Trabalho Prático 3 - AEDS III

- *André Augustho Sarti Trindade Silva - 2021.1.08.001*
- *Vinicius Eduardo de Souza Honório - 2021.1.08.024*

RESOLUÇÃO

Na busca de uma heurística construtiva para solução do problema, utilizamos primeiramente a ordenação de vetores por *Quick Sort* e tomamos de base os dois vértices de maior aresta. Assim, com a lógica similar à do algoritmo de Prim, pôde-se criar um caminho sem ciclo (que é um tipo de árvore binária) até formar-se um subgrafo de 5 vértices.

Já para busca largura, a solução obtida anteriormente é modificada de forma a verificar se a alteração de um dos vértices desta gerará uma solução melhor ou não para o problema em questão.

CONCLUSÃO

A dificuldade na resolução foi a criação de uma árvore binária, porém, o caminho formado já seria uma árvore, então pouco nos importou outros modelos de resposta.

Algumas instâncias não puderam ser executadas devido a estouro de memória, pois o programa não suportava uma matriz de tamanho tão grande quanto necessário

Diferentemente do das expectativas: o processo de busca largura feito, por exemplo, 1000 vezes, foi extremamente rápido.

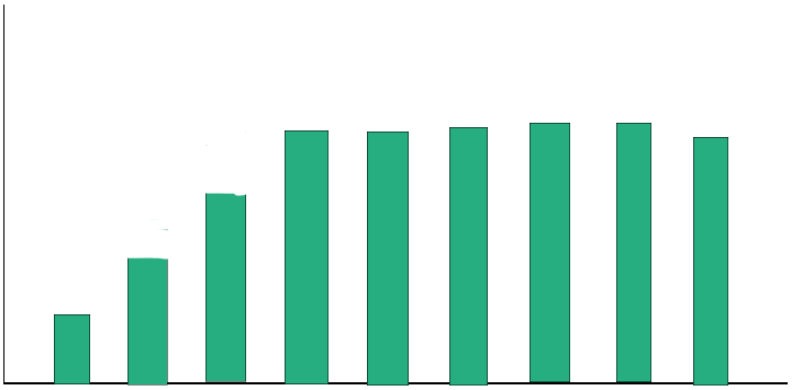
PSEUDOCÓDIGO

```
// Heurística Construtiva

reverseQuickSort(MatrizArestas) // Ordena, em ordem decrescente, a matriz de arestas
Solucao ← Solucao ∪ MatrizArestas[0][1]  ∪ MatrizArestas[0][2] // Adiciona, no vetor Solucao, os dois vértices da aresta mais pesada como base
pontaA ← MatrizArestas[0][1]
pontaB ← MatrizArestas[0][2]
i ← 1
enquanto nElementos(Solucao) < 5 faça
        se matrizArestas[i][1] pertence a {pontoA, pontoB} então // Se a próxima aresta mais pesada tiver um vértice adjacente a pontaA ou pontaB, então este vértice será adicionado à solução
                se matrizArestas[i][2] não pertence a Solucao então
                        Solucao <- Solucao ∪ MatrizArestas[i][2]
                        se MatrizArestas[i][1] == pontoA então // O vértice adicionado se torna uma nova ponta
                                pontoA ← MatrizArestas[i][2]
                        senão
                                pontoB ← MatrizArestas[i][2]
                        fimse
                fimse
        fimse
        se matrizArestas[i][2] pertence a {pontoA, pontoB} então
                se matrizArestas[i][1] não pertence a Solucao então
                        Solucao ← Solucao ∪ MatrizArestas[i][1]
                        se MatrizArestas[i][2] == pontoA então
                                pontoA ← MatrizArestas[i][1]
                        senão
                                pontoB ← MatrizArestas[i][1]
                        fimse
                fimse
        fimse

        i++
fimenquanto
```

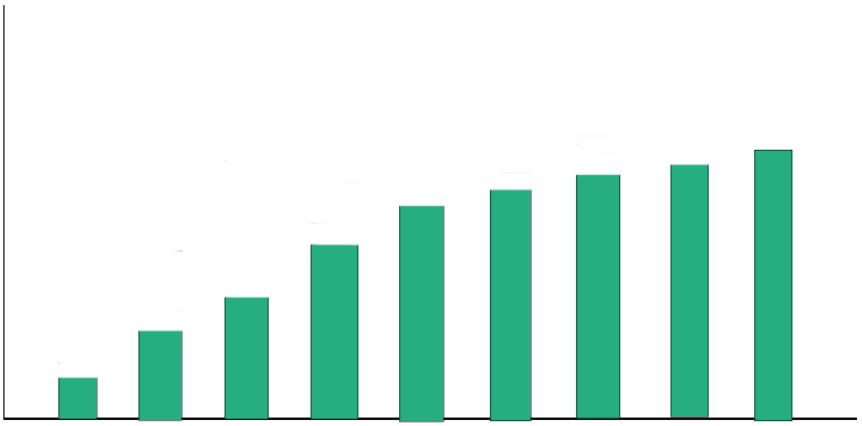
```
// Busca Local

v ← 1
enquanto condiçãoDeParada == Falsa faça
        para i ← 1; i <= 5; i++ faça
                SolucaoAux ← Solucao
                soma(SolucaoAux,i,v) // soma: Soma v à posição SolucaoAux[i] do
vetor, garantindo SolucaoAux[i] <= nVertices
                se custo(SolucaoAux) > custo(SolucaoMaior) entao
                        SolucaoMaior ← SolucaoAux
                fimse
                SolucaoAux ← Solucao
                subtrai(SolucaoAux,i,v) // subtrai: Subtrai v à posição SolucaoAux[i] do
vetor, garantindo SolucaoAux[i] >= 1
                se custo(SolucaoAux) > custo(SolucaoMaior) entao
                        SolucaoMaior ← SolucaoAux
                fimse
        fimpara
        v++
fimenquanto
```

RESULTADOS

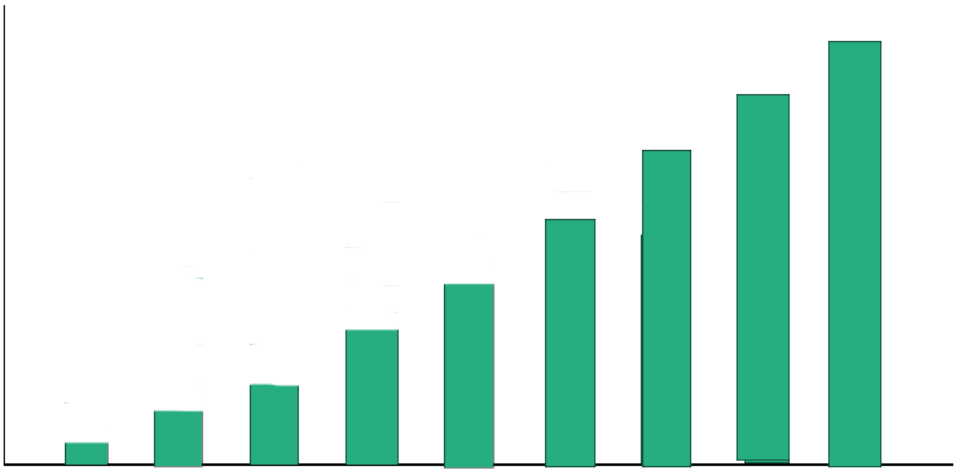
10 Segundos



30 Segundos



60 Segundos



a28.txt
78 1 96 1 95
1203

att48.txt
19 4 17 4 37
33392

att532.txt
472 1 489 1 465
35188

berlin52.txt
42 52 2 52 7
6792

bier127.txt
104 98 128 101 107
129464354

brd14051.txt
terminate called after throwing an instance of 'std::length_error'
  what(): vector::_M_default_append

burma14.txt
1 5 10 4 15
1997164871

d15112.txt
terminate called after throwing an instance of 'std::length_error'
  what(): vector::_M_default_append

eil101.txt
86 65 38 65 43
350