República Bolivariana de Venezuela

Universidad Centroccidental Lisandro Alvarado

Decanato De Ciencias y Tecnología

Departamento de Ingeniería Telemática

Barquisimeto-Lara

# Implementando A\*

Gabriel Rojas

28.454.911

Ingeniería Telemática

**Análisis del Problema:**

Se nos pide implementar el algoritmo A\* para encontrar la ruta más corta entre Arad y Bucarest en el mapa de Rumania, utilizando las heurísticas proporcionadas.

**Elementos Clave:**

* *Algoritmo A:*\* Un algoritmo de búsqueda informado que utiliza una función heurística para estimar la distancia al nodo objetivo.
* **Mapa de Rumania:** Un grafo que representa las ciudades y las carreteras que las conectan, con distancias asociadas.
* **Heurísticas:** Estimaciones de la distancia a Bucarest para cada ciudad.

**Codigo:**

import heapq

# Definimos el grafo que representa el mapa rodoviario de Rumania

grafo = {

'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},

'Zerind': {'Arad': 75, 'Oradea': 71},

'Oradea': {'Zerind': 71, 'Sibiu': 151},

'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},

'Timisoara': {'Arad': 118, 'Lugoj': 111},

'Lugoj': {'Timisoara': 111, 'Mehadia': 70},

'Mehadia': {'Lugoj': 70, 'Drobeta': 75},

'Drobeta': {'Mehadia': 75, 'Craiova': 120},

'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},

'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},

'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},

'Fagaras': {'Sibiu': 99, 'Bucharest': 211},

'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90},

'Giurgiu': {'Bucharest': 90},

'Eforie': {'Hirsova': 86},

'Hirsova': {'Eforie': 86, 'Urziceni': 98},

'Urziceni': {'Hirsova': 98, 'Vaslui': 142, 'Iasi': 92},

'Vaslui': {'Urziceni': 142, 'Iasi': 92, 'Neamt': 87},

'Iasi': {'Urziceni': 92, 'Vaslui': 92, 'Neamt': 87},

'Neamt': {'Iasi': 87, 'Sinaia': 84},

'Sinaia': {'Neamt': 84, 'Bucharest': 246},

'Sinaia': {'Neamt': 84, 'Bucharest': 246}

}

# Definimos la función de costo que calcule la distancia entre dos ciudades

def costo(grafo, ciudad1, ciudad2):

return grafo[ciudad1][ciudad2]

# Definimos la función heurística que estime la distancia desde una ciudad hasta 'Bucharest'

heuristica = {

'Arad': 366,

'Bucharest': 0,

'Craiova': 160,

'Drobeta': 242,

'Eforie': 161,

'Fagaras': 176,

'Giurgiu': 77,

'Hirsova': 151,

'Iasi': 226,

'Lugoj': 244,

'Mehadia': 241,

'Neamt': 234,

'Oradea': 380,

'Pitesti': 98,

'Rimnicu Vilcea': 193,

'Sibiu': 253,

'Sinaia': 140,

'Timisoara': 329,

'Urziceni': 80,

'Vaslui': 142,

'Zerind': 374

}

# Implementamos el algoritmo A\*

def a\_estrella(grafo, costo, heuristica, inicio, fin):

abierto = []

cerrado = set()

heapq.heappush(abierto, (0, inicio))

costo\_acumulado = {inicio: 0}

camino = {inicio: None}

while abierto:

\_, ciudad\_actual = heapq.heappop(abierto)

if ciudad\_actual == fin:

break

cerrado.add(ciudad\_actual)

for ciudad\_siguiente in grafo[ciudad\_actual]:

if ciudad\_siguiente in cerrado:

continue

costo\_nuevo = costo\_acumulado[ciudad\_actual] + costo(grafo, ciudad\_actual, ciudad\_siguiente)

if ciudad\_siguiente not in costo\_acumulado or costo\_nuevo < costo\_acumulado[ciudad\_siguiente]:

costo\_acumulado[ciudad\_siguiente] = costo\_nuevo

prioridad = costo\_nuevo + heuristica[ciudad\_siguiente]

heapq.heappush(abierto, (prioridad, ciudad\_siguiente))

camino[ciudad\_siguiente] = ciudad\_actual

# Reconstruimos el camino óptimo

camino\_optimo = []

ciudad\_actual = fin

while ciudad\_actual is not None:

camino\_optimo.append(ciudad\_actual)

ciudad\_actual = camino.get(ciudad\_actual)

camino\_optimo.reverse()

print("La ruta óptima es:", " -> ".join(camino\_optimo))

print("El costo total es:", costo\_acumulado[fin])

return camino\_optimo, costo\_acumulado[fin]

# Ejecutamos el algoritmo A\*

inicio = 'Arad'

fin = 'Bucharest'

a\_estrella(grafo, costo, heuristica, inicio, fin)