Universidade de Pernambuco – UPE
Campus Garanhuns, PE
Disciplina de Inteligência Artificial
Adjailson Ferreira de Melo

Projeto I - Busca

A proposta de usar o algoritmo A* para encontrar a menor rota da cidade de Garanhuns-PE até Caruaru-PE. De acordo com o site do Google Maps a distância entre essas duas cidades é de 96,80 km. O tempo estimado do percurso da viagem entre as duas cidades é de aproximadamente 1 h 35 min. Já em linha reta entre essas duas cidades a distância é de 85,56 km, como mostra a Figura 1.

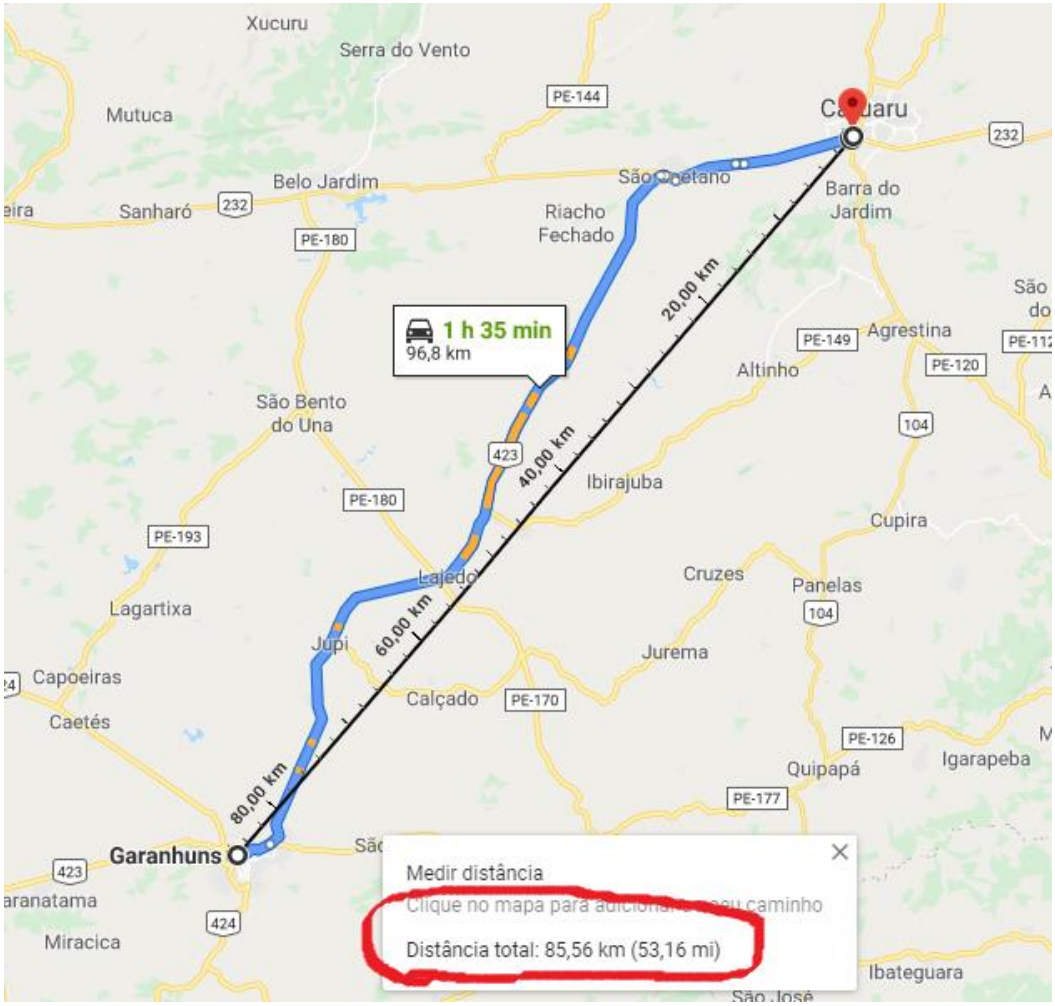

Figura 1: Rota entre Garanhuns e Caruaru.

Ao usar o algoritmo para encontrar a melhor rota para chegar entre uma cidade e outra, considerando o menor custo em Km e a condição do caminho (Figura 3). Para isso foi usado outras cidades com outras possíveis rotas para chegar ao mesmo destino, Figura 2.

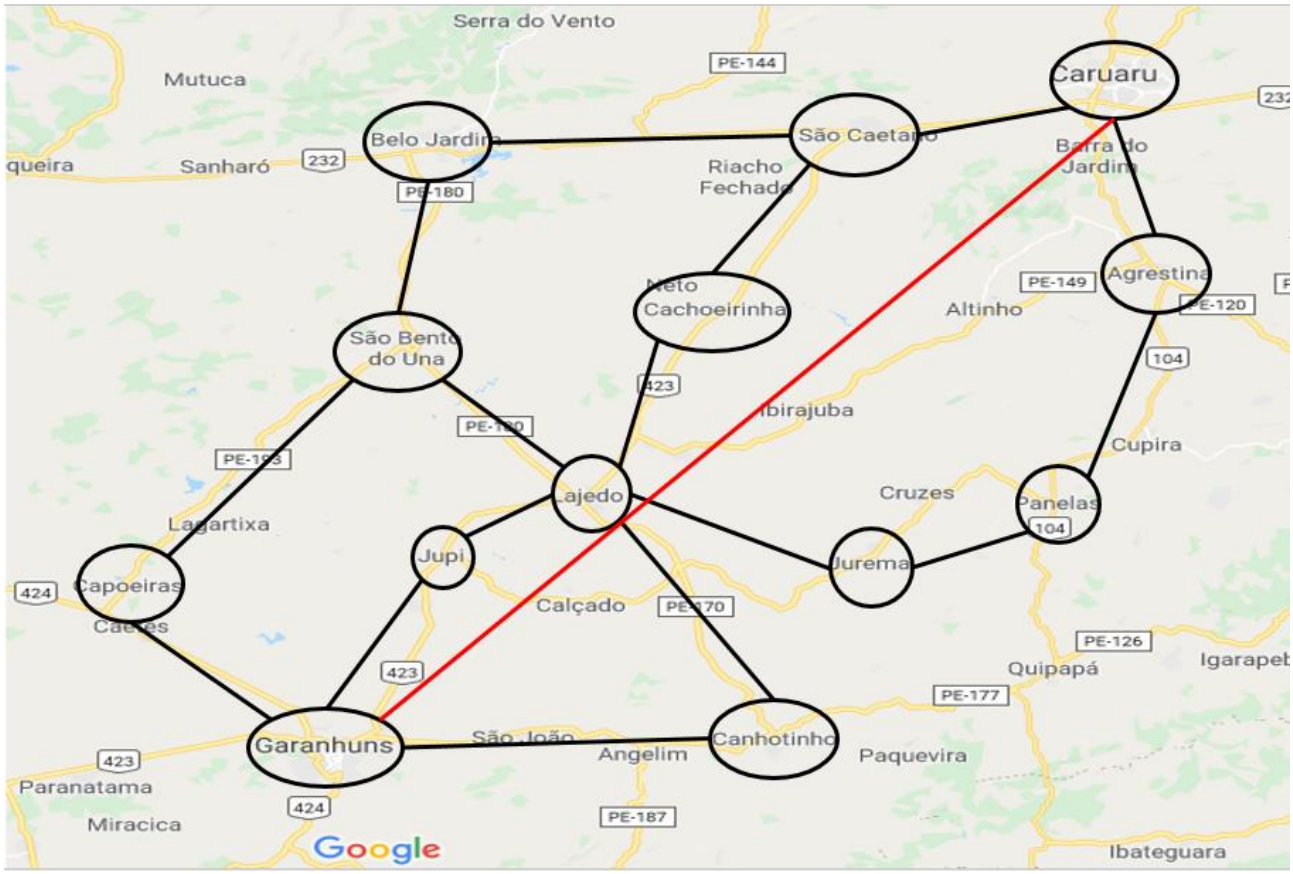

Figura 2: Algumas cidades vizinhas de Garanhuns a Caruaru.

Para encontrar os valores reais usei o Google Maps para calcular as distâncias entre os valores heurísticos usando a própria ferramenta do Google Maps, uma outra forma também seria usando uma escala sobre a imagem capturada e transformando nos valores verdadeiros, mas isso pode não ter tanta precisão.

Na Figura 3, temos todas as informações dos valores reais (cor preta) e heurístico (cor roxo) mais duas observações: a aresta de cor amarela (Capoeiras a São Bento do Una) leva mais tempo devido à grande quantidade de buracos, assim também para a aresta laranja (Lajedo a São Bento do Una) devido às 22 lombadas físicas existentes. O tempo da aresta amarela é de 25 e a laranja 20, esses valores são somados sobre os pesos reais de suas respectivas arestas.

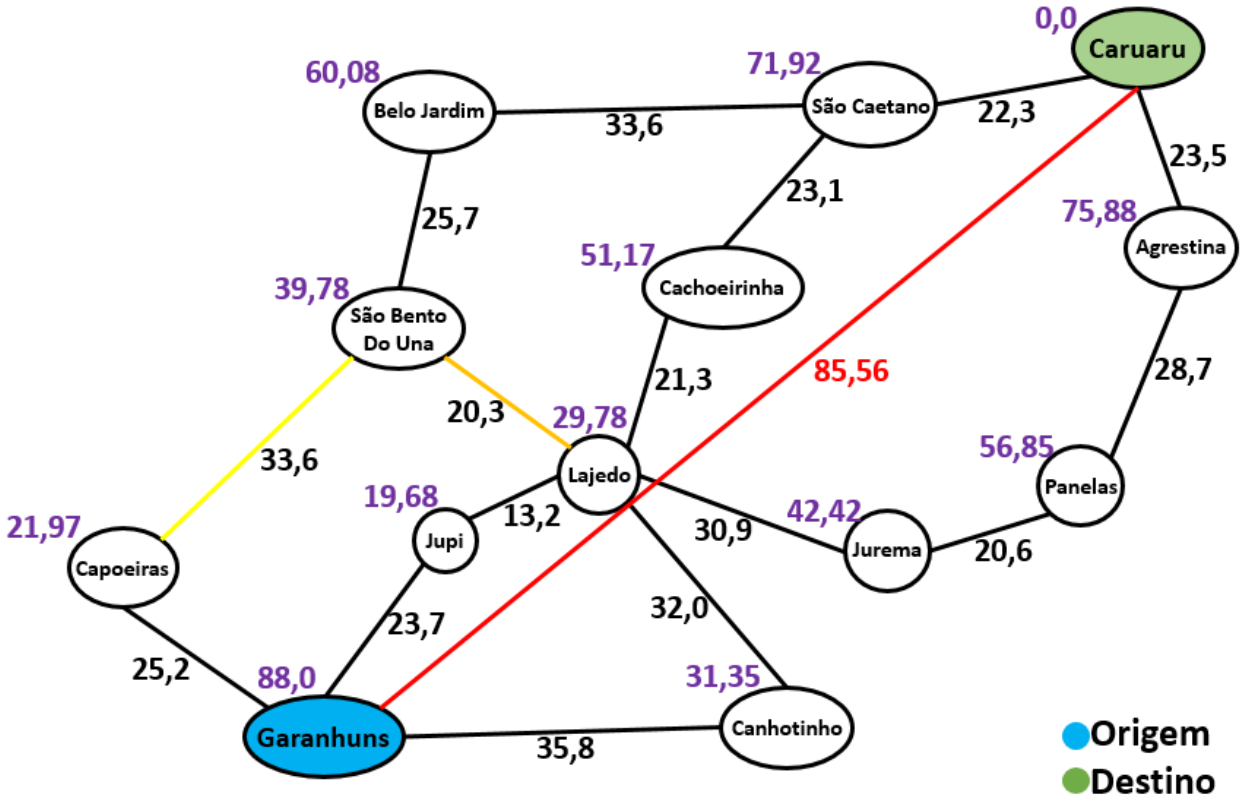

Figura 3: Mapa completo com todas as informações detalhadas.

O algoritmo foi construído na linguagem Python na versão 3.7 usando o conceito de orientação a objetos. Para desenvolver o algoritmo de busca A* foi modelado o problema através de um grafo não direcionado com seus respectivos vértices e arestas.

Três classes foram criadas para representar o grafo, a classe Vertice Figura 4, a classe Aresta Figura 5 e a classe Grafo Figura 6, onde contêm todos os métodos do grafo e o algoritmo A*.

```python
# coding: utf-8
class Vertice:

    BRANCO = 0 #public
    CINZA = 1 #public
    PRETO = 2 #public

    def __init__(self, nome, valor):
        #Declaração variáveis private
        self.nome = nome
        self.valor = valor #Valor heurístico
        self.custo = 0 #Calcular os somatórios
        self.cor = self.BRANCO #Inicia todos branco
        self.predecessor = None

    def getNome(self):
        return self.nome
    def setNome(self, nome):
        self.nome = nome

    def getValor(self):
        return self.valor
```

Figura 4: Classe Vertice e seus atributos e métodos em Python.

```python
# coding: utf-8
class Aresta:

    def __init__(self):
        #Declaração variáveis private
        self.origem = None
        self.destino = None
        self.peso = None #Km reais
        self.tempo = None #Tempo percuso

    def getOrigem(self):
        return self.origem
    def setOrigem(self, origem):
        self.origem = origem

    def getDestino(self):
        return self.destino
    def setDestino(self, destino):
        self.destino = destino

    def getPeso(self):
        return self.peso
```

Figura 5: Classe Aresta e seus atributos e métodos em Python.

```python
# coding: utf-8
from Vertice import *
from Aresta import *

class Grafo:

    def __init__(self):
        #Declaração variáveis private
        self.vertices = [] #lista de vertices
        self.arestas = [] #lista de arestas

    # Métodos padrão grafos
    def addVertice(self, nome, valor):
        if not self.contem(nome):
            self.vertices.append(Vertice(nome, valor))

    def addAresta(self, origem, destino, peso, tempo):
        vOrigem = self.obterVertice(origem)
        vDestino = self.obterVertice(destino)
        aresta_x = Aresta()
        aresta_x.setOrigem(vOrigem)
        aresta_x.setDestino(vDestino)
        aresta_x.setPeso(peso)
        aresta_x.setTempo(tempo)
        self.arestas.append(aresta_x)
```

Figura 6: Classe Grafo e seus atributos e métodos e importação das classes Vertice e Aresta.

Na classe Grafo como mostra à Figura 6, temos uma outra diversidade de métodos necessários e incluindo o método de busca A* como mostra a Figura 7, logo abaixo.

```
109     def buscaAestrela(self, origem, destino):
110         try:
111             caminhos = []
112             aux = []
113 
114             v_inicial = self.obterVertice(origem)
115             v_inicial.setCusto(0)
116             v_inicial.setCor(Vertice.CINZA)
117 
118             aux.append(v_inicial)
119             while(len(aux) != 0):
120                 u = aux.pop(0)
121                 caminhos.append(u.getNome())
122                 if u == self.obterVertice(destino):#Se é objetivo
123                     caminhos.append(None)#guardar total custo
124                     caminhos[-1] = "Total = "+str(u.getCusto())
125                     return caminhos#break;
126 
127                 menor = 2147483647 #MAX_VALUE
128                 menorVertice = None
129 
130                 for v in self.adjacentes(u.getNome()):
131 
132                     if v.getCor() != Vertice.PRETO:
133                         #G(n):
134                         v.setCusto(u.getCusto()+self.obterAresta(u, v).getPeso())
135                         #G(n) + a condição que é o tempo:
136                         peso = v.getCusto() + self.obterAresta(u, v).getTempo()
137                         #h <= h*:
138                         if ((peso + v.getValor()) <= menor and v.getCor() == Vertice.BRANCO):
139                             menor = peso + v.getValor() #G(n) + H(n)
140                             menorVertice = v
141                             menorVertice.setCusto(peso)
142                         v.setCor(Vertice.CINZA)
143 
144                 menorVertice.setCor(Vertice.PRETO)
145                 aux.append(menorVertice)
146                 u.setCor(Vertice.PRETO)
147         except:
148             return "Erro!\nVerifique os valores Heurísticos!"
149 
```

Figura 6: Algoritmo de busca A*.

Agora temos mais um arquivo Python chamado de Main Figura 8, que contém toda representação do mapa da Figura 3.

```python
# coding: utf-8
from Grafo import *

g = Grafo()
'''
1º paramentro, nome vertice
2º paramentro, valor heurístico
'''
g.addVertice("Garanhuns",85.56)
g.addVertice("Capoeiras",21.97)
g.addVertice("Jupi",19.68)
g.addVertice("Canhotinho",31.35)
g.addVertice("São Bento do Una",39.78)
g.addVertice("Lajedo",29.78)
g.addVertice("Jurema",42.42)
g.addVertice("Belo Jardim",60.08)
g.addVertice("Cachoeirinha",51.17)
g.addVertice("Panelas",56.85)
g.addVertice("São Caetano",71.92)
g.addVertice("Agrestina",75.88)
g.addVertice("Caruaru",0)
'''
Grafo não direcionado

1º paramentro, vertice origem
2º paramentro, vertice destino
3º paramentro, peso real da aresta
4º paramentro, tempo percuso
'''
g.addAresta("Garanhuns","Capoeiras",25.2, 0)
g.addAresta("Garanhuns","Jupi",23.7, 0)
g.addAresta("Garanhuns","Canhotinho",35.8, 0)

g.addAresta("Capoeiras","São Bento do Una",33.6, 25)#aresta amarela
g.addAresta("Jupi","Lajedo",13.2, 0)
g.addAresta("Canhotinho","Lajedo",32, 0)

g.addAresta("São Bento do Una","Belo Jardim",25.7, 0)
g.addAresta("São Bento do Una","Lajedo",20.3, 20)#aresta laranja
g.addAresta("Lajedo","Cachoeirinha",21.3, 0)
g.addAresta("Lajedo","Jurema",30.9, 0)

g.addAresta("Belo Jardim","São Caetano",33.6, 0)
g.addAresta("Cachoeirinha","São Caetano",23.1, 0)
g.addAresta("Panelas","Jurema",20.6, 0)
g.addAresta("Panelas","Agrestina",28.7, 0)

g.addAresta("São Caetano","Caruaru",22.3, 0)
g.addAresta("Agrestina","Caruaru",23.5, 0)

print(g.buscaAestrela("Garanhuns", "Caruaru"))

```

Figura 8: Arquivo Main com a criação do mapa em grafo.

Na saída do algoritmo obtivemos o seguinte resultado, Figura 9:

```
>>>
 RESTART: .\Main.py

['Garanhuns', 'Jupi', 'Lajedo', 'Cachoeirinha', 'São Caetano', 'Caruaru',
'Total = 103.60000000000001']
>>>
```

Figura 8: Execução da busca saindo de Garanhuns até Caruaru.

Mas, porque o resultado total é 103,60 Km se na Figura 1 mostra 96,8 Km? – esse valor maior um pouco é devido aos caminhos reais informados pelo Google Maps entre uma cidade e outra, por ter calculado um ponto de origem do centro da cidade ao outro centro da cidade destino.

Referências

COPPIN, Bem. **Inteligência Artificial**. Edição: 1. Editora: LTC; Edição: 1 (27 de junho de 2017).

NASCIMENTO, Serafim. **Busca Heurística (Parte 1): Busca Gulosa.** (2017). Disponível em: <https://www.youtube.com/watch?v=fJGmyfV9zPs>. Acesso em 09 de setembro de 2019.

NASCIMENTO, Serafim. **Busca Heurística (Parte 2): Busca A* (A estrela).** (2017). Disponível em: <https://www.youtube.com/watch?v=FU6JQaRMMDM>. Acesso em 09 de setembro de 2019.

Anexos

- Código fonte completo do projeto: https://github.com/Adjailson/AlgoritmoBusca
- Tabela com os valores heurísticos:

Nome da cidade	Heurística
Garanhuns (origem)	85,56
Capoeiras	21,97
Jupi	19,68
Canhotinho	31,35
São Bento do Una	39,78
Lajedo	29,78
Jurema	42,42
Belo Jardim	60,08
Cachoeirinha	51,17
Panelas	56,85

São Caetano	71,92
Agrestina	75,88
Caruaru (destino)	0