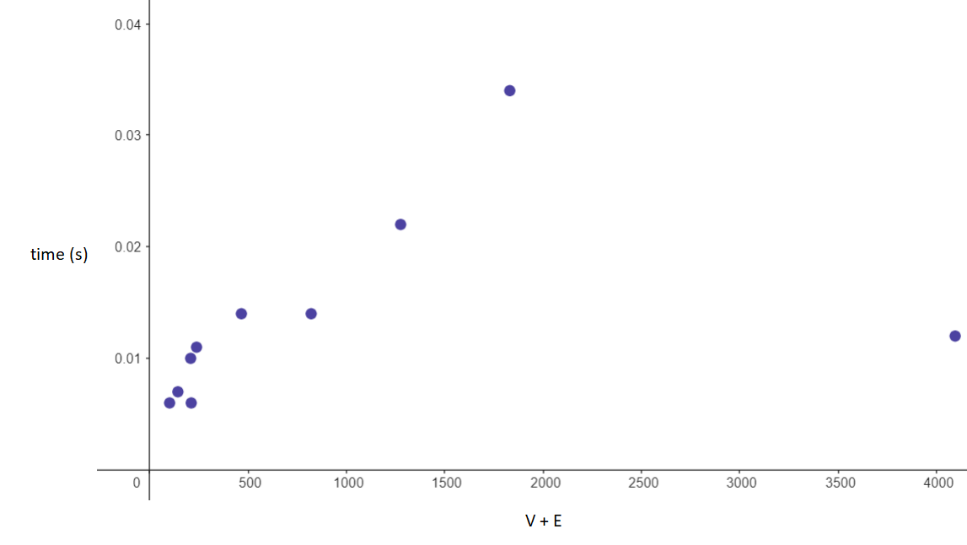
}

Complexidade global da solução: O(E logV)

**Avaliação Experimental dos Resultados**



O gráfico do tempo em função dos vértices tem complexidade O(E logV)



Como é possível observar, não se conseguiu atingir a complexidade desejada, pois tem uma complexidade O(E logV).