CEFET-MG
Centro Federal de Educação Tecnológica de Minas Gerais

Relatório Prova 3 / TP 3

Problema do Caixeiro Viajante

Bruna Gomes Camilo

Erick Henrique Dutra de Souza

Lucas Cota Dornelas

Setembro de 2021

Laboratório de Algoritmo e Estrutura de Dados II

1. Descrição

O Problema do Caixeiro Viajante é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

Para a solução desse problema, utilizamos estratégias como heurísticas de construção do método guloso, onde a solução foi construída elemento a elemento, seguindo o critério de tempo e distância percorrida para a otimização. Além dessa técnica, utilizamos a força bruta como comparação, que consiste em testar todas as opções possíveis que solucionam o problema e registrar a solução ótima.

2. Implementação

pacote `heuristica`

classe `HeuristicaGulosa`

```java
package heuristica;
import grafo.Grafo;

You, 14 minutes ago | 2 authors (You and others)
// graph:           Grafo      You, 3 hours ago • fix: refactor code heuristica gulosa
// visitsVertices:  Array auxiliar que armazena as visitas á um vertices do grafo
// bestWay:         Melhor caminho/solução
// totalDistance:   Soma do peso de todas arestas

public class HeuristicaGulosa {
    private Grafo graph;
    private boolean[] visitsVertices;
    private int[][] bestWay;
    private int shortestDistance = 0;

    public HeuristicaGulosa(Grafo graph){
        int numVertices = graph.numVertices();

        this.graph = graph;
        this.visitsVertices = new boolean[numVertices];
        this.bestWay = new int[graph.numVertices()][3];

        for(int i=0; i < numVertices; i++){
            this.visitsVertices[i] = false;
        }
    }


    public boolean visitouTodos(boolean[] visitsVertices){
        boolean result = true;
        for (boolean vertice : visitsVertices) {
            if(!vertice){
                result = false;
            }
        }
        return result;
    }

    public int[][] encontraCaminho(){
        int vi=0, verticeX, pesoV, i=0;
        visitsVertices[0] = true;
        for(; visitouTodos(visitsVertices) == false; i++) {
            verticeX=graph.menorListaAdjacencia(vi, visitsVertices).v2();
            pesoV=graph.menorListaAdjacencia(vi, visitsVertices).peso();

            bestWay[i][0] = vi;
            bestWay[i][1] = verticeX;
            bestWay[i][2] = pesoV;

            shortestDistance += pesoV;
            vi=verticeX;
            visitsVertices[verticeX] = true;
        }
        visitsVertices[0] = false;

        verticeX=graph.menorListaAdjacencia(vi, visitsVertices).v2();
        pesoV=graph.menorListaAdjacencia(vi, visitsVertices).peso();

```

```
57          bestWay[i][0] = vi;
58          bestWay[i][1] = verticeX;
59          bestWay[i][2] = pesoV;
60
61          shortestDistance += pesoV;
62
63          return bestWay;
64      }
65
66      public int getPesoTotal(){
67          return shortestDistance;
68      }
69  }
70
```

pacote `heuristica`

classe `TestaHeuristicaGulosa`

```
1   package heuristica;
2
3   import grafo.Grafo;
4   import utils.ReadTSP;
5
6   import java.io.IOException;
    ErickHDdS, 2 days ago | 1 author (ErickHDdS)
7   public class TestaHeuristica {
8
        Run | Debug
9       public static void main(String[] args) throws IOException {
10          ReadTSP file;
11          Grafo grafo;
12          HeuristicaGulosa heuristicaGulosa;
13
14          for (int i = 0; i < 3; i++) {
15              String fileName;
16              switch (i) {
17                  case 0:
18                      fileName = "pa561";      ErickHDdS, 4 days ago • FIX: Ajeitando as ordens
19                      break;
20                  case 1:
21                      fileName = "si535";
22                      break;
23                  case 2:
24                      fileName = "si1032";
25                      break;
26                  default:
27                      fileName = null;
28              }
29
30              file = new ReadTSP(fileName);
31          grafo = file.getGrafo();
32          heuristicaGulosa = new HeuristicaGulosa(grafo);
33          heuristicaGulosa.encontraCaminho();
34
35          System.out.println("Arquivo: " + fileName +
36                             ".tsp\nDistancia calculada pela Heuristica Gulosa: "
37                             + heuristicaGulosa.getPesoTotal());      You, seconds ago • Uncommitted changes
38  }
39
40
```

pacote `forcaBruta`

classe `ForcaBruta`

```java
package forcaBruta;

import grafo.*;
import grafo.Grafo.*;
import java.util.ArrayList;
import java.util.HashMap;

You, seconds ago | 2 authors (Bruna Gomes and others)
// graph:               Grafo
// maps:                Mapa que relaciona o vertice que visitou com a distancia total que percorreu
//                      para visita-lo
// shortestDistance:    Menor distancia (solução ótima)

public class ForcaBruta {
    private Grafo graph;
    private HashMap<Integer, ArrayList<Integer>> maps;
    private int shortestDistance = 0;

    public ForcaBruta(Grafo graph) {
        this.graph = graph;
        this.maps = new HashMap<>();
    }

    public void backtracking(int pointA) {
        ArrayList<Integer> visited = new ArrayList<>();
        ArrayList<Aresta> arestasAdj = this.arestasAdj(pointA);

        // caminha de A até o vertice
        for (int i = 0; i < arestasAdj.size(); i++) {
            int pointC = arestasAdj.get(i).v2();
            visita(visited, pointA, pointA, pointC);
```

```
31          }
32      }
33
34      public ArrayList<Aresta> arestasAdj(int v) {
35          ArrayList<Aresta> arestas = new ArrayList<>();
36          for (int i = 0; i < this.graph.numVertices(); i++)
37              if (this.graph.getPeso(v, i) > 0)
38                  arestas.add(new Aresta(v, i, this.graph.getPeso(v, i)));
39          return arestas;
40      }
41
42      private void visita(final ArrayList<Integer> visited, int pontoInicial, int pointA, int pointB) {
43          int totalDistance;
44          ArrayList<Aresta> arestasAdj = this.arestasAdj(pointA);
45          ArrayList visitedCopy = new ArrayList(visited);
46          visitedCopy.add(pointA);
47          // salta o vertice visitado anteriormente e pula-o
48          for (int i = 0; i < arestasAdj.size(); i++) {
49              int pointC = arestasAdj.get(i).v2();                                    // vertice atual → pointC
50              if (!visitedCopy.contains(pointC)) {
51                  if (visitedCopy.size() == this.graph.numVertices() - 1) {
52                      visitedCopy.add(pointB);
53                      visitedCopy.add(pontoInicial);
54                      totalDistance = 0;
55                      for (int j = 0; j < visitedCopy.size(); j++) {
56                          if (j < visitedCopy.size() - 1)
57                              totalDistance += this.graph.getPeso(
58                                      (int) visitedCopy.get(j),
59                                      (int) visitedCopy.get(j + 1));      You, seconds ago • Uncommi
60                      }
61                      final ArrayList visitedCopy2 = new ArrayList(visitedCopy);
62                      this.maps.put(totalDistance, visitedCopy2);

63                      visitedCopy.remove(visitedCopy.size() - 1);
64                      visitedCopy.remove(visitedCopy.size() - 1);
65                  } else if (pointC != pointB) {
66                      visita(visitedCopy, pontoInicial, pointC, pointB);
67                  }
68              }
69          }
70      }
71
72      public void solucaoOtima() {
73          this.shortestDistance = this.maps.keySet().stream().findFirst().get();
74          for (Integer distance : this.maps.keySet())                             // menor distancia
75              if (distance < this.shortestDistance)
76                  this.shortestDistance = distance;
77          System.out.print("\nMenor caminho: ");
78          for (int j = 0; j < this.maps.get(this.shortestDistance).size(); j++)
79              System.out.print(this.maps.get(this.shortestDistance).get(j) + " ");
80          System.out.println("\nDistancia: " + this.shortestDistance);
81      }
82  }
```

pacote `forcaBruta`

classe `TestaForcaBruta`

```java
package forcaBruta;

import grafo.Grafo;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Random;

You, seconds ago | 3 authors (ErickHDdS and others)
public class TestaForcaBruta {
    static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    public static void criaGrafo(Grafo grafo, int numVertices, int i, int j, int a, int b) {
        Random rand = new Random();
        int n = rand.nextInt(b - a) + a;                                    // numero aleatorio entre a e b
        if (i == numVertices) {
            return;
        } else if (j < numVertices - 1) {
            grafo.insereArestaBidirecionada(i, j, n);
            j++;
            criaGrafo(grafo, numVertices, i, j, a, b);
        } else if (j == numVertices - 1) {
            grafo.insereArestaBidirecionada(i, j, n);
            i++;
            j = i + 1;
            criaGrafo(grafo, numVertices, i, j, a, b);
        }
    }


    Run | Debug
    public static void main(String[] args) throws IOException {
        long time0, time1;
        System.out.print("Numero de vertices: ");
        int numVertices = Integer.parseInt(in.readLine());
        Grafo grafo = new Grafo(numVertices);
        System.out.print("Distancia gerada aleatoriamente do menor para o maior. \nEntre: ");
        int a = Integer.parseInt(in.readLine());
        System.out.print("ate: ");
        int b = Integer.parseInt(in.readLine());
        criaGrafo(grafo, numVertices, 0, 1, a, b);
        System.out.println("Grafo gerado: ");
        grafo.imprime();
        ForcaBruta f = new ForcaBruta(grafo);
        System.out.println("\nPonto de partida: ");
        int pointA = Integer.parseInt(in.readLine());
        time0 = System.nanoTime();
        f.backtracking(pointA);
        f.solucaoOtima();
        time1 = System.nanoTime();
        NumberFormat formatter = new DecimalFormat("#0.00000");

        System.out.println(
                "\nTempo de execução: " +
                formatter.format((time1 - time0) * Math.pow(10, -9)) +
                " segundo(s)\n");      You, seconds ago • Uncommitted changes
    }
}
```

pacote `grafo`

classe `Grafo`

grafo > Grafo.java > ...

```
You, 2 days ago | 1 author (You)
/*
 Disponibilizado em Projeto de Algoritmos com implementações em Java e C++
 Autor: Nivio Ziviani
*/
      You, 2 days ago • FIX: resolvendo os conflitos do pacote grafo
You, 5 days ago
package grafo;

You, 2 days ago | 1 author (You)
public class Grafo {

    You, 5 days ago | 1 author (You)
    public static class Aresta {
        private int v1, v2, peso;

        public Aresta(int v1, int v2, int peso) {
            this.v1 = v1;
            this.v2 = v2;
            this.peso = peso;
        }

        public int peso() {
            return this.peso;
        }

        public int v1() {
            return this.v1;
        }

        public int v2() {
            return this.v2;
        }
    }

    private int mat[][];                                    // pesos do tipo inteiro
    private int numVertices;
    private int pos[];                                      // posicao atual ao se percorrer os adjs de um vertice v
```

grafo > Grafo.java > ...

```
    public Grafo(int numVertices) {
        this.mat = new int[numVertices][numVertices];
        this.pos = new int[numVertices];
        this.numVertices = numVertices;
        for (int i = 0; i < this.numVertices; i++) {
            for (int j = 0; j < this.numVertices; j++)
                this.mat[i][j] = 0;
            this.pos[i] = -1;
        }
    }

    public void insereAresta(int v1, int v2, int peso) {
        this.mat[v1][v2] = peso;
    }
```

grafo > Grafo.java > ...

```java
    public void insereArestaBidirecionada(int v1, int v2, int peso) {
        this.mat[v1][v2] = peso;
        this.mat[v2][v1] = peso;
    }

    public boolean existeAresta(int v1, int v2) {
        return (this.mat[v1][v2] > 0);
    }

    public Aresta primeiroListaAdj(int v) {
        this.pos[v] = -1;
        return this.proxAdj(v);
    }

    public boolean listaAdjVazia(int v) {
        for (int i = 0; i < this.numVertices; i++) {
            if (this.mat[v][i] > 0)
                return false;
        }
        return true;
    }

    public Aresta proxAdj(int v) {
        this.pos[v]++;
        while ((this.pos[v] < this.numVertices) && (this.mat[v][this.pos[v]] == 0))
            this.pos[v]++;
        if (this.pos[v] == this.numVertices)
            return null;
        else
            return new Aresta(v, this.pos[v], this.mat[v][this.pos[v]]);
    }
```

grafo > Grafo.java > ...

```java

    // Encontrar a menor aresta na lista de adjacencias
    public Aresta menorListaAdjacencia(int v, boolean[] visitados) {
        int aux, menor = Integer.MAX_VALUE;
        int i, menorI=0;
        for(i=0; i<this.numVertices; i++)  {
                aux = this.mat[v][i];
                if((aux<menor && v!=i) && !visitados[i]){
                    menor = aux;
                    menorI = i;
                }
        }
        Aresta Menor = new Aresta(v, menorI, menor);
        return Menor;
    }
```

grafo > Grafo.java > ...

```java
    public Aresta retiraAresta(int v1, int v2) {
        if (this.mat[v1][v2] == 0)
            return null;                                        //Aresta nao existe
        else {
            Aresta aresta = new Aresta(v1, v2, this.mat[v1][v2]);
            this.mat[v1][v2] = 0;
            return aresta;
        }
    }

    public void imprime() {
        System.out.print("   ");
        for (int i = 0; i < this.numVertices; i++)
            System.out.print(i + "   ");

        System.out.println();
        for (int i = 0; i < this.numVertices; i++) {
            System.out.print(i + "  ");
            for (int j = 0; j < this.numVertices; j++)
                System.out.print(this.mat[i][j] + "   ");
            System.out.println();
        }
    }

    public int numVertices() {
        return this.numVertices;
    }

    public int getPeso(int v1, int v2){
        return this.mat[v1][v2];
    }
```

grafo > Grafo.java > ...

```java
    public Grafo grafoTransposto() {
        Grafo grafoT = new Grafo(this.numVertices);
        for (int v = 0; v < this.numVertices; v++) {
            if (!this.listaAdjVazia(v)) {
                Aresta adj = this.primeiroListaAdj(v);
                while (adj != null) {
                    grafoT.insereAresta(adj.v2(), adj.v1(), adj.peso());
                    adj = this.proxAdj(v);
                }
            }
        }
        return grafoT;
    }
}
```

pacote `utils`

classe `ReadTSP`

utils > ReadTSP.java > ReadTSP > geraGrafo(String)

You, 4 days ago | 1 author (You)

```java
package utils;

import grafo.Grafo;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

You, 4 days ago | 1 author (You)
public class ReadTSP {

    private Scanner leitor;
    private int [][] matriz;
    private Grafo grafo;
    public int tamanho;

    public ReadTSP(String file) throws FileNotFoundException, IOException {
        this.leitor = new Scanner(new File("./files/" + file + ".tsp"));            // abrindo o arquivo

        // lendo algumas informacoes do arquivo .tsp
        while(!leitor.next().equals("DIMENSION:"));
        tamanho = Integer.parseInt(leitor.next());

        while (!leitor.next().equals("EDGE_WEIGHT_SECTION"));

        matriz = geraMatriz(file);
        grafo = geraGrafo(file);
    }
```

utils > ReadTSP.java > ReadTSP > geraGrafo(String)

```java
    public int [][] geraMatriz(String file) {
        int [][] matriz = new int [tamanho][tamanho];
        int linha = -1;
        int coluna = -1;

        if(file.equals("pa561")) {                                          // arquivo que a diagonal e inferior
            linha = 0;
            coluna = 0;
            while(leitor.hasNext()) {
                String str = leitor.next();
                if (str.equals("DISPLAY_DATA_SECTION"))
                    break;
                if (str.equals("0")) {
                    matriz[linha][coluna] = Integer.parseInt(str);
                    linha++;
                    coluna=0;
                }
                else {
                    matriz[linha][coluna] = Integer.parseInt(str);
                    coluna++;
                }
            }
        }
```

utils > ReadTSP.java > ReadTSP > geraGrafo(String)

```
52          else {                                                  // arquivo que a diagonal e superior
53              while(leitor.hasNext()) {
54                  String str = leitor.next();
55                  if (str.equals("EOF"))
56                      break;
57                  if (str.equals("0")) {
58                      linha++;
59                      coluna = linha;
60                      matriz[linha][coluna] = Integer.parseInt(str);
61                  }
62                  else {
63                      coluna++;
64                      matriz[linha][coluna] = Integer.parseInt(str);
65                  }
66              }
67          }
68          return matriz;
69      }
70
71      public Grafo geraGrafo(String file) {                       // gerar o grafo a partir do arquivo
72          Grafo grafo = new Grafo(tamanho);
73          if (file.equals("pa561")) {                             // arquivo que a diagonal e inferior
74              for (int i = tamanho-1; i >=0; i--) {
75                  for (int j = 0; j < i; j++)
76                      grafo.insereArestaBidirecionada(i, j, matriz[i][j]);
77              }
78          }
79          else {                                                  // arquivo que a diagonal e superior
80              for (int i = 0; i < tamanho; i++) {
81                  for (int j = i; j < tamanho; j++)
82                      grafo.insereArestaBidirecionada(i, j, matriz[i][j]);
83              }
84          }
85          return grafo;
86      }
87
88      public Grafo getGrafo() {
89          return grafo;
90      }
91  }
```

pacote `files`

Contém os arquivos para testes da heurística aplicada.

3. Casos de teste

a. Força Bruta

Para executar o algoritmo desenvolvido, temos os seguintes casos de testes que serão executados para o algoritmo de Força Bruta:

```
Numero de vertices: 2
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1
0  0   8
1  8   0

Ponto de partida:
0

Menor caminho: 0 1 0
Distancia: 16

Tempo de execução: 0,00666 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 3
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2
0  0   5   8
1  5   0   4
2  8   4   0

Ponto de partida:
0

Menor caminho: 0 1 2 0
Distancia: 17

Tempo de execução: 0,00703 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 4
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3
0  0   7   3   1
1  7   0   5   6
2  3   5   0   1
3  1   6   1   0

Ponto de partida:
0

Menor caminho: 0 1 2 3 0
Distancia: 14

Tempo de execução: 0,00723 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 5
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4
0  0   7   7   3   2
1  7   0   3   7   1
2  7   3   0   7   4
3  3   7   7   0   3
4  2   1   4   3   0

Ponto de partida:
0

Menor caminho: 0 3 2 1 4 0
Distancia: 16

Tempo de execução: 0,00780 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 6
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5
0  0   7   2   8   3   3
1  7   0   8   2   1   2
2  2   8   0   4   8   1
3  8   2   4   0   3   6
4  3   1   8   3   0   2
5  3   2   1   6   2   0

Ponto de partida:
0

Menor caminho: 0 2 5 1 3 4 0
Distancia: 13

Tempo de execução: 0,00926 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 7
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6
0  0   6   4   4   1   6   5
1  6   0   4   8   2   6   3
2  4   4   0   1   6   6   3
3  4   8   1   0   3   2   4
4  1   2   6   3   0   4   4
5  6   6   6   2   4   0   4
6  5   3   3   4   4   4   0

Ponto de partida:
0

Menor caminho: 0 2 3 5 6 1 4 0
Distancia: 17

Tempo de execução: 0,01115 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 8
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7
0  0   4   5   6   1   5   1   2
1  4   0   8   6   7   2   5   5
2  5   8   0   1   8   5   1   8
3  6   6   1   0   2   3   6   6
4  1   7   8   2   0   4   4   5
5  5   2   5   3   4   0   1   5
6  1   5   1   6   4   1   0   1
7  2   5   8   6   5   5   1   0

Ponto de partida:
0

Menor caminho: 0 4 3 2 6 5 1 7 0
Distancia: 15

Tempo de execução: 0,01854 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 9
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7   8
0  0   8   4   3   2   4   2   2   8
1  8   0   3   5   6   4   7   5   4
2  4   3   0   7   2   7   4   3   8
3  3   5   7   0   7   8   1   6   5
4  2   6   2   7   0   7   5   2   1
5  4   4   7   8   7   0   2   8   8
6  2   7   4   1   5   2   0   3   6
7  2   5   3   6   2   8   3   0   4
8  8   4   8   5   1   8   6   4   0

Ponto de partida:
0

Menor caminho: 0 3 6 5 1 8 4 2 7 0
Distancia: 22

Tempo de execução: 0,06088 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 10
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7   8   9
0  0   2   4   8   8   2   7   3   2   8
1  2   0   8   7   7   1   3   8   3   4
2  4   8   0   6   4   5   2   3   5   1
3  8   7   6   0   8   4   3   5   3   4
4  8   7   4   8   0   5   1   6   1   7
5  2   1   5   4   5   0   2   7   5   4
6  7   3   2   3   1   2   0   3   7   5
7  3   8   3   5   6   7   3   0   4   2
8  2   3   5   3   1   5   7   4   0   6
9  8   4   1   4   7   4   5   2   6   0

Ponto de partida:
0

Menor caminho: 0 1 5 3 8 4 6 2 9 7 0
Distancia: 20

Tempo de execução: 0,38524 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 11
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10
0  0   1   5   5   8   4   2   2   1   6   6
1  1   0   7   1   8   3   1   8   4   6   2
2  5   7   0   6   3   2   1   6   1   1   3
3  5   1   6   0   1   1   1   3   1   6   3
4  8   8   3   1   0   8   1   8   3   3   8
5  4   3   2   1   8   0   4   2   8   2   7
6  2   1   1   1   1   4   0   3   2   1   2
7  2   8   6   3   8   2   3   0   2   4   2
8  1   4   1   1   3   8   2   2   0   4   6
9  6   6   1   6   3   2   1   4   4   0   3
10  6   2   3   3   8   7   2   2   6   3   0

Ponto de partida:
0

Menor caminho: 0 1 10 7 5 3 4 6 9 2 8 0
Distancia: 14

Tempo de execução: 2,46268 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 12
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10   11
0  0   8   4   2   7   3   8   2   7   2   5   5
1  8   0   7   7   8   5   2   4   5   8   1   7
2  4   7   0   2   8   8   8   4   1   6   3   3
3  2   7   2   0   5   8   7   5   2   2   7   5
4  7   8   8   5   0   4   2   6   2   3   7   6
5  3   5   8   8   4   0   5   8   3   3   8   4
6  8   2   8   7   2   5   0   7   4   7   4   5
7  2   4   4   5   6   8   7   0   8   6   4   7
8  7   5   1   2   2   3   4   8   0   4   6   2
9  2   8   6   2   3   3   7   6   4   0   5   6
10  5   1   3   7   7   8   4   4   6   5   0   3
11  5   7   3   5   6   4   5   7   2   6   3   0

Ponto de partida:
0

Menor caminho: 0 7 10 1 6 4 5 11 8 2 3 9 0
Distancia: 28

Tempo de execução: 27,22451 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 13
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10   11   12
0  0   1   6   2   3   6   3   5   7   6   4   5   5
1  1   0   4   2   1   3   4   7   7   2   1   3   7
2  6   4   0   4   2   7   3   8   8   6   1   7   8
3  2   2   4   0   4   1   6   5   5   4   6   1   2
4  3   1   2   4   0   1   7   8   4   8   6   5   4
5  6   3   7   1   1   0   4   6   2   8   3   4   8
6  3   4   3   6   7   4   0   2   8   7   3   5   3
7  5   7   8   5   8   6   2   0   3   8   5   1   8
8  7   7   8   5   4   2   8   3   0   1   4   1   3
9  6   2   6   4   8   8   7   8   1   0   1   5   1
10  4   1   1   6   6   3   3   5   4   1   0   7   2
11  5   3   7   1   5   4   5   1   1   5   7   0   7
12  5   7   8   2   4   8   3   8   3   1   2   7   0

Ponto de partida:
0

Menor caminho: 0 1 10 2 4 5 3 12 9 8 11 7 6 0
Distancia: 18

Tempo de execução: 329,92090 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

```
Numero de vertices: 14
Distancia gerada aleatoriamente do menor para o maior.
Entre: 1
ate: 9
Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10   11   12   13
0  0   1   4   7   4   2   5   1   8   5   6   1   3   1
1  1   0   4   1   6   5   3   6   4   7   4   5   3   6
2  4   4   0   3   4   8   5   7   1   8   8   7   6   8
3  7   1   3   0   6   2   3   1   7   6   6   8   1   4
4  4   6   4   6   0   3   6   8   5   8   3   2   4   2
5  2   5   8   2   3   0   1   1   6   1   3   6   3   3
6  5   3   5   3   6   1   0   3   8   4   4   1   3   5
7  1   6   7   1   8   1   3   0   6   6   2   3   3   8
8  8   4   1   7   5   6   8   6   0   5   3   5   5   8
9  5   7   8   6   8   1   4   6   5   0   4   7   2   1
10  6   4   8   6   3   3   4   2   3   4   0   1   7   7
11  1   5   7   8   2   6   1   3   5   7   1   0   2   7
12  3   3   6   1   4   3   3   3   5   2   7   2   0   5
13  1   6   8   4   2   3   5   8   8   1   7   7   5   0

Ponto de partida:
0

Menor caminho: 0 1 3 12 9 13 4 2 8 10 11 6 5 7 0
Distancia: 21

Tempo de execução: 4782,32253 segundo(s)

PS C:\Users\erick\OneDrive\Documentos\GitHub\Repositorios\Algoritmo-e-Estrutura-de-Dados-II\AEDS II\CaixeiroViajante\CaixeiroViajante>
```

b. Heurística Gulosa

Para executar o algoritmo desenvolvido, disponibilizamos os casos de testes para executar o algoritmo da Heurística Gulosa em um link, pois se trata de arquivos grandes em formato *.tsp*:

- pa561.tsp, o problema possui 561 cidades e as distâncias estão disponíveis em forma de matriz de adjacência, mas somente a diagonal inferior desta matriz.
- si535.tsp, o problema possui 535 cidades e as distâncias estão disponíveis em forma de matriz de adjacência, mas somente a diagonal superior desta matriz.
- si1032.tsp, o problema possui 1032 cidades e as distâncias estão disponíveis em forma de matriz de adjacência, mas somente a diagonal superior desta matriz.

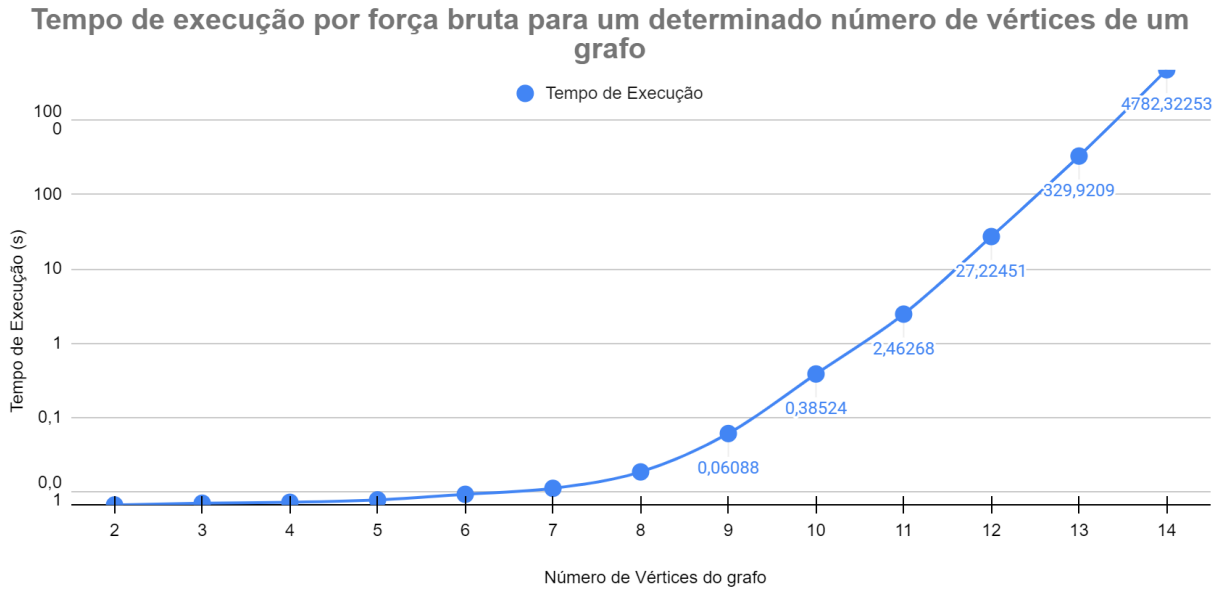
4. Resultados

a. Força Bruta

Como observamos pela seção 3.a, temos que os resultados obtidos são:

Número de Vértices	Distância		Ponto de partida	Distancia encontrada	Tempo de execução (s)
	Entre	Até			
2	1	9	0	16	0,00666
3	1	9	0	17	0,00703
4	1	9	0	14	0,00723
5	1	9	0	16	0,00780
6	1	9	0	13	0,00926
7	1	9	0	17	0,01115
8	1	9	0	15	0,01854
9	1	9	0	22	0,06088
10	1	9	0	20	0,38524
11	1	9	0	14	2,46268
12	1	9	0	28	27,22451
13	1	9	0	18	329,9209
14	1	9	0	21	4782,32253

Em termos gráficos, temos:



Sendo assim, observamos que o tempo necessário para resolver o problema do caixeiro viajante, quando temos um crescimento do tamanho do problema, utilizando-se a força bruta, temos um crescimento exponencial do tempo necessário.

b. Heurística Gulosa

Aplicando os arquivos testes da seção 3.b, temos que os resultados obtidos são:

Arquivo	Distância calculada pela Heurística Gulosa
pa561.tsp	3422
si535.tsp	50144
si1032.tsp	94571

5. Considerações Finais

Ao fim da realização deste presente trabalho, conseguimos compreender não apenas de modo teórico, mas também prático os conceitos da força bruta e da heurística do algoritmo guloso. Com os dados apresentados, percebemos claramente que o método da heurística gulosa é extremamente superior ao método de força bruta, utilizando-se o parâmetro de tempo de execução e o número de comparações realizadas. Além disso, concluímos que os casos de testes foram executados com sucesso.