Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica

Inteligencia Artificial

Manual de Usuario del programa A*

Profesor:

Abel Herrera Camacho

Alumnos:

Carrasco Ruiz Alan Uriel

Jimenez Cervantes Angel Mauricio

Reyes Arroyo Pablo Antonio

Semestre 2024-1

Fecha de Entrega: 08 de noviembre de 2023

Contenido

Propósito

El propósito de este manual es facilitar la comprensión al usuario, de la operación de captura y consulta de la información que retorna el programa “Método de búsqueda A*”.

Objetivo del Manual

Este manual tiene la finalidad de dar a conocer las operaciones básicas del software que se requiere para generar una búsqueda de la ruta más optima entre dos ciudades usando el método de búsqueda A*, además de dar a conocer el método de emplear de manera correcta dicho software.

Introducción

Objetivo del programa A*

- El software debe poder iniciar en cualquier ciudad y llegar a Bucharest.
- El software debe indicar paso a paso su avance.
- Debe entregar la(s) trayectoria(s) óptima(s) y su costo total.

Implementación del Programa

a) Requerimientos del Hardware

Se deberá contar con:

- Computadora Personal
- Conexión a internet

b) Requerimientos de Software

Se deberá contar con:

- Sistema operativo Windows 8,10,11
- IDE de Python o en su defecto Visual Studio Code
- Python instalado

Instalación de bibliotecas necesarias

WINDOWS:

Con Python previamente instalado y configurado, bastará con ir al símbolo del sistema o poner el buscador “cmd”. Una vez abierto (símbolo de sistema), se deberá ingresar los siguientes comandos:

Para instalar la primera biblioteca:

```
$ pip install networkx
```

Para instalar la segunda biblioteca:

```
$ pip install -U matplotlib
```

LINUX:

Para el caso de Linux, es algo similar. Se deberá abrir una ventana de terminal (puede ser en el buscador o dentro de la terminal de visual studio code) y se deberán ingresar los siguientes comandos:

Para instalar la primera biblioteca:

```
$ pip3 install networkx
```

Para instalar la segunda biblioteca:

```
$ pip3 install -U matplotlib
```

Para mayor detalle de la instalación, proporcionamos los siguientes links de referencia:

Instalación de networkx:

https://networkx.org/documentation/stable/install.html

Instalación de matplotlib:

https://matplotlib.org/stable/users/installing/index.html

Módulos

a) Modulo main

```python
# Importa las bibliotecas necesarias
import networkx as nx  # Para trabajar con grafos
import matplotlib.pyplot as plt  # Para visualizar el grafo
from nodo import *
from costos import *
import visitas

# Define la función principal del programa
def main():

    # Lista de ciudades disponibles
    cities = [
    'Oradea', 'Zerind', 'Arad', 'Sibiu', 'Timisoara', 'Lugoj', 'Mehadia',
    'Dobreta', 'Craiova', 'Rimnicu Vilcea', 'Fagaras', 'Pitesti',
    'Bucharest', 'Giurgiu', 'Urziceni','Hirsova', 'Eforie', 'Vaslui',
    'Iasi', 'Neamt']

    # Imprime la lista de ciudades disponibles
    print("\n\t **************************** Metodo de Busqueda A* usando Mapa de Rumania 2 ****************************\n")
    print("\n\t ======================================== Ciudades Disponibles ========================================\n\n")
    print(cities)
    print("\n\t ======================================================================================================\n\n")
    # Solicita al usuario que ingrese una ciudad de inicio válida
    flag = False
    while(flag == False):
        print()
        start = input('Ingresa el punto de partida: ')
        end = "Bucharest"
        if(start in cities):
            flag = True
    print("\n")

    # Configuración del grafo
    numNodes = len(cities)
    numEdges = 23
    directed = False
    cost = True

    # Costos directos desde cada ciudad a Bucarest
    straight_Cost = {'Arad':366, 'Bucharest':0, 'Craiova':160, 'Dobreta':242,
        'Eforie':161, 'Fagaras':178, 'Giurgiu':77, 'Hirsova':151, 'Iasi':226,
        'Lugoj':244, 'Mehadia':241, 'Neamt':234, 'Oradea':380, 'Pitesti':98,
        'Rimnicu Vilcea':193, 'Sibiu':253, 'Timisoara':329, 'Urziceni':80, 'Vaslui':199,
        'Zerind':374}

    # Crear una instancia de la clase 'costos'
    Camino = costos(numNodes, numEdges, directed, cost, straight_Cost)
    Camino.start_costos(sorted(cities))

    # Si la ciudad de inicio es "Bucharest", muestra la información y termina
    if(start == "Bucharest"):
        print('Trayectoria: ', start)
        print('El costo total es: ', Camino.straight_Cost.get(start))
        return

    # Si la ciudad de inicio es diferente, muestra la información de inicio y ejecuta visitas
    else:
        print('Ciudad inicial', start)
        print('Costo: ', Camino.straight_Cost.get(start))
        visitas.visitas(Camino, start, 0, start ,end)

        # Crea un grafo utilizando networkx
        G = nx.Graph()

        # Agrega nodos y bordes al grafo desde la variable 'visitas.trajectory'
        for item in visitas.trajectory:
            cost, path = list(item.keys())[0], list(item.values())[0]
            cities_in_path = path.split('/')
            for i in range(len(cities_in_path) - 1):
                G.add_edge(cities_in_path[i], cities_in_path[i + 1], weight=cost)

        # Ajusta la posición de los nodos en el grafo y define etiquetas de bordes
        pos = nx.spring_layout(G)
        labels = nx.get_edge_attributes(G, 'weight')

        # Dibuja el grafo y muestra las etiquetas de los bordes
        nx.draw(G, pos, with_labels=True,node_size=1000, node_color="lightblue", font_size=8, font_weight='bold', font_color="black")
        nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

        # Muestra el grafo
        plt.suptitle("Metodo Grafico A*")
        plt.show()

# Verifica si el código se ejecuta como un script principal
if __name__ == '__main__':
    import grafico_Rumania# mandamos el mapa para que el usuario observe la ciudad de inicio y tiene que cerrarlo para continuar
    main()
```

b) Modulo grafico_Rumania

```python
#Se crea el grafo aparte,para unicamente lectura por parte del usuario
import networkx as nx
import matplotlib.pyplot as plt


# Define la clase nodo
class Nodo:
    def __init__(self, ciudad):
        self.ciudad = ciudad
        self.adjacent_Nodes = []

# Define la clase costos
class Costos:
    def __init__(self):
        self.nodes = []

    def add_city(self, ciudad, conexiones):
        node = Nodo(ciudad)
        for conexion, peso in conexiones.items():
            node.adjacent_Nodes.append((conexion, peso))
        self.nodes.append(node)

# Crear una instancia de la clase Costos
mapa_rumania = Costos()

# Agregar ciudades y conexiones
mapa_rumania.add_city("Arad", {"Zerind": 75, "Sibiu": 140, "Timisoara": 118})
mapa_rumania.add_city("Bucharest", {"Fagaras": 211, "Pitesti": 101, "Giurgiu": 90, "Urziceni": 85})
mapa_rumania.add_city("Craiova", {"Dobreta": 120, "Rimnicu Vilcea": 146, "Pitesti": 138})
mapa_rumania.add_city("Dobreta", {"Craiova": 120, "Mehadia": 75})
mapa_rumania.add_city("Eforie", {"Hirsova": 86})
mapa_rumania.add_city("Fagaras", {"Sibiu": 99, "Bucharest": 211})
mapa_rumania.add_city("Giurgiu", {"Bucharest": 90})
mapa_rumania.add_city("Hirsova", {"Urziceni": 98, "Eforie": 86})
mapa_rumania.add_city("Iasi", {"Neamt": 87, "Vaslui": 92})
mapa_rumania.add_city("Lugoj", {"Timisoara": 111, "Mehadia": 70})
mapa_rumania.add_city("Mehadia", {"Lugoj": 70, "Dobreta": 75})
mapa_rumania.add_city("Neamt", {"Iasi": 87})
mapa_rumania.add_city("Oradea", {"Zerind": 71, "Sibiu": 151})
mapa_rumania.add_city("Pitesti", {"Rimnicu Vilcea": 97, "Craiova": 138, "Bucharest": 101})
mapa_rumania.add_city("Rimnicu Vilcea", {"Sibiu": 80, "Craiova": 146, "Pitesti": 97})
mapa_rumania.add_city("Sibiu", {"Oradea": 151, "Arad": 140, "Rimnicu Vilcea": 80, "Fagaras": 99})
mapa_rumania.add_city("Timisoara", {"Lugoj": 111, "Arad": 118})
mapa_rumania.add_city("Urziceni", {"Bucharest": 85, "Hirsova": 98, "Vaslui": 142})
mapa_rumania.add_city("Vaslui", {"Iasi": 92, "Urziceni": 142})
mapa_rumania.add_city("Zerind", {"Oradea": 71, "Arad": 75})

# Crear un objeto grafo de NetworkX
G = nx.Graph()

# Agregar nodos y aristas al grafo
for node in mapa_rumania.nodes:
    G.add_node(node.ciudad)
    for conexion, peso in node.adjacent_Nodes:
        G.add_edge(node.ciudad, conexion, weight=peso)

# Dibujar el grafo

pos = nx.spring_layout(G)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw(G, pos, with_labels=True, node_size=1000, node_color="lightblue", font_size=8, font_weight='bold', font_color="black")
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.title("Mapa de Rumania")
plt.suptitle("Mapa de Rumania 2")

plt.show()

```

c) Modulo nodo

```python
class nodo: #Clase Nodo.
    adjacent_Nodes = []
    ciudad = ""

    def __init__(self, ciudad):
        self.ciudad = ciudad
        self.adjacent_Nodes = []

```

d) Modulo costos

```python
import nodo #Creación del Grafo del Mapa de Rumania 2 a usar en el programa

class costos:
    nodes = []
    num_Nodes = 0
    num_Edges = 0
    directed = False
    cost = False
    straight_Cost = {}

    def __init__(self, numNodes, numEdges, directed, cost, straight_Cost):
        self.num_Nodes = numNodes
        self.num_Edges = numEdges
        self.directed = directed
        self.cost = cost
        self.straight_Cost = straight_Cost

    def start_costos(self, cities):
        self.add_cities(cities)
        self.add_edges()


    def add_cities(self, cities):
        for i in cities:
            self.nodes.append(nodo.nodo(i))

```

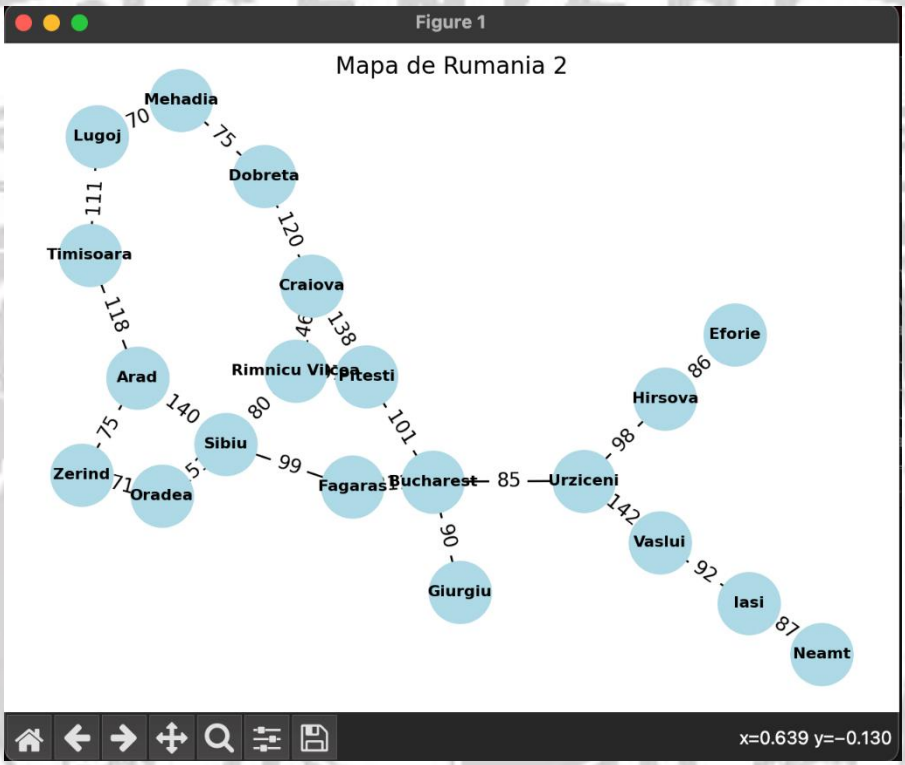
```python
def add_edges(self):
    #Se añaden las ciudades vecinas.
    #Arad
    self.nodes[0].adjacent_Nodes.append({'Zerind':75})
    self.nodes[0].adjacent_Nodes.append({'Sibiu':140})
    self.nodes[0].adjacent_Nodes.append({'Timisoara':118})

    #Bucharest
    self.nodes[1].adjacent_Nodes.append({'Fagaras':211})
    self.nodes[1].adjacent_Nodes.append({'Pitesti':101})
    self.nodes[1].adjacent_Nodes.append({'Giurgiu':90})
    self.nodes[1].adjacent_Nodes.append({'Urziceni':85})

    #Craiova
    self.nodes[2].adjacent_Nodes.append({'Dobreta':120})
    self.nodes[2].adjacent_Nodes.append({'Rimnicu Vilcea':146})
    self.nodes[2].adjacent_Nodes.append({'Pitesti':138})

    #Dobreta
    self.nodes[3].adjacent_Nodes.append({'Craiova':120})
    self.nodes[3].adjacent_Nodes.append({'Mehadia':75})

    #Eforie
    self.nodes[4].adjacent_Nodes.append({'Hirsova':86})

    #Fagaras
    self.nodes[5].adjacent_Nodes.append({'Sibiu':99})
    self.nodes[5].adjacent_Nodes.append({'Bucharest':211})

    #Giurgiu
    self.nodes[6].adjacent_Nodes.append({'Bucharest':90})

    #Hirsova
    self.nodes[7].adjacent_Nodes.append({'Urziceni':98})
    self.nodes[7].adjacent_Nodes.append({'Eforie':86})

    #Iasi
    self.nodes[8].adjacent_Nodes.append({'Neamt':87})
    self.nodes[8].adjacent_Nodes.append({'Vaslui':92})

    #Lugoj
    self.nodes[9].adjacent_Nodes.append({'Timisoara':111})
    self.nodes[9].adjacent_Nodes.append({'Mehadia':70})

    #Mehadia
    self.nodes[10].adjacent_Nodes.append({'Lugoj':70})
    self.nodes[10].adjacent_Nodes.append({'Dobreta':75})

    #Neamt
    self.nodes[11].adjacent_Nodes.append({'Iasi':87})

    #Oradea
    self.nodes[12].adjacent_Nodes.append({'Zerind':71})
    self.nodes[12].adjacent_Nodes.append({'Sibiu':151})

    #Pitesti
    self.nodes[13].adjacent_Nodes.append({'Rimnicu Vilcea':97})
    self.nodes[13].adjacent_Nodes.append({'Craiova':138})
    self.nodes[13].adjacent_Nodes.append({'Bucharest':101})

    #Rimnicu
    self.nodes[14].adjacent_Nodes.append({'Sibiu':80})
    self.nodes[14].adjacent_Nodes.append({'Craiova':146})
    self.nodes[14].adjacent_Nodes.append({'Pitesti':97})

    #Sibiu
    self.nodes[15].adjacent_Nodes.append({'Oradea':151})
    self.nodes[15].adjacent_Nodes.append({'Arad':140})
    self.nodes[15].adjacent_Nodes.append({'Rimnicu Vilcea':80})
    self.nodes[15].adjacent_Nodes.append({'Fagaras':99})

    #Timisoara
    self.nodes[16].adjacent_Nodes.append({'Lugoj':111})
    self.nodes[16].adjacent_Nodes.append({'Arad':118})

    #Urziceni
    self.nodes[17].adjacent_Nodes.append({'Bucharest':85})
    self.nodes[17].adjacent_Nodes.append({'Hirsova':98})
    self.nodes[17].adjacent_Nodes.append({'Vaslui':142})

    #Vaslui
    self.nodes[18].adjacent_Nodes.append({'Iasi':92})
    self.nodes[18].adjacent_Nodes.append({'Urziceni':142})

    #Zerind
    self.nodes[19].adjacent_Nodes.append({'Oradea':71})
    self.nodes[19].adjacent_Nodes.append({'Arad':75})
```

e) Modulo visitas

```python
visit = []
trajectory = []
globCity = ''
h = 1

def visitas(objcostos, strCurrentCity, temp2, start, end):
    global visit, trajectory, globCity, h

    print("================================================== Paso "+str(h)+"  ==================================================\n")
    visit.append(strCurrentCity) #La ciudad actual se añade a la lista de visitados.
    for i in objcostos.nodes: # El ciclo sirve para recorrer la lista de los nodos que son adyacentes.
        if(i.ciudad == strCurrentCity): # Si la ciudad de la lista es la misma a la que se le está pasando en la función entra al otro ciclo for
            for j in i.adjacent_Nodes: # Se busca el costo de los nodos vecinos a la ciudad actual por ejemplo SIBIU = FAG,RIMNI,ARAD, 0RAD
                if(str( list(j.keys())[0] ) in visit):  #Si los vecinos del nuevo nodo ya están en visitados no se vuelven a agregar.
                    continue
                else: #Caso contrario los vecinos que no se han marcado como visitados se tienen que agregar en el diccionario.
                    tmp_city = str( list(j.keys())[0] ) # Se guarda temporalmente la ciudad vecina.

                    if(h == 1):
                        tmp_city += '/'  + globCity + start # Solo sucede en la primera llamada a la función, sirve para agregar a Sibiu en la trayectoria.

                    tmp_city += '/'  + globCity #Se actualiza la trayectoria de la ciudad -> EJEMPLO SIBIU/FAGARAS/CRAVIOTA.
                    tmp_distance = (int( list(j.values())[0])) + temp2 # Se suma la distancia de la arista y del origen(temp2) se actualiza antes de llamarse recursivamente.
                    tmp_total = tmp_distance
                    tmp_total += objcostos.straight_Cost.get(str(list(j.keys())[0])) # Se suma el costo de linea directa de Bucharest.
                    print("Suma de la ciudad ", tmp_city, ": ", tmp_distance, " + ", objcostos.straight_Cost.get(str(list(j.keys())[0])), "=", tmp_total)
                    trajectory.append({tmp_total:tmp_city})                          # Se agrega en la lista el diccionario la ciudad y su costo.


    print("\nCiudades visitadas", visit)
    print('Nodos y distancias finales: ', trajectory)
    h += 1

    globCity, min_distance = minimum() #Se regresa el valor más pequeño de la lista junto con su ciudad.

    for j in trajectory: # Caso contrario al anterior for se va verificando si ya llegamos a Bucharest con la ruta mas óptima.
        if(end in str( list(j.values())[0]) and min_distance == list(j.keys())[0]):
            print('\nTrayectoria: ', list(j.values())[0])
            print('El costo total es: ', list(j.keys())[0])
            return

    trajectory.remove({min_distance:globCity}) # Se borra la ciudad y su costo de la lista por que ya no pertece a la lista de nodos y distancias.
    min_city = globCity.split('/')[0] # Se obtiene únicamente la ciudad de menor distancia, no toda la trayectoria.
    min_distance -= objcostos.straight_Cost.get(min_city) # Para sumar los siguientes valores se tiene que restar el costo directo, esto sirve para sumarselos a las ciudades vecinas.

    print("\n-> Trayectoria con costo mínimo: '"+globCity+"'\n")
    visitas(objcostos, min_city, min_distance, start, end) #Se llama recursivamente la función.


def minimum(): # Función que regresa el valor mínimo de la ciudad en la lista trayectoría, con esto buscamos la forma más óptima para avanzar a la ciudad destino
    global trajectory
    strTmp = []
    intTmp = []
    for i in trajectory:
        intTmp.append(int( list(i.keys())[0] ))
        strTmp.append(str( list(i.values())[0] ))

    for j in trajectory:
        if(j.get(min(intTmp)) != None):
            return j.get(min(intTmp)), min(intTmp)
```

Modo de uso

Step 1. Al levantar el proyecto con el comando correspondiente (python main.py), se presentará en pantalla el mapa de Rumania. Se deberá cerrar para poder continuar con la ejecución.



Step 2. Se deberá observar el listado de las ciudades disponibles de las cuales se puede partir, para posteriormente seleccionar una.

```
****************************** Metodo de Busqueda A* usando Mapa de Rumania 2 ******************************

======================================== Ciudades Disponibles ========================================

['Oradea', 'Zerind', 'Arad', 'Sibiu', 'Timisoara', 'Lugoj', 'Mehadia', 'Dobreta', 'Craiova', 'Rimnicu Vilcea', 'Fagaras', 'Pitesti'
, 'Bucharest', 'Giurgiu', 'Urziceni', 'Hirsova', 'Eforie', 'Vaslui', 'Iasi', 'Neamt']

==========================================================================================================
```

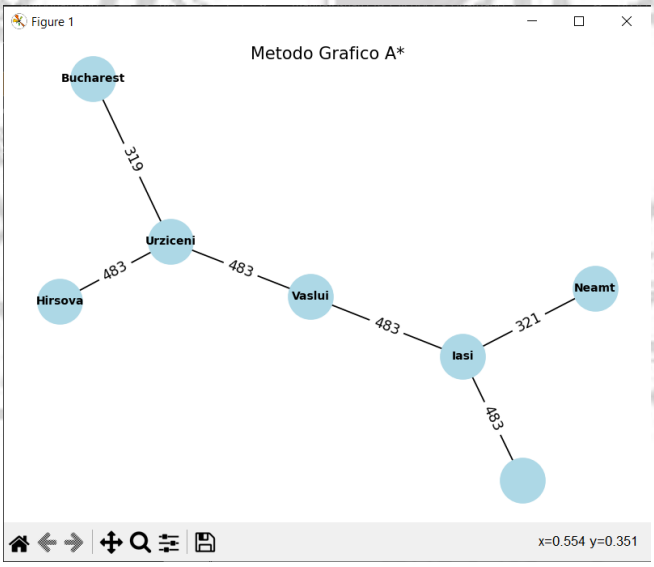
Step 3. Una vez seleccionada la ciudad de partida, se deberá ingresar el nombre de ésta respetando mayúsculas y minúsculas.

```
Ingresa el punto de partida:
```

Step 4: Se mostrará en pantalla la implementación del método de búsqueda, mostrando las rutas creadas y sus respectivos costos, además de los nodos visitados con respecto a cada iteración.

```
Ingresa el punto de partida: Iasi


Ciudad inicial Iasi
Costo:  226
================================================ Paso 1  ================================================

Suma de la ciudad  Neamt/Iasi/ :  87  +  234 = 321
Suma de la ciudad  Vaslui/Iasi/ :  92  +  199 = 291

Ciudades visitadas ['Iasi']
Nodos y distancias finales:  [{321: 'Neamt/Iasi/'}, {291: 'Vaslui/Iasi/'}]

-> Trayectoria con costo mínimo: 'Vaslui/Iasi/'

================================================ Paso 2  ================================================

Suma de la ciudad  Urziceni/Vaslui/Iasi/ :  234  +  80 = 314

Ciudades visitadas ['Iasi', 'Vaslui']
Nodos y distancias finales:  [{321: 'Neamt/Iasi/'}, {314: 'Urziceni/Vaslui/Iasi/'}]

-> Trayectoria con costo mínimo: 'Urziceni/Vaslui/Iasi/'

================================================ Paso 3  ================================================

Suma de la ciudad  Bucharest/Urziceni/Vaslui/Iasi/ :  319  +  0 = 319
Suma de la ciudad  Hirsova/Urziceni/Vaslui/Iasi/ :  332  +  151 = 483

Ciudades visitadas ['Iasi', 'Vaslui', 'Urziceni']
Nodos y distancias finales:  [{321: 'Neamt/Iasi/'}, {319: 'Bucharest/Urziceni/Vaslui/Iasi/'}, {483: 'Hirsova/Urziceni/Vaslui/Iasi/'
}]

Trayectoria:  Bucharest/Urziceni/Vaslui/Iasi/
El costo total es:  319
```

Step 5. Se presentará en pantalla el mapa el recorrido realizado en el mapa.

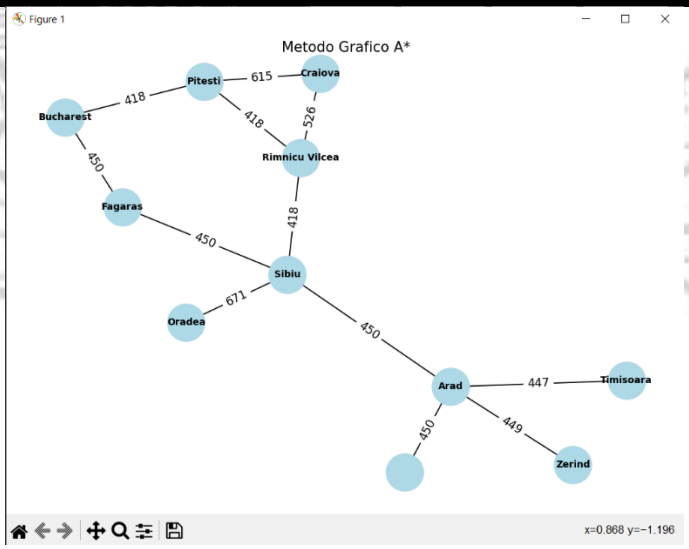


Step 6. Se deberá cerrar el mapa para terminar la ejecución del programa.

Pruebas

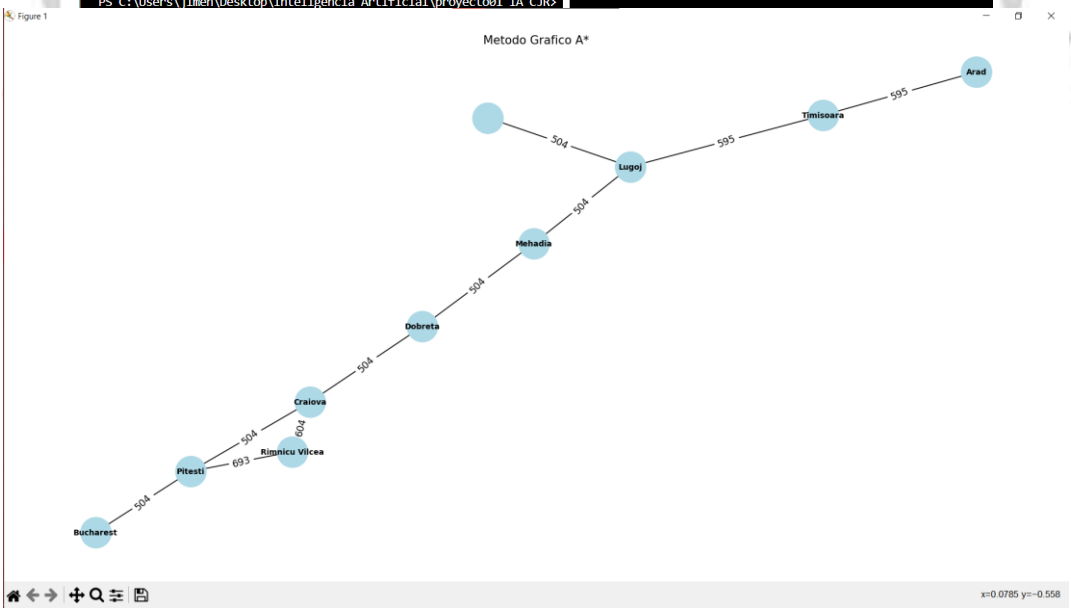
a) Ejemplo 1: Arad a Bucharest

```
Ingresa el punto de partida: Arad


Ciudad inicial Arad
Costo:  366
================================================== Paso 1  ==================================================

Suma de la ciudad  Zerind/Arad/ :  75  +  374 = 449
Suma de la ciudad  Sibiu/Arad/ :  140  +  253 = 393
Suma de la ciudad  Timisoara/Arad/ :  118  +  329 = 447

Ciudades visitadas ['Arad']
Nodos y distancias finales:  [{449: 'Zerind/Arad/'}, {393: 'Sibiu/Arad/'}, {447: 'Timisoara/Arad/'}]

-> Trayectoria con costo mínimo: 'Sibiu/Arad/'

================================================== Paso 2  ==================================================

Suma de la ciudad  Oradea/Sibiu/Arad/ :  291  +  380 = 671
Suma de la ciudad  Rimnicu Vilcea/Sibiu/Arad/ :  220  +  193 = 413
Suma de la ciudad  Fagaras/Sibiu/Arad/ :  239  +  178 = 417

Ciudades visitadas ['Arad', 'Sibiu']
Nodos y distancias finales:  [{449: 'Zerind/Arad/'}, {447: 'Timisoara/Arad/'}, {671: 'Oradea/Sibiu/Arad/'}, {413: 'Rimnicu Vilcea/S
ibiu/Arad/'}, {417: 'Fagaras/Sibiu/Arad/'}]

-> Trayectoria con costo mínimo: 'Rimnicu Vilcea/Sibiu/Arad/'

================================================== Paso 3  ==================================================

Suma de la ciudad  Craiova/Rimnicu Vilcea/Sibiu/Arad/ :  366  +  160 = 526
Suma de la ciudad  Pitesti/Rimnicu Vilcea/Sibiu/Arad/ :  317  +  98 = 415

Ciudades visitadas ['Arad', 'Sibiu', 'Rimnicu Vilcea']
Nodos y distancias finales:  [{449: 'Zerind/Arad/'}, {447: 'Timisoara/Arad/'}, {671: 'Oradea/Sibiu/Arad/'}, {417: 'Fagaras/Sibiu/Ar
ad/'}, {526: 'Craiova/Rimnicu Vilcea/Sibiu/Arad/'}, {415: 'Pitesti/Rimnicu Vilcea/Sibiu/Arad/'}]

-> Trayectoria con costo mínimo: 'Pitesti/Rimnicu Vilcea/Sibiu/Arad/'

================================================== Paso 4  ==================================================

Suma de la ciudad  Craiova/Pitesti/Rimnicu Vilcea/Sibiu/Arad/ :  455  +  160 = 615
Suma de la ciudad  Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/ :  418  +  0 = 418

Ciudades visitadas ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti']
Nodos y distancias finales:  [{449: 'Zerind/Arad/'}, {447: 'Timisoara/Arad/'}, {671: 'Oradea/Sibiu/Arad/'}, {417: 'Fagaras/Sibiu/Ar
ad/'}, {526: 'Craiova/Rimnicu Vilcea/Sibiu/Arad/'}, {615: 'Craiova/Pitesti/Rimnicu Vilcea/Sibiu/Arad/'}, {418: 'Bucharest/Pitesti/R
imnicu Vilcea/Sibiu/Arad/'}]

-> Trayectoria con costo mínimo: 'Fagaras/Sibiu/Arad/'

================================================== Paso 5  ==================================================

Suma de la ciudad  Bucharest/Fagaras/Sibiu/Arad/ :  450  +  0 = 450

Ciudades visitadas ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Fagaras']
Nodos y distancias finales:  [{449: 'Zerind/Arad/'}, {447: 'Timisoara/Arad/'}, {671: 'Oradea/Sibiu/Arad/'}, {526: 'Craiova/Rimnicu
Vilcea/Sibiu/Arad/'}, {615: 'Craiova/Pitesti/Rimnicu Vilcea/Sibiu/Arad/'}, {418: 'Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/'}, {
450: 'Bucharest/Fagaras/Sibiu/Arad/'}]

Trayectoria:  Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/
El costo total es:  418
PS C:\Users\jimen\Desktop\Inteligencia Artificial\proyecto01_IA_CJR>
```



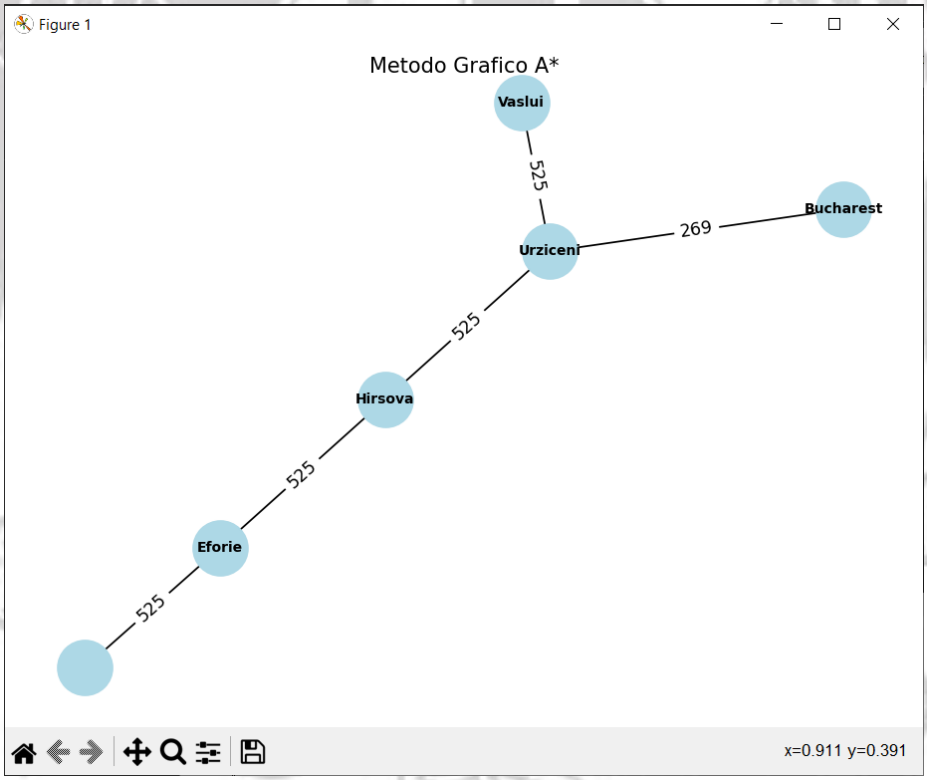
b) Ejemplo 2: Lugoj a Bucharest

```
Ingresa el punto de partida: Lugoj


Ciudad inicial Lugoj
Costo:  244
================================================== Paso 1  ==================================================

Suma de la ciudad  Timisoara/Lugoj/ :  111  +  329 = 440
Suma de la ciudad  Mehadia/Lugoj/ :  70  +  241 = 311

Ciudades visitadas ['Lugoj']
Nodos y distancias finales:  [{440: 'Timisoara/Lugoj/'}, {311: 'Mehadia/Lugoj/'}]

-> Trayectoria con costo mínimo: 'Mehadia/Lugoj/'

================================================== Paso 2  ==================================================

Suma de la ciudad  Dobreta/Mehadia/Lugoj/ :  145  +  242 = 387

Ciudades visitadas ['Lugoj', 'Mehadia']
Nodos y distancias finales:  [{440: 'Timisoara/Lugoj/'}, {387: 'Dobreta/Mehadia/Lugoj/'}]

-> Trayectoria con costo mínimo: 'Dobreta/Mehadia/Lugoj/'

================================================== Paso 3  ==================================================

Suma de la ciudad  Craiova/Dobreta/Mehadia/Lugoj/ :  265  +  160 = 425

Ciudades visitadas ['Lugoj', 'Mehadia', 'Dobreta']
Nodos y distancias finales:  [{440: 'Timisoara/Lugoj/'}, {425: 'Craiova/Dobreta/Mehadia/Lugoj/'}]

-> Trayectoria con costo mínimo: 'Craiova/Dobreta/Mehadia/Lugoj/'

================================================== Paso 4  ==================================================

Suma de la ciudad  Rimnicu Vilcea/Craiova/Dobreta/Mehadia/Lugoj/ :  411  +  193 = 604
Suma de la ciudad  Pitesti/Craiova/Dobreta/Mehadia/Lugoj/ :  403  +  98 = 501

Ciudades visitadas ['Lugoj', 'Mehadia', 'Dobreta', 'Craiova']
Nodos y distancias finales:  [{440: 'Timisoara/Lugoj/'}, {604: 'Rimnicu Vilcea/Craiova/Dobreta/Mehadia/Lugoj/'}, {501: 'Pitesti/Cra
iova/Dobreta/Mehadia/Lugoj/'}]

-> Trayectoria con costo mínimo: 'Timisoara/Lugoj/'

================================================== Paso 5  ==================================================

Suma de la ciudad  Arad/Timisoara/Lugoj/ :  229  +  366 = 595

Ciudades visitadas ['Lugoj', 'Mehadia', 'Dobreta', 'Craiova', 'Timisoara']
Nodos y distancias finales:  [{604: 'Rimnicu Vilcea/Craiova/Dobreta/Mehadia/Lugoj/'}, {501: 'Pitesti/Craiova/Dobreta/Mehadia/Lugoj/
'}, {595: 'Arad/Timisoara/Lugoj/'}]

-> Trayectoria con costo mínimo: 'Pitesti/Craiova/Dobreta/Mehadia/Lugoj/'

================================================== Paso 6  ==================================================

Suma de la ciudad  Rimnicu Vilcea/Pitesti/Craiova/Dobreta/Mehadia/Lugoj/ :  500  +  193 = 693
Suma de la ciudad  Bucharest/Pitesti/Craiova/Dobreta/Mehadia/Lugoj/ :  504  +  0 = 504

Ciudades visitadas ['Lugoj', 'Mehadia', 'Dobreta', 'Craiova', 'Timisoara', 'Pitesti']
Nodos y distancias finales:  [{604: 'Rimnicu Vilcea/Craiova/Dobreta/Mehadia/Lugoj/'}, {595: 'Arad/Timisoara/Lugoj/'}, {693: 'Rimnic
u Vilcea/Pitesti/Craiova/Dobreta/Mehadia/Lugoj/'}, {504: 'Bucharest/Pitesti/Craiova/Dobreta/Mehadia/Lugoj/'}]

Trayectoria:  Bucharest/Pitesti/Craiova/Dobreta/Mehadia/Lugoj/
El costo total es:  504
PS C:\Users\jimen\Desktop\Inteligencia Artificial\proyecto01 IA CJR>
```



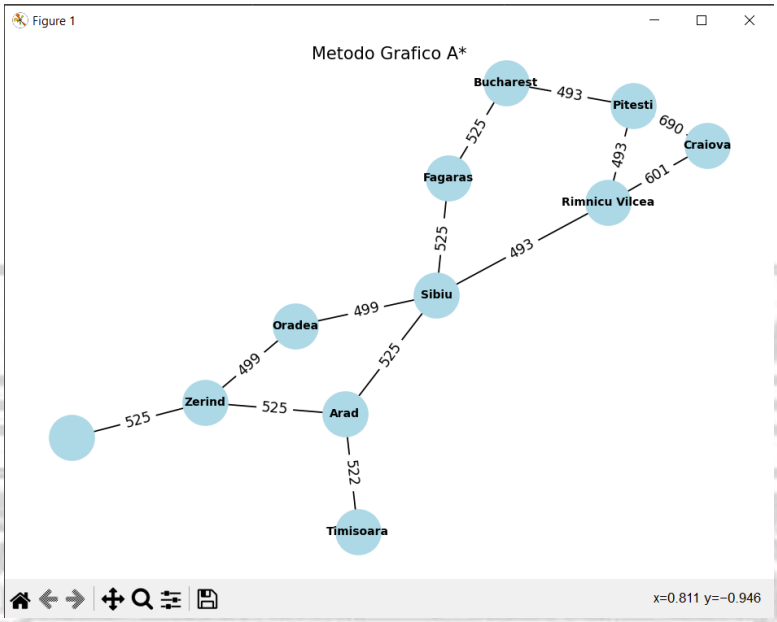
c) Ejemplo 3: Eforie a Bucharest

```
Ciudad inicial Eforie
Costo:  161
================================================ Paso 1  ================================================

Suma de la ciudad  Hirsova/Eforie/ :  86  +  151 = 237

Ciudades visitadas ['Eforie']
Nodos y distancias finales:  [{237: 'Hirsova/Eforie/'}]

-> Trayectoria con costo mínimo: 'Hirsova/Eforie/'

================================================ Paso 2  ================================================

Suma de la ciudad  Urziceni/Hirsova/Eforie/ :  184  +  80 = 264

Ciudades visitadas ['Eforie', 'Hirsova']
Nodos y distancias finales:  [{264: 'Urziceni/Hirsova/Eforie/'}]

-> Trayectoria con costo mínimo: 'Urziceni/Hirsova/Eforie/'

================================================ Paso 3  ================================================

Suma de la ciudad  Bucharest/Urziceni/Hirsova/Eforie/ :  269  +  0 = 269
Suma de la ciudad  Vaslui/Urziceni/Hirsova/Eforie/ :  326  +  199 = 525

Ciudades visitadas ['Eforie', 'Hirsova', 'Urziceni']
Nodos y distancias finales:  [{269: 'Bucharest/Urziceni/Hirsova/Eforie/'}, {525: 'Vaslui/Urziceni/Hirsova/Eforie/'}]

Trayectoria:  Bucharest/Urziceni/Hirsova/Eforie/
El costo total es:  269
PS C:\Users\jimen\Desktop\Inteligencia Artificial\proyecto01_IA_CJR>
```



d) Ejemplo 4: Zerind a Bucharest

```
Ingresa el punto de partida: Zerind


Ciudad inicial Zerind
Costo:  374
================================================== Paso 1  ==================================================

Suma de la ciudad  Oradea/Zerind/ :  71  +  380 = 451
Suma de la ciudad  Arad/Zerind/ :  75  +  366 = 441

Ciudades visitadas ['Zerind']
Nodos y distancias finales:  [{451: 'Oradea/Zerind/'}, {441: 'Arad/Zerind/'}]

-> Trayectoria con costo mínimo: 'Arad/Zerind/'

================================================== Paso 2  ==================================================

Suma de la ciudad  Sibiu/Arad/Zerind/ :  215  +  253 = 468
Suma de la ciudad  Timisoara/Arad/Zerind/ :  193  +  329 = 522

Ciudades visitadas ['Zerind', 'Arad']
Nodos y distancias finales:  [{451: 'Oradea/Zerind/'}, {468: 'Sibiu/Arad/Zerind/'}, {522: 'Timisoara/Arad/Zerind/'}]

-> Trayectoria con costo mínimo: 'Oradea/Zerind/'

================================================== Paso 3  ==================================================

Suma de la ciudad  Sibiu/Oradea/Zerind/ :  222  +  253 = 475

Ciudades visitadas ['Zerind', 'Arad', 'Oradea']
Nodos y distancias finales:  [{468: 'Sibiu/Arad/Zerind/'}, {522: 'Timisoara/Arad/Zerind/'}, {475: 'Sibiu/Oradea/Zerind/'}]

-> Trayectoria con costo mínimo: 'Sibiu/Arad/Zerind/'

================================================== Paso 4  ==================================================

Suma de la ciudad  Rimnicu Vilcea/Sibiu/Arad/Zerind/ :  295  +  193 = 488
Suma de la ciudad  Fagaras/Sibiu/Arad/Zerind/ :  314  +  178 = 492

Ciudades visitadas ['Zerind', 'Arad', 'Oradea', 'Sibiu']
Nodos y distancias finales:  [{522: 'Timisoara/Arad/Zerind/'}, {475: 'Sibiu/Oradea/Zerind/'}, {488: 'Rimnicu Vilcea/Sibiu/Arad/Zeri
nd/'}, {492: 'Fagaras/Sibiu/Arad/Zerind/'}]

-> Trayectoria con costo mínimo: 'Sibiu/Oradea/Zerind/'

================================================== Paso 5  ==================================================

Suma de la ciudad  Rimnicu Vilcea/Sibiu/Oradea/Zerind/ :  302  +  193 = 495
Suma de la ciudad  Fagaras/Sibiu/Oradea/Zerind/ :  321  +  178 = 499

Ciudades visitadas ['Zerind', 'Arad', 'Oradea', 'Sibiu', 'Sibiu']
Nodos y distancias finales:  [{522: 'Timisoara/Arad/Zerind/'}, {488: 'Rimnicu Vilcea/Sibiu/Arad/Zerind/'}, {492: 'Fagaras/Sibiu/Ara
d/Zerind/'}, {495: 'Rimnicu Vilcea/Sibiu/Oradea/Zerind/'}, {499: 'Fagaras/Sibiu/Oradea/Zerind/'}]

-> Trayectoria con costo mínimo: 'Rimnicu Vilcea/Sibiu/Arad/Zerind/'

================================================== Paso 6  ==================================================

Suma de la ciudad  Craiova/Rimnicu Vilcea/Sibiu/Arad/Zerind/ :  441  +  160 = 601
Suma de la ciudad  Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/ :  392  +  98 = 490

Ciudades visitadas ['Zerind', 'Arad', 'Oradea', 'Sibiu', 'Sibiu', 'Rimnicu Vilcea']
Nodos y distancias finales:  [{522: 'Timisoara/Arad/Zerind/'}, {492: 'Fagaras/Sibiu/Arad/Zerind/'}, {495: 'Rimnicu Vilcea/Sibiu/Ora
dea/Zerind/'}, {499: 'Fagaras/Sibiu/Oradea/Zerind/'}, {601: 'Craiova/Rimnicu Vilcea/Sibiu/Arad/Zerind/'}, {490: 'Pitesti/Rimnicu Vi
lcea/Sibiu/Arad/Zerind/'}]

-> Trayectoria con costo mínimo: 'Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/'

================================================== Paso 7  ==================================================

Suma de la ciudad  Craiova/Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/ :  530  +  160 = 690
Suma de la ciudad  Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/ :  493  +  0 = 493

Ciudades visitadas ['Zerind', 'Arad', 'Oradea', 'Sibiu', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti']
Nodos y distancias finales:  [{522: 'Timisoara/Arad/Zerind/'}, {492: 'Fagaras/Sibiu/Arad/Zerind/'}, {495: 'Rimnicu Vilcea/Sibiu/Ora
dea/Zerind/'}, {499: 'Fagaras/Sibiu/Oradea/Zerind/'}, {601: 'Craiova/Rimnicu Vilcea/Sibiu/Arad/Zerind/'}, {690: 'Craiova/Pitesti/Ri
mnicu Vilcea/Sibiu/Arad/Zerind/'}, {493: 'Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/'}]

-> Trayectoria con costo mínimo: 'Fagaras/Sibiu/Arad/Zerind/'

================================================== Paso 8  ==================================================

Suma de la ciudad  Bucharest/Fagaras/Sibiu/Arad/Zerind/ :  525  +  0 = 525

Ciudades visitadas ['Zerind', 'Arad', 'Oradea', 'Sibiu', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Fagaras']
Nodos y distancias finales:  [{522: 'Timisoara/Arad/Zerind/'}, {495: 'Rimnicu Vilcea/Sibiu/Oradea/Zerind/'}, {499: 'Fagaras/Sibiu/O
radea/Zerind/'}, {601: 'Craiova/Rimnicu Vilcea/Sibiu/Arad/Zerind/'}, {690: 'Craiova/Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/'}, {4
93: 'Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/'}, {525: 'Bucharest/Fagaras/Sibiu/Arad/Zerind/'}]

Trayectoria:  Bucharest/Pitesti/Rimnicu Vilcea/Sibiu/Arad/Zerind/
El costo total es:  493
PS C:\Users\jimen\Desktop\Inteligencia Artificial\proyecto01_IA_CJR>
```



Repositorio GitHub

Enlace:

https://github.com/Mauricio658/proyecto_IA-2024-1.git

Codigo QR: