Inteligencia Artificial
Grupo: 06
UNAM / FI / DIE
Semestre 2022-2

Proyecto 1

Método de Busqueda Uniforme: Mapa de Rumania

Garcés Gómez Eduardo Tonathiu
Quintanar Ramírez Luis Enrique
Sánchez Cano Alan

Índice

1. Objetivos

- Analizar y desarrollar el método de busqueda uniforme para el mapa de Rumania
- Realizar un programa en python que realice el método para encontar la ruta óptima de una ciudad a otra
- Indicar el avance paso por paso en el software

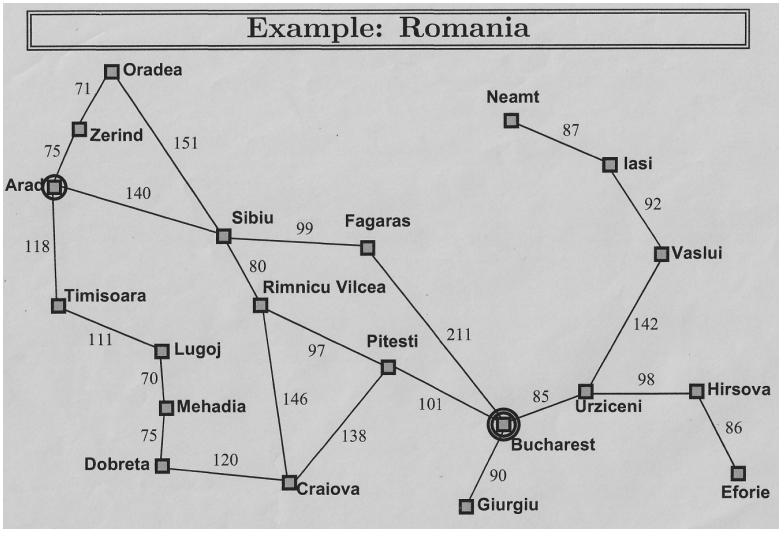


Figura 1: Mapa de Rumania en el cual nos basamos

2. Introducción

La búsqueda de costo uniforme es un algoritmo de búsqueda basado en árboles, cuya estrategia se basa en expandir los nodos de menor costo. Mediante está búsqueda se puede encontrar la solución más barata, siempre y cuando se satisfaga que el costo de la ruta no debe disminuir conforme se avanza la búsqueda
Para realizar programar el método nos basamos en el siguiente algoritmo:

1. Leemos las ciudades de Origen y Destino
2. Expandimos la ciudad de origen
3. Mientras que la ruta de menor costo no nos lleve al destino
   - Expandimos la ruta de menor costo
   - Si hay rutas que nos lleven al mismo destino, mantenemos la de menor costo y descartamos las demás
4. Cuando se encuentre la ruta de menor costo que nos lleve a la ciudad de Destino, terminamos el programa y mostramos el resultado

3. Programa

Para la solución necesitaremos los siguientres atributos: una lista de rutas que nos dará las rutas disponibles, una lista de las ciudades expandidas, y el diccionario que represente las ciudades, con sus vecinos y el costo de viaje

```
2  class Mapa:
3      # Lista de rutas posibles
4      rutas = []
5      # Lista de ciudades expandidas
6      expandidas = []
7      # Mapa de estados con sus vecinos
8      estados ={
9          'Oradea':{'Zerind':71,'Sibiu':151},
10         'Zerind':{'Arad':75,'Oradea':71},
11         'Arad':{'Zerind':75,'Sibiu':140,'Timisoara':118},
12         'Sibiu':{'Oradea':151,'Arad':140,'Fagaras':99,'Rimnicu Vilcea':80},
13         'Fagaras':{'Sibiu':99,'Bucharest':211},
14         'Timisoara':{'Arad':118,'Lugoj':111},
15         'Lugoj':{'Timisoara':111,'Mehadia':70},
16         'Mehadia':{'Lugoj':70,'Dobreta':75},
17         'Dobreta':{'Mehadia':75,'Craiova':120},
18         'Craiova':{'Dobreta':120,'Pitesti':138,'Rimnicu Vilcea':146},
19         'Pitesti':{'Rimnicu Vilcea':97,'Craiova':138,'Bucharest':101},
20         'Rimnicu Vilcea':{'Sibiu':80,'Craiova':146,'Pitesti':97},
21         'Bucharest':{'Fagaras':211,'Pitesti':101,'Giurgiu':90,'Urziceni':85},
22         'Giurgiu':{'Bucharest':90},
23         'Urziceni':{'Bucharest':85,'Hirsova':98,'Vaslui':142},
24         'Hirsova':{'Urziceni':98,'Eforie':86},
25         'Eforie':{'Hirsova':86},
26         'Vaslui':{'Urziceni':142,'Iasi':92},
27         'Iasi':{'Vaslui':92,'Neamt':87},
28         'Neamt':{'Iasi':87}
29     }
30
```

Figura 2: Atributos de la clase Mapa

Los métodos que usaremos serán:

- getMapa: nos devuelve el mapa de Rumania

- printMapa: Imprime el mapa con un formato donde muestra la ciudad, sus vecinos y el costo de viaje
- ordenaMapa: Devuelve el mapa ordenado alfabeticamente, tanto las ciudades como sus vecinos

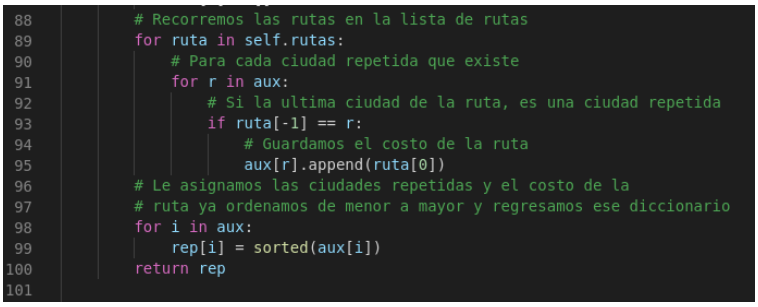
```
31      # Devuelve el diccionario de los estados
32      def getMapa(self):
33          return self.estados
34
35      # Imprime cada Estados con sus vecinos y el costo
36      def printMapa(self):
37          cad = "CIUDAD: \t \t VECINOS Y COSTOS DE VIAJE"
38          for k in self.estados:
39              cad = cad + "\n| {:<15} |".format(k)
40              for ciudad,costo in self.estados[k].items():
41                  cad = cad+"{:<15}:{:3} | ".format(ciudad,costo)
42          print(cad)
43
44      # Ordena el mapa por orden alfanumerico, tanto los estados como sus vecinos
45      def ordenaMapa(self):
46          # Variable auxiliar
47          tempEstados = self.estados
48          # Reiniciamos los estados
49          self.estados = {}
50          # Para cada ciudad en los estados ordenados
51          for p in sorted(tempEstados):
52              # Agregamos los estados
53              self.estados[p] = tempEstados[p]
54              # Borramos a sus vecinos
55              self.estados[p] = {}
56              # Para cada vecino ordenado
57              for q in sorted(tempEstados[p]):
58                  # Agregamos a los vecinos
59                  self.estados[p][q] = tempEstados[p][q]
60
```

Figura 3: Metodos: getMapa, printMapa, ordenaMapa

El método buscar repetidos nos devuelve las rutas que llegan a la misma ciudad ordenadas de menor a mayor costo. Éste método se utiliza para borrar las rutas con mayor costo de viaje

```
61      # Buscamos las rutas que lleguen a la misma ciudad
62      def buscarRepetidos(self):
63          # Variable axiliar para hacer los cambios
64          aux = {}
65          # Variable con las rutas repetidas
66          rep = {}
67          # Para cada ruta de la lista de rutas
68          for ruta in self.rutas:
69              # Guardamos la lista de ciudades al final de la ruta
70              # que no existen en aux
71              if ruta[-1] not in aux:
72                  # Cuando no existe, sólo se ha guardado una vez
73                  aux[ruta[-1]] = 1
74              else:
75                  # Si existe, aumentamos el contador en uno
76                  aux[ruta[-1]] = aux[ruta[-1]] + 1
77          # Guaramos las ciudades y la cantidad de veces que se encuentran en las rutas
78          for k,v in aux.items():
79              # Si es mayor a uno, guardamos la ciudad, debido a que se repite
80              if v > 1:
81                  rep[k] = v
82          # Reiniciamos la variable auxiliar
83          aux = {}
84          # Le asinamos al diccionario, la lista de ciudades repetidas, y le
85          # añadimos una lista que guardará el costo de cada ruta
86          for i in rep:
87              aux[i] = []
```

Figura 4: Metodos: buscarRepetidos-1

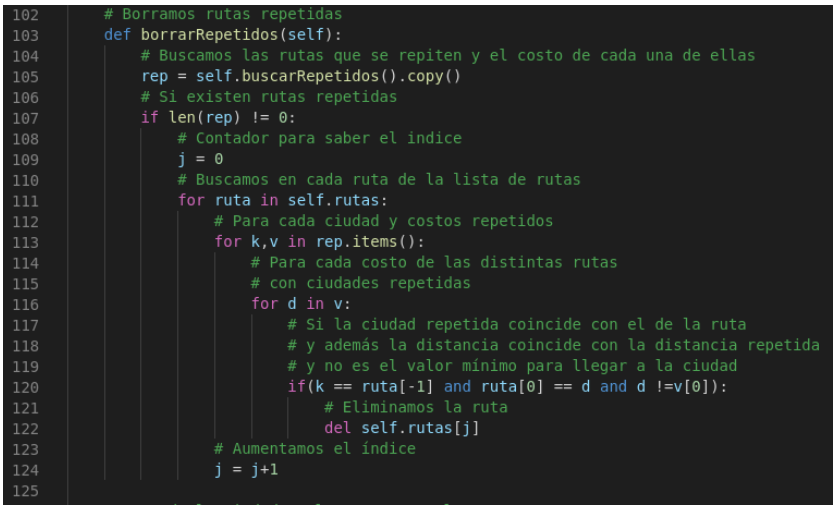


```
        # Recorremos las rutas en la lista de rutas
        for ruta in self.rutas:
            # Para cada ciudad repetida que existe
            for r in aux:
                # Si la ultima ciudad de la ruta, es una ciudad repetida
                if ruta[-1] == r:
                    # Guardamos el costo de la ruta
                    aux[r].append(ruta[0])
        # Le asignamos las ciudades repetidas y el costo de la
        # ruta ya ordenamos de menor a mayor y regresamos ese diccionario
        for i in aux:
            rep[i] = sorted(aux[i])
        return rep

```

Figura 5: Metodos: buscarRepetidos-2

El método borrarRepetidos borra cada una de las rutas, en la lista de rutas, que lleven al mismo destino, a excepción del de menor costo de viaje



```
    # Borramos rutas repetidas
    def borrarRepetidos(self):
        # Buscamos las rutas que se repiten y el costo de cada una de ellas
        rep = self.buscarRepetidos().copy()
        # Si existen rutas repetidas
        if len(rep) != 0:
            # Contador para saber el indice
            j = 0
            # Buscamos en cada ruta de la lista de rutas
            for ruta in self.rutas:
                # Para cada ciudad y costos repetidos
                for k,v in rep.items():
                    # Para cada costo de las distintas rutas
                    # con ciudades repetidas
                    for d in v:
                        # Si la ciudad repetida coincide con el de la ruta
                        # y además la distancia coincide con la distancia repetida
                        # y no es el valor mínimo para llegar a la ciudad
                        if(k == ruta[-1] and ruta[0] == d and d !=v[0]):
                            # Eliminamos la ruta
                            del self.rutas[j]
                # Aumentamos el indice
                j = j+1

```

Figura 6: Metodos: borrarRepetidos

El método expandirCiudad expande la ruta que recibe como argumento, y las agrega a la lista de rutas

```python
    # Expande la ciudad en las rutas que la contengan
    def expandirCiudad(self, ciudad):
        # Guardará los indices donde se encuentra la ciudad expandida
        indices = []
        # Recorremos todas las ciudades en las rutas
        for cdad in range(len(self.rutas)):
            # Si se encuentra la ciudad a expandir
            if ciudad in self.rutas[cdad] :
                # Guardamos la lista para agregar las expansiones
                lst = self.rutas[cdad]
                # El indice donde se encuentra la ciudad se guarda
                indices.append(cdad)
                # Para cada vecino de la ciudad a expandir
                for vecino in self.getMapa()[ciudad].keys():
                    # Si el vecino no se ha expandido
                    if vecino not in self.expandidas:
                        ls = []
                        ls = ls + lst
                        # Agregamos al vecino en la ruta a expandir
                        ls.append(vecino)
                        # Lo añadimos a la lista de rutas
                        self.rutas.append(ls)
                        # Sumamos el costo de viaje a la ciudad agregada
                        self.rutas[-1][0] = self.rutas[-1][0] + self.getMapa()[ciudad].get(vecino)
        # Borramos las rutas de las ciudades antes de ser expandidas
        for i in indices:
            del self.rutas[i]
        self.expandidas.append(ciudad)

```

Figura 7: Metodos: expandirCiudad

El método buscarRutaMenor nos devuelve la siguiente ciudad a expandir, es decir ruta de menor costo que existe en la lista de rutas. Mientras que el método leerCiudades, recibe la entrada de las ciudades de Origen y de Destino

```python
    def buscaRutaMenor(self):
        # Costos de cada ruta en las rutas
        costos = []
        # Para cada ruta en las rutas
        for ruta in self.rutas:
            # Guardamos los costos de cada una
            costos.append(ruta[0])
        # El minimo es el primero de la lista ordenada
        min = sorted(costos)[0]
        # Buscamos en cada ruta de la ruta
        for ruta in self.rutas:
            # Si el costo de la ruta es el mínimo
            if ruta[0] == min:
                # Regresamos la siguiente ciudad a expandir
                return ruta[-1]
```

Figura 8: Metodos: buscarRutaMenor

```python
    # Lee las ciudades de origen y de destino
    def leerCiudades(self):
        # Leemos la ciudad de origen
        self.ciudadOrigen = input('IINGRESE LA CIUDAD DE ORIGEN: ')
        while (self.ciudadOrigen not in self.getMapa()):
            print('CIUDAD DE ORIGEN NO ENCONTRADA')
            self.ciudadOrigen = input('INGRESE UNA CIUDAD DE ORIGEN VÁLIDA: ')
        ls=[0]
        ls.append(self.ciudadOrigen)
        self.rutas.append(ls)

        # Leemos la ciudad de destino
        self.ciudadDestino = input('INGRESE LA CIUDAD DE DESTINO: ')
        while (self.ciudadDestino not in self.getMapa() or self.ciudadDestino == self.ciudadOrigen):
            print('CIUDAD DE DESTINO NO ENCONTRADA')
            self.ciudadDestino = input('INGRESE UNA CIUDAD DE DESTINO VÁLIDA: ')

```

Figura 9: Metodos: leerCiudades

El método imprimeDatos, muestra las ciudades expandidas y cada una de las rutas paso a paso

```
189     # Imprime ciudades expandidas y lista de rutas
190     def imprimeDatos(self):
191         exp = "CIUDADES EXPANDIDAS:\n \t "
192         # Para cada ciudad expandida
193         for i in self.expandidas:
194             # Concatenamos la ciudad para darle formato
195             exp = exp + i + ", "
196 
197         rutas = "RUTAS: \n \t"
198         # Para cada ruta de la lista de rutas
199         for i in self.rutas:
200             # Agregamos el costo de la ruta
201             rutas = rutas + str(i[0])
202             # Para cada ciudad de la ruta
203             for j in range(len(i)-1):
204                 # Le damos formato para imprimir
205                 rutas = rutas + "->" + str(i[j+1])
206             rutas = rutas + "\n\t"
207         # Imprimimos las ciudades expandidad
208         print(exp)
209         # Imprimimos las rutas de las ciudades
210         print(rutas)
211 
```

Figura 10: Metodos: imprimeDatos

El método BCU contiene el algoritmo planteado en la introducción para realizar la búsqueda por el método de costo uniforme

```
212     # Busqueda de Costo Uniforme
213     def BCU(self):
214         # Leemos las ciudades de Origen y Desino
215         self.leerCiudades()
216         # Imprimimos la ciudad de origen y el costo
217         self.imprimeDatos()
218         # Expandimos la ciudad de origen
219         self.expandirCiudad(self.ciudadOrigen)
220         # Imprimimos los datos nuevos
221         self.imprimeDatos()
222         # Mientras que la ruta de menor costo no nos lleve a la ciudad de Destino
223         while self.buscaRutaMenor() != self.ciudadDestino:
224             # Expandimos la ciudad de menor costo
225             self.expandirCiudad(self.buscaRutaMenor())
226             # Imprimimos los nuevos dato
227             self.imprimeDatos()
228             # Borramos las rutas que lleguen a la misma ciudad y que
229             # impliquen un mayor costo
230             self.borrarRepetidos()
231             # Si la ruta de menor costo nos lleva al destino imprimimos un mensaje de éxito
232             if self.buscaRutaMenor() == self.ciudadDestino:
233                 print("RUTA MINIMA ENCONTRADA\n")
234                 for rutas in self.rutas:
235                     if rutas[1] == self.ciudadOrigen and rutas[-1] == self.ciudadDestino:
236                         print("\tCOSTO: "+str(rutas[0]))
237                         stri = "\tRUTA "
238                         for r in range(len(rutas)-1):
239                             stri = stri +"-> " + rutas[r+1]
240                 print(stri)
```

Figura 11: Metodos: BCU

El constructor nos ordena el mapa alfabeticamente cada que se instancia un objeto. Y cuando se llama al main, instacia un objeto mapa, imprime el mapa y realiza el método de búsqueda

```
241     # Constructor. Cuando se instancia un objeto, ordena el mapa
242     def __init__(self):
243         self.ordenaMapa()
244
245 if __name__ == "__main__":
246     # Creamos un mapa
247     e = Mapa()
248     # Imprimimos el mapa
249     e.printMapa()
250     # Iniciamos el algoritmo de Búsqueda de Costo Uniforme
251     e.BCU()
252
```

Figura 12: Metodos: constructor, main

4. Uso del Software

El software fue probado y diseñado en equipos con las siguientes características.

- Sistema Operativo Linux
- Sistema Operativo Windows
- Python 3, aunque puede funcionar en versiones anteriores de python

Para correr el software, en la carpeta donde se encuentra descargado, usamos el siguiente comando en la terminal:
$ python3 proyecto1.py
Y nos mostrará las ciudades y sus vecinos con el costo de viaje de la siguiente manera:

```
luis@debian:~/Descargas/IA/Proyecto1/Proyecto1-IA$ python3 proyecto1.py
CIUDAD:                 VECINOS Y COSTOS DE VIAJE
| Arad            |Sibiu          :140 | Timisoara      :118 | Zerind         : 75 |
| Bucharest       |Fagaras        :211 | Giurgiu        : 90 | Pitesti        :101 | Urziceni       : 85 |
| Craiova         |Dobreta        :120 | Pitesti        :138 | Rimnicu Vilcea :146 |
| Dobreta         |Craiova        :120 | Mehadia        : 75 |
| Eforie          |Hirsova        : 86 |
| Fagaras         |Bucharest      :211 | Sibiu          : 99 |
| Giurgiu         |Bucharest      : 90 |
| Hirsova         |Eforie         : 86 | Urziceni       : 98 |
| Iasi            |Neamt          : 87 | Vaslui         : 92 |
| Lugoj           |Mehadia        : 70 | Timisoara      :111 |
| Mehadia         |Dobreta        : 75 | Lugoj          : 70 |
| Neamt           |Iasi           : 87 |
| Oradea          |Sibiu          :151 | Zerind         : 71 |
| Pitesti         |Bucharest      :101 | Craiova        :138 | Rimnicu Vilcea : 97 |
| Rimnicu Vilcea  |Craiova        :146 | Pitesti        : 97 | Sibiu          : 80 |
| Sibiu           |Arad           :140 | Fagaras        : 99 | Oradea         :151 | Rimnicu Vilcea : 80 |
| Timisoara       |Arad           :118 | Lugoj          :111 |
| Urziceni        |Bucharest      : 85 | Hirsova        : 98 | Vaslui         :142 |
| Vaslui          |Iasi           : 92 | Urziceni       :142 |
| Zerind          |Arad           : 75 | Oradea         : 71 |
IINGRESE LA CIUDAD DE ORIGEN:
```

Figura 13: Muestra ciudades y sus vecinos con el costo de viaje

Ingresaremos la ciudad de origen y la de destino

```
luis@debian:~/Descargas/IA/Proyecto1/Proyecto1-IA$ python3 proyecto1.py
CIUDAD:                 VECINOS Y COSTOS DE VIAJE
| Arad            |Sibiu          :140 | Timisoara      :118 | Zerind         : 75 |
| Bucharest       |Fagaras        :211 | Giurgiu        : 90 | Pitesti        :101 | Urziceni       : 85 |
| Craiova         |Dobreta        :120 | Pitesti        :138 | Rimnicu Vilcea :146 |
| Dobreta         |Craiova        :120 | Mehadia        : 75 |
| Eforie          |Hirsova        : 86 |
| Fagaras         |Bucharest      :211 | Sibiu          : 99 |
| Giurgiu         |Bucharest      : 90 |
| Hirsova         |Eforie         : 86 | Urziceni       : 98 |
| Iasi            |Neamt          : 87 | Vaslui         : 92 |
| Lugoj           |Mehadia        : 70 | Timisoara      :111 |
| Mehadia         |Dobreta        : 75 | Lugoj          : 70 |
| Neamt           |Iasi           : 87 |
| Oradea          |Sibiu          :151 | Zerind         : 71 |
| Pitesti         |Bucharest      :101 | Craiova        :138 | Rimnicu Vilcea : 97 |
| Rimnicu Vilcea  |Craiova        :146 | Pitesti        : 97 | Sibiu          : 80 |
| Sibiu           |Arad           :140 | Fagaras        : 99 | Oradea         :151 | Rimnicu Vilcea : 80 |
| Timisoara       |Arad           :118 | Lugoj          :111 |
| Urziceni        |Bucharest      : 85 | Hirsova        : 98 | Vaslui         :142 |
| Vaslui          |Iasi           : 92 | Urziceni       :142 |
| Zerind          |Arad           : 75 | Oradea         : 71 |
IINGRESE LA CIUDAD DE ORIGEN: Arad
INGRESE LA CIUDAD DE DESTINO: Bucharest
```

Figura 14: Ingreso de las ciudades Arad-Bucharest

Una vez que ingresemos las ciudades nos desplegará paso a paso las ciudades que fue expandiendo hasta encontrar la ruta optima. Para mostrar el mensaje de exito con el resultado

```
CIUDADES EXPANDIDAS:

RUTAS:
        0->Arad

CIUDADES EXPANDIDAS:
         Arad,
RUTAS:
        140->Arad->Sibiu
        118->Arad->Timisoara
        75->Arad->Zerind

CIUDADES EXPANDIDAS:
         Arad, Zerind,
RUTAS:
        140->Arad->Sibiu
        118->Arad->Timisoara
        146->Arad->Zerind->Oradea

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara,
RUTAS:
        140->Arad->Sibiu
        146->Arad->Zerind->Oradea
        229->Arad->Timisoara->Lugoj

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu,
RUTAS:
        146->Arad->Zerind->Oradea
        229->Arad->Timisoara->Lugoj
        239->Arad->Sibiu->Fagaras
        291->Arad->Sibiu->Oradea
        220->Arad->Sibiu->Rimnicu Vilcea
```

Figura 15: Muestra paso a paso de la solucion - 1

```
CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea,
RUTAS:
        229->Arad->Timisoara->Lugoj
        239->Arad->Sibiu->Fagaras
        220->Arad->Sibiu->Rimnicu Vilcea

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea,
RUTAS:
        229->Arad->Timisoara->Lugoj
        239->Arad->Sibiu->Fagaras
        366->Arad->Sibiu->Rimnicu Vilcea->Craiova
        317->Arad->Sibiu->Rimnicu Vilcea->Pitesti

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj,
RUTAS:
        239->Arad->Sibiu->Fagaras
        366->Arad->Sibiu->Rimnicu Vilcea->Craiova
        317->Arad->Sibiu->Rimnicu Vilcea->Pitesti
        299->Arad->Timisoara->Lugoj->Mehadia

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj, Fagaras,
RUTAS:
        366->Arad->Sibiu->Rimnicu Vilcea->Craiova
        317->Arad->Sibiu->Rimnicu Vilcea->Pitesti
        299->Arad->Timisoara->Lugoj->Mehadia
        450->Arad->Sibiu->Fagaras->Bucharest

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj, Fagaras, Mehadia,
RUTAS:
        366->Arad->Sibiu->Rimnicu Vilcea->Craiova
        317->Arad->Sibiu->Rimnicu Vilcea->Pitesti
        450->Arad->Sibiu->Fagaras->Bucharest
        374->Arad->Timisoara->Lugoj->Mehadia->Dobreta
```

Figura 16: Muestra paso a paso de la solucion - 2

```
CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj, Fagaras, Mehadia,
RUTAS:
        366->Arad->Sibiu->Rimnicu Vilcea->Craiova
        317->Arad->Sibiu->Rimnicu Vilcea->Pitesti
        450->Arad->Sibiu->Fagaras->Bucharest
        374->Arad->Timisoara->Lugoj->Mehadia->Dobreta

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj, Fagaras, Mehadia, Pitesti,
RUTAS:
        366->Arad->Sibiu->Rimnicu Vilcea->Craiova
        450->Arad->Sibiu->Fagaras->Bucharest
        374->Arad->Timisoara->Lugoj->Mehadia->Dobreta
        418->Arad->Sibiu->Rimnicu Vilcea->Pitesti->Bucharest
        455->Arad->Sibiu->Rimnicu Vilcea->Pitesti->Craiova

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Craiova,
RUTAS:
        374->Arad->Timisoara->Lugoj->Mehadia->Dobreta
        418->Arad->Sibiu->Rimnicu Vilcea->Pitesti->Bucharest
        486->Arad->Sibiu->Rimnicu Vilcea->Craiova->Dobreta

CIUDADES EXPANDIDAS:
         Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Craiova, Dobreta,
RUTAS:
        418->Arad->Sibiu->Rimnicu Vilcea->Pitesti->Bucharest

RUTA MÍNIMA ENCONTRADA

        COSTO: 418
        RUTA -> Arad-> Sibiu-> Rimnicu Vilcea-> Pitesti-> Bucharest
```

Figura 17: Muestra paso a paso de la solucion - 3