Centro Federal de Educação Tecnológica de Minas Gerais

Engenharia de Computação

Laboratório de Algoritmos e Estruturas de Dados II

TRABALHO PRÁTICO 2

Algoritmo de força bruta e uma heurística para o problema do Caixeiro Viajante

Thamiris Souza Madeira Ferreira – 20193008037

Hugo Barbosa Santana Silva – 20193002946

Belo Horizonte

July 12, 2022

1 Contextualização

O problema do Caixeiro Viajante consiste em, dado um conjunto de cidades onde existe um caminho entre cada par de cidade com uma distância positiva, encontrar um caminho que, a partir de uma cidade, visita-se todas as cidades e retorna à cidade inicial percorrendo a menor distância possível.

Na parte 1 do trabalho, foi implementado o método de força bruta para solucionar o problema, ou seja, um algoritmo que determina todas as possíveis rotas e escolhe o melhor, ou seja, a menor; geradas instâncias de 2 à 13, com valores aleatórios, e aplicado o método de força bruta e computado o tempo de execução durante a aplicação em cada uma das instâncias geradas.

Na parte 2, foi implementada uma heurística gulosa para encontrar uma solução para o problema do caixeiro viajante. Aplicamos o método implementado em três instâncias do problema disponíveis no moodle e verificada a distância calculada pela heurística:

- **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;
- **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;
- **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;

2 Código Fonte

Nosso código é composto das classes principais *Grafo*, *ForcaBruta* e *HeuristicaGulosa*, e suas respectivas classes de teste.

Também foi implementada uma classe para ler os arquivos de teste que foram disponibilizados no moodle (*ReadTSP*).

2.1 Classe *Grafo*

```
1  public class Grafo {
2
3      public static class Aresta {
4          private int v1, v2, peso;
5
6          public Aresta(int v1, int v2, int peso) {
7              this.v1 = v1;
8              this.v2 = v2;
9              this.peso = peso;
10         }
11
12         public int peso() {
13             return this.peso;
14         }
15
16         public int v1() {
17             return this.v1;
18         }
19
20         public int v2() {
21             return this.v2;
22         }
23     }
24
25     private int mat[][]; // pesos do tipo inteiro
26     private int numVertices;
27     private int pos[]; // posicao atual ao se percorrer os
           adjs de um vertice v
28
29     public Grafo(int numVertices) {
30         this.mat = new int[numVertices][numVertices];
31         this.pos = new int[numVertices];
32         this.numVertices = numVertices;
33         for (int i = 0; i < this.numVertices; i++) {
34             for (int j = 0; j < this.numVertices; j++)
35                 this.mat[i][j] = 0;
36             this.pos[i] = -1;
37         }
38     }
```

```
39
40      public void insereAresta(int v1, int v2, int peso) {
41          this.mat[v1][v2] = peso;
42      }
43
44      public void insereArestaBidirecionada(int v1, int v2, int
           peso) {
45          this.mat[v1][v2] = peso;
46          this.mat[v2][v1] = peso;
47      }
48
49      public boolean existeAresta(int v1, int v2) {
50          return (this.mat[v1][v2] > 0);
51      }
52
53      public Aresta primeiroListaAdj(int v) {
54          this.pos[v] = -1;
55          return this.proxAdj(v);
56      }
57
58      public boolean listaAdjVazia(int v) {
59          for (int i = 0; i < this.numVertices; i++) {
60              if (this.mat[v][i] > 0)
61                  return false;
62          }
63          return true;
64      }
65
66      public Aresta proxAdj(int v) {
67          this.pos[v]++;
68          while ((this.pos[v] < this.numVertices) && (this.mat[v
               ][this.pos[v]] == 0))
69              this.pos[v]++;
70          if (this.pos[v] == this.numVertices)
71              return null;
72          else
73              return new Aresta(v, this.pos[v], this.mat[v][this
                   .pos[v]]);
74      }
75
76      // Encontrar a menor aresta na lista de adjacencias
```

```
77      public Aresta menorListaAdjacencia(int v, boolean[]
           visitados) {
78          int aux, menor = Integer.MAX_VALUE;
79          int i, menorI = 0;
80          for (i = 0; i < this.numVertices; i++) {
81              aux = this.mat[v][i];
82              if ((aux < menor && v != i) && !visitados[i]) {
83                  menor = aux;
84                  menorI = i;
85              }
86          }
87          Aresta Menor = new Aresta(v, menorI, menor);
88          return Menor;
89      }
90
91      public Aresta retiraAresta(int v1, int v2) {
92          if (this.mat[v1][v2] == 0)
93              return null; // Aresta nao existe
94          else {
95              Aresta aresta = new Aresta(v1, v2, this.mat[v1][v2
                   ]);
96              this.mat[v1][v2] = 0;
97              return aresta;
98          }
99      }
100
101     public void imprime() {
102         System.out.print("    ");
103         for (int i = 0; i < this.numVertices; i++)
104             System.out.print(i + "    ");
105
106         System.out.println();
107         for (int i = 0; i < this.numVertices; i++) {
108             System.out.print(i + "   ");
109             for (int j = 0; j < this.numVertices; j++)
110                 System.out.print(this.mat[i][j] + "    ");
111             System.out.println();
112         }
113     }
114
115     public int numVertices() {
```

```
116         return this.numVertices;
117     }
118 
119     public int getPeso(int v1, int v2) {
120         return this.mat[v1][v2];
121     }
122 
123     public Grafo grafoTransposto() {
124         Grafo grafoT = new Grafo(this.numVertices);
125         for (int v = 0; v < this.numVertices; v++) {
126             if (!this.listaAdjVazia(v)) {
127                 Aresta adj = this.primeiroListaAdj(v);
128                 while (adj != null) {
129                     grafoT.insereAresta(adj.v2(), adj.v1(),
                           adj.peso());
130                     adj = this.proxAdj(v);
131                 }
132             }
133         }
134         return grafoT;
135     }
136 }
```

2.2 Classe *ForcaBruta*

```
1  public class ForcaBruta {
2      private Grafo grafo;
3      private HashMap<Integer, ArrayList<Integer>> mapa; //
           relaciona o vertice que visitou com a distancia total
           que percorreu para visita-lo
4      private int menorDistancia = 0;
5  
6      public ForcaBruta(Grafo grafo) {
7          this.grafo = grafo;
8          this.mapa = new HashMap<>();
9      }
10 
11     public void backtracking(int pointA) {
12         ArrayList<Integer> visited = new ArrayList<>();
```

```
13          ArrayList<Aresta> arestasAdj = this.arestasAdj(pointA)
                ;
14
15          // caminha de A ate o vertice
16          for (int i = 0; i < arestasAdj.size(); i++) {
17              int pointC = arestasAdj.get(i).v2();
18              visita(visited, pointA, pointA, pointC);
19          }
20      }
21
22      public ArrayList<Aresta> arestasAdj(int v) {
23          ArrayList<Aresta> arestas = new ArrayList<>();
24          for (int i = 0; i < this.grafo.numVertices(); i++)
25              if (this.grafo.getPeso(v, i) > 0)
26                  arestas.add(new Aresta(v, i, this.grafo.
                        getPeso(v, i)));
27          return arestas;
28      }
29
30      private void visita(final ArrayList<Integer> visited, int
            pontoInicial, int pointA, int pointB) {
31          int totalDistance;
32          ArrayList<Aresta> arestasAdj = this.arestasAdj(pointA)
                ;
33          ArrayList visitedCopy = new ArrayList(visited);
34          visitedCopy.add(pointA);
35
36          // salta o vertice visitado anteriormente e pula-o
37          for (int i = 0; i < arestasAdj.size(); i++) {
38              int pointC = arestasAdj.get(i).v2(); // vertice
                    atual -> pointC
39              if (!visitedCopy.contains(pointC)) {
40                  if (visitedCopy.size() == this.grafo.
                        numVertices() - 1) {
41                      visitedCopy.add(pointB);
42                      visitedCopy.add(pontoInicial);
43                      totalDistance = 0;
44                      for (int j = 0; j < visitedCopy.size(); j
                            ++) {
45                          if (j < visitedCopy.size() - 1)
46                              totalDistance += this.grafo.
```

```
                                    getPeso(
47                                          (int) visitedCopy.get(j),
48                                          (int) visitedCopy.get(j +
                                               1));
49                          }
50                          final ArrayList visitedCopy2 = new
                               ArrayList(visitedCopy);
51                          this.mapa.put(totalDistance, visitedCopy2)
                               ;
52                          visitedCopy.remove(visitedCopy.size() - 1)
                               ;
53                          visitedCopy.remove(visitedCopy.size() - 1)
                               ;
54                      } else if (pointC != pointB) {
55                          visita(visitedCopy, pontoInicial, pointC,
                               pointB);
56                      }
57                  }
58              }
59          }
60
61          public void solucaoOtima() {
62              this.menorDistancia = this.mapa.keySet().stream().
                   findFirst().get();
63
64              for (Integer distance : this.mapa.keySet()) // menor
                   distancia
65                  if (distance < this.menorDistancia)
66                      this.menorDistancia = distance;
67              System.out.print("\nMenor caminho: ");
68
69              for (int j = 0; j < this.mapa.get(this.menorDistancia)
                   .size(); j++)
70                  System.out.print(this.mapa.get(this.menorDistancia
                       ).get(j) + " ");
71              System.out.println("\nDistancia: " + this.
                   menorDistancia);
72          }
73      }
```

2.3 Heurística Gulosa

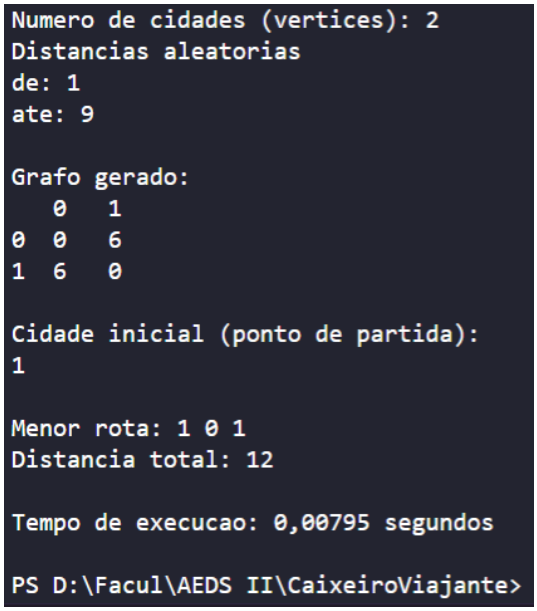
```
1  public class HeuristicaGulosa {
2      private Grafo grafo;
3      private boolean[] visitas; // armazena as visitas a um dos
           vertices do grafo
4      private int[][] melhorCaminho;
5      private int menorDistancia = 0;
6
7      public HeuristicaGulosa(Grafo grafo) {
8          int numVertices = grafo.numVertices();
9
10         this.grafo = grafo;
11         this.visitas = new boolean[numVertices];
12         this.melhorCaminho = new int[grafo.numVertices()][3];
13
14         for (int i = 0; i < numVertices; i++) {
15             this.visitas[i] = false;
16         }
17     }
18
19     public boolean visitouTodos(boolean[] visitas) {
20         boolean result = true;
21         for (boolean vertice : visitas) {
22             if (!vertice) {
23                 result = false;
24             }
25         }
26         return result;
27     }
28
29     public int[][] encontraCaminho() {
30         int vi = 0, verticeX, pesoV, i = 0;
31         visitas[0] = true;
32         for (; visitouTodos(visitas) == false; i++) {
33             verticeX = grafo.menorListaAdjacencia(vi, visitas)
                   .v2();
34             pesoV = grafo.menorListaAdjacencia(vi, visitas).
                   peso();
35
36             melhorCaminho[i][0] = vi;
```

```
37              melhorCaminho[i][1] = verticeX;
38              melhorCaminho[i][2] = pesoV;
39
40              menorDistancia += pesoV;
41              vi = verticeX;
42              visitas[verticeX] = true;
43          }
44          visitas[0] = false;
45
46          verticeX = grafo.menorListaAdjacencia(vi, visitas).v2
                ();
47          pesoV = grafo.menorListaAdjacencia(vi, visitas).peso()
                ;
48
49          melhorCaminho[i][0] = vi;
50          melhorCaminho[i][1] = verticeX;
51          melhorCaminho[i][2] = pesoV;
52
53          menorDistancia += pesoV;
54
55          return melhorCaminho;
56      }
57
58      public int getPesoTotal() {
59          return menorDistancia;
60      }
61  }
```

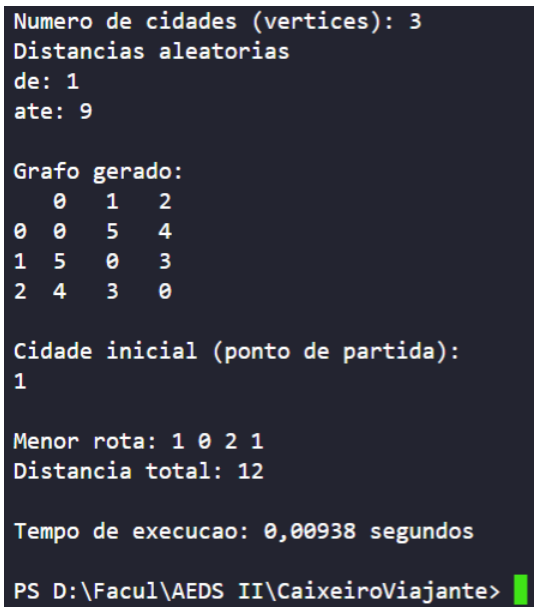
3 Algoritmos

3.1 Força Bruta

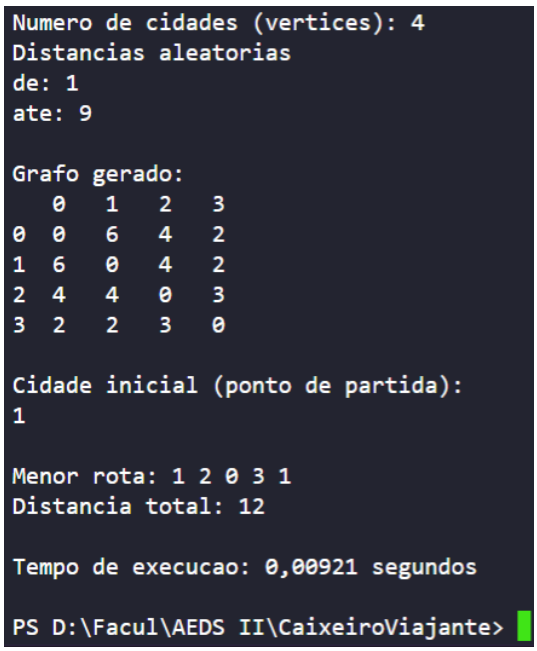
Executamos o algoritmo de força bruta com instâncias de 2 á 13 e utilizamos os seguintes casos teste:



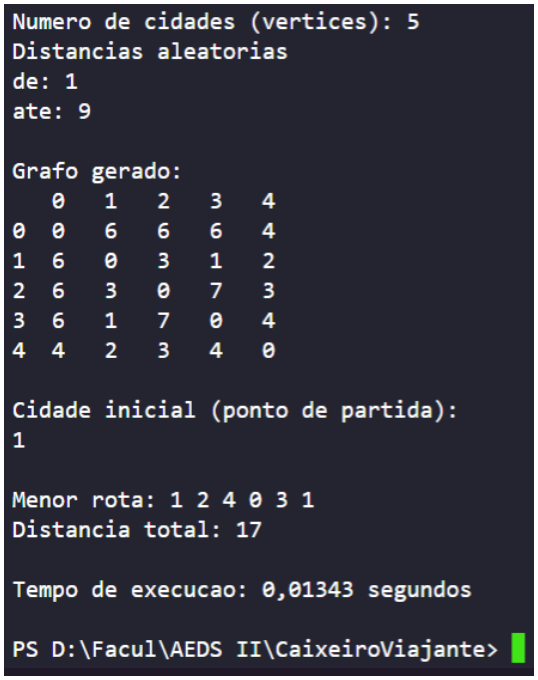
(a) n = 2



(b) n = 3



(c) n = 4



(d) n = 5

```
Numero de cidades (vertices): 6
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5
0  0   1   6   5   2   5
1  1   0   8   1   1   5
2  6   8   0   3   3   2
3  5   1   3   0   8   7
4  2   1   3   8   0   1
5  5   5   2   7   1   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 0 4 5 2 3 1
Distancia total: 10

Tempo de execucao: 0,01158 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(e) n = 6

```
Numero de cidades (vertices): 7
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6
0  0   8   6   3   6   1   8
1  8   0   3   5   7   2   7
2  6   3   0   1   8   2   1
3  3   5   1   0   4   6   1
4  6   7   8   4   0   8   4
5  1   2   2   6   8   0   1
6  8   7   1   1   4   1   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 2 6 4 3 0 5 1
Distancia total: 18

Tempo de execucao: 0,01555 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(f) n = 7

```
Numero de cidades (vertices): 8
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6   7
0  0   5   5   1   8   8   4   2
1  5   0   2   6   1   1   2   2
2  5   2   0   1   6   2   2   7
3  1   6   1   0   3   1   5   3
4  8   1   6   3   0   6   6   4
5  8   1   2   1   6   0   2   4
6  4   2   2   5   6   2   0   3
7  2   2   7   3   4   4   3   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 4 7 0 3 2 6 5 1
Distancia total: 14

Tempo de execucao: 0,02731 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(g) n = 8

```
Numero de cidades (vertices): 9
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6   7   8
0  0   5   6   3   5   7   3   2   4
1  5   0   5   2   8   5   6   4   3
2  6   5   0   6   8   4   2   1   8
3  3   2   6   0   1   4   8   6   2
4  5   8   8   1   0   2   1   8   4
5  7   5   4   4   2   0   6   2   2
6  3   6   2   8   1   6   0   7   8
7  2   4   1   6   8   2   7   0   8
8  4   3   8   2   4   2   8   8   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 3 0 7 2 6 4 5 8 1
Distancia total: 18

Tempo de execucao: 0,07474 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(h) n = 9

```
Numero de cidades (vertices): 10
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6   7   8   9
0  0   5   8   7   1   8   8   5   6   6
1  5   0   1   1   8   6   3   5   8   4
2  8   1   0   3   6   2   1   2   8   2
3  7   1   3   0   2   5   7   2   3   4
4  1   8   6   2   0   5   5   4   3   7
5  8   6   2   5   5   0   6   3   6   3
6  8   3   1   7   5   6   0   4   7   8
7  5   5   2   2   4   3   4   0   3   8
8  6   8   8   3   3   6   7   3   0   1
9  6   4   2   4   7   3   8   8   1   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 3 7 0 4 8 9 5 2 6 1
Distancia total: 22

Tempo de execucao: 0,48654 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(i) n = 10

```
Numero de cidades (vertices): 11
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10
0  0   8   8   5   1   3   5   6   5   4   6
1  8   0   4   7   8   1   2   4   5   2   4
2  8   4   0   4   3   5   8   5   5   6   2
3  5   7   4   0   5   7   2   2   4   8   3
4  1   8   3   5   0   7   4   1   1   1   3
5  3   1   5   7   7   0   4   1   2   8   3
6  5   2   8   2   4   4   0   1   1   5   4
7  6   4   5   2   1   1   1   0   1   5   5
8  5   5   5   4   1   2   1   1   0   1   6
9  4   2   6   8   1   8   5   5   1   0   6
10  6   4   2   3   3   3   4   5   6   6   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 5 0 4 2 10 3 7 6 8 9 1
Distancia total: 20

Tempo de execucao: 3,02585 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(j) n = 11

```
Numero de cidades (vertices): 12
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10   11
0  0   4   1   4   2   3   2   3   4   6   4   3
1  4   0   7   6   8   3   1   7   6   6   4   4
2  1   7   0   5   3   8   1   5   7   2   8   7
3  4   6   5   0   3   2   6   5   5   7   6   3
4  2   8   3   3   0   5   6   8   8   8   1   2
5  3   3   8   2   5   0   4   1   3   6   3   1
6  2   1   1   6   6   4   0   5   3   7   1   8
7  3   7   5   5   8   1   5   0   8   8   7   4
8  4   6   7   5   8   3   3   8   0   4   1   3
9  6   6   2   7   8   6   7   8   4   0   3   6
10  4   4   8   6   1   3   1   7   1   3   0   3
11  3   4   7   3   2   1   8   4   3   6   3   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 6 10 8 9 2 0 7 5 3 4 11 1
Distancia total: 25

Tempo de execucao: 32,20225 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(k) n = 12

```
Numero de cidades (vertices): 13
Distancias aleatorias
de: 1
ate: 9

Grafo gerado:
   0   1   2   3   4   5   6   7   8   9   10   11   12
0  0   6   8   3   2   1   2   2   4   3   1   3   5
1  6   0   7   8   7   3   1   3   6   3   8   1   4
2  8   7   0   8   1   6   5   1   8   7   7   4   5
3  3   8   8   0   4   6   6   7   4   5   1   4   6
4  2   7   1   4   0   7   3   2   2   5   5   7   8
5  1   3   6   6   7   0   7   4   5   8   3   7   2
6  2   1   5   6   3   7   0   1   4   4   7   8   1
7  2   3   1   7   2   4   1   0   8   2   7   3   7
8  4   6   8   4   2   5   4   8   0   6   4   3   2
9  3   3   7   5   5   8   4   2   6   0   1   5   1
10  1   8   7   1   5   3   7   7   4   1   0   8   6
11  3   1   4   4   7   7   8   3   3   5   8   0   5
12  5   4   5   6   8   2   1   7   2   1   6   5   0

Cidade inicial (ponto de partida):
1

Menor rota: 1 6 12 5 0 3 10 9 7 2 4 8 11 1
Distancia total: 20

Tempo de execucao: 399,96806 segundos

PS D:\Facul\AEDS II\CaixeiroViajante>
```

(l) n = 13

3.2 Heurística

As distâncias calculadas pela heurística gulosa foram:

```
Arquivo: pa561.tsp
Distancia total: 3422

Arquivo: si535.tsp
Distancia total: 50144

Arquivo: si1032.tsp
Distancia total: 94571

PS D:\Facul\AEDS II\CaixeiroViajante>
```

Figure 1: Heurística Gulosa

4 Gráfico

De acordo com os dados obtidos dos casos teste, conseguimos montar o seguinte gráfico exponencial:

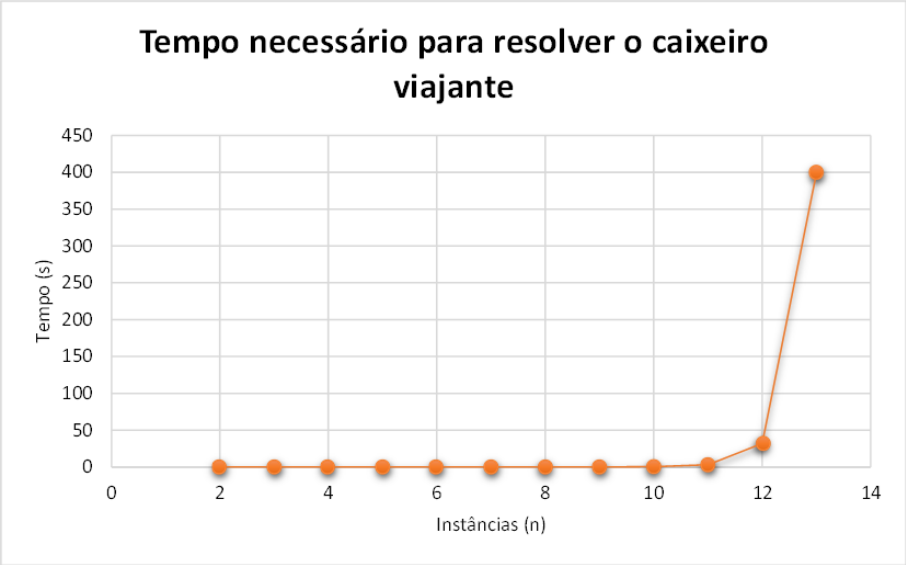


Figure 2: Gráfico exponencial

5 Considerações finais

Tentamos executar o algoritmo de força bruta de 2 á 14, porém com n = 13 o algoritmo já estava demorando muito para ser executado. Com n = 14 o tempo de espera estava muito alto e com isso resolvemos considerar apenas com n até 13.