Inteligencia Artificial.

Grupo 6.

Dr. Herrera Camacho José Abel.

Proyecto 1

Método de búsqueda uniforme.

Arriaga Mejía José Carlos

González Del Moral Ángel

Martínez Vázquez Kevin Alexander

2022-2

21/04/22

Introducción.

En este programa se buscará la trayectoria menos costosa para llegar de una ciudad a otra con el método uniforme.

Algoritmo empleado.

1) Identificamos la ciudad inicial, la cual será la primera ciudad actual, y la ciudad destino.

2) Se expanden las ciudades, que se pueden visitar desde la ciudad actual, y se coloca el costo para llegar a ellas. La ciudad expandida se marca como cuidad visitada.

3) Se busca el menor costo de todos los nodos expandidos y se marca como ciudad actual a la que tiene el menor costo. Si la ciudad marcada como actual ya fue una ciudad visitada se repite el paso 3) y se manda a tierra esa trayectoria.

4) Se comprueba si la ciudad actual es la ciudad destino, sino repetir paso 2).

Manual de Usuario.

El programa le solicitara al usuario la ciudad de donde quiere partir y la ciudad a donde quiere llegar, el usuario deberá ingresar las 3 primeras letras del nombre de las ciudades para que el programa funcione correctamente, si las ciudades no se encuentran en el mapa de Rumania el programa se lo indicara y se le preguntara sí quiere calcular otra trayectoria, para esto tendrá que contestar si o no, no importa si es mayúsculas, minúsculas o la combinación de ambos, el programa lo detectara, siempre y cuando sea si o no. El programa ira imprimiendo paso por paso las trayectorias que va calculando con el método, cuando encuentra la ciudad actual, terminará la impresión de los pasos e imprimirá la trayectoria final y el costo que esta tiene. Al usuario se le preguntara si quiere calcular otra trayectoria, para esto tendrá que contestar si o no, no importa si es mayúsculas, minúsculas o la combinación de ambos, el programa lo detectara, siempre y cuando sea si o no.

Código en Python.

```
1   import copy
2
3   #Declaración del mapa de Rumania con sus ciudades colindantes
4   #y su costo de traslación
5
6   Rumania = [["Ara","Sib", 140],["Ara","Zer", 75],["Ara","Tim", 118],
7              ["Sib","Fag", 99],["Sib","Rim", 80],["Zer","Ora", 71],
8              ["Ora","Sib", 151],["Tim","Lug", 111],["Lug","Meh", 70],
9              ["Meh","Dob",75],["Dob","Cra",120],["Rim","Cra",146],
10             ["Rim","Pit",97],["Fag","Buc",211],["Pit","Buc",101],
11             ["Pit","Cra",138],["Buc","Giu",90],["Buc","Urz",85],
12             ["Urz","Vas",142],["Urz","Hir",98],["Vas","Ias",92],
13             ["Hir","Efo",86],["Ias","Nea",87]]
14
15  #Lista que guarda los recorridos (trayectorias), que puede tomar desde
16  #un nodo inicial
17  recorridos=list()
18
19  #Lista de ciudades expandidas
20  ciudadesVisitadas = list()
```

```python
# Función distanciaTotal
#Funcion que devuelve la distancia total
#de la trayectoria actual

def distanciaTotal (distancia):
    return distancia[1]

# Función existe
# Función que comprueba la integridad de las entradas
# se asegura de que las ciudades de inicio y fin ingresadas
# se encuentren en el mapa de Rumania

def existe(ciudad):
    for i in range (len(Rumania)):
        for j in range (2):
            if(ciudad.casefold() == str(Rumania[i][j]).casefold()):
                return Rumania[i][j]
    return None
```

```python
# Función divNodo
# Función que divide el nodo actual (ciudad actual)
# para poder crear las diferentes trayectorias a sus
# ciudades colindantes. (El numero de copias depende
# de la cantidad de ciudades colindantes)

def divNodo(nodo_div,num_div):
    global recorridos

    for i in range (num_div-1):
        recorridos.append(copy.deepcopy(nodo_div))
```

```python
# Función insertar recorridos
# Esta funcion inserta a una lista todas las posibles
# trayectorias desde el nodo actual (ciudad actual).

def insertarRecorrido(nodoActual,ciudadesSiguientes):
    global recorridos

    if(len(recorridos)== 0):
        tmp2=list()
        tmp2.append(nodoActual)
        recorridos.append([tmp2,0])

    banderaPrimero = True
    nodo_div = list()
    cantidadrecorridos= len(recorridos)
    i = 0
    j = 0
    while j < cantidadrecorridos:
        if(recorridos[j][0][-1]==nodoActual):

            if(banderaPrimero):
                nodo_div = recorridos[j]
                divNodo(nodo_div,len(ciudadesSiguientes))
                cantidadrecorridos = len(recorridos)
                banderaPrimero = False

            recorridos[j][0].append(ciudadesSiguientes[i][0])
            recorridos[j][1]+=ciudadesSiguientes[i][1]
            i+=1
        j+=1
```

```python
# Función posiblesCiudades
# Función que verifica para el nodo actual (ciudad actual), cuales
# son los diferentes nodos a los que se puede ir (Siguientes ciudades)

def posiblesCiudades (nodoActual):
    destinos = list()

    for i in range (len(Rumania)):
        for j in range (2):
            if(nodoActual == Rumania[i][j]):
                if(j == 0 and Rumania[i][1] not in ciudadesVisitadas):
                    destinos.append([Rumania[i][1],Rumania[i][2]])
                elif (j==1 and Rumania[i][0] not in ciudadesVisitadas):
                    destinos.append([Rumania[i][0],Rumania[i][2]])
    return destinos

# Función Expandir
# Esta función expande el nodo e inserta en la lista que
# guarda las trayectorias todas las posibles trayectorias
# a las que podamos trasladarnos

def expandir(nodoActual):

    global ciudadesVisitadas,recorridos

    ciudadesSiguientes = posiblesCiudades(nodoActual)
    ciudadesVisitadas.append(nodoActual)

    if(len(ciudadesSiguientes) == 0):
        recorridos.pop(0)
    else:
        insertarRecorrido(nodoActual,ciudadesSiguientes)

    recorridos.sort(key = distanciaTotal)
    Tierra()

    return recorridos[0][0][-1]
```

```
126
127   # Función Tierra
128   #Función cuyo proposito es eliminar de la lista de trayectorias
129   #las ciudades ya visitadas de mayor costo
130
131   def Tierra ():
132       global recorridos
133
134       for x in recorridos:
135           for y in recorridos:
136               if(x[0][-1] == y[0][-1] and x[1] < y[1]):
137                   recorridos.remove(y)
138
```

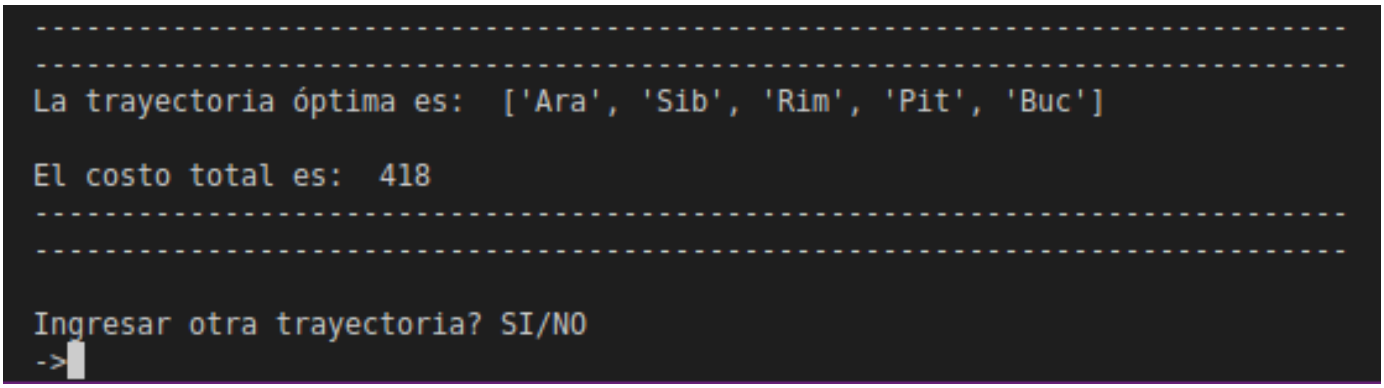
```
139
140   # Función Main
141   #Función donde se inicializa el proceso de busqueda
142   #a partir de la introducción de los datos iniciales
143   #y la llamada a las funciones que realizan la busqueda
144
145   def main():
146       ciudadInicio = existe (input("Ciudad de inicio:\n->"))
147       ciudadFin = existe(input("Ciudad de fin:\n->"))
148       x = 1
149       if(ciudadInicio!= None and ciudadFin != None):
150           ciudadActual=ciudadInicio
151
152           while(ciudadActual!=ciudadFin):
153               print("\nPaso: ",x)
154               ciudadActual = expandir(ciudadActual)
155               print(recorridos)
156               print("\n\n")
157               x+=1
158           print("--------------------------------------------------------------------------------")
159           print("--------------------------------------------------------------------------------")
160           print("La trayectoria óptima es: ", recorridos[0][0])
161           print("\nEl costo total es: ", recorridos[0][1])
162           print("--------------------------------------------------------------------------------")
163           print("--------------------------------------------------------------------------------")
164
165       else:
166           print("Las ciudades ingresadas no se encuentran en el mapa")
167
```

```
169
170   #Bloque de codigo donde se ejecuta la función main
171   #Este bloque se encuentra dentro de una especie de ciclo
172   #"DO-WHILE" con el motivo de poder realizar multiples
173   #busquedas en una sola ejecución del programa
174   while True:
175       main()
176       opcion = input("\nIngresar otra trayectoria? SI/NO\n->").lower();
177       recorridos.clear()
178       ciudadesVisitadas.clear()
179
180       if opcion == "no":
181           break
```

Ejemplos utilizando el programa.

Ej. 1 de Arad a Bucharest.

```
Ciudad de inicio:
->ara
Ciudad de fin:
->buc

Paso:  1
[[['Ara', 'Zer'], 75], [['Ara', 'Tim'], 118], [['Ara', 'Sib'], 140]]

Paso:  2
[[['Ara', 'Tim'], 118], [['Ara', 'Sib'], 140], [['Ara', 'Zer', 'Ora'], 146]]

Paso:  3
[[['Ara', 'Sib'], 140], [['Ara', 'Zer', 'Ora'], 146], [['Ara', 'Tim', 'Lug'], 229]]

Paso:  4
[[['Ara', 'Zer', 'Ora'], 146], [['Ara', 'Sib', 'Rim'], 220], [['Ara', 'Tim', 'Lug'], 229], [['Ara', 'Sib', 'Fag'], 239]]

Paso:  5
[[['Ara', 'Sib', 'Rim'], 220], [['Ara', 'Tim', 'Lug'], 229], [['Ara', 'Sib', 'Fag'], 239]]

Paso:  6
[[['Ara', 'Tim', 'Lug'], 229], [['Ara', 'Sib', 'Fag'], 239], [['Ara', 'Sib', 'Rim', 'Pit'], 317], [['Ara', 'Sib', 'Rim', 'Cra'], 366]]

Paso:  7
[[['Ara', 'Sib', 'Fag'], 239], [['Ara', 'Tim', 'Lug', 'Meh'], 299], [['Ara', 'Sib', 'Rim', 'Pit'], 317], [['Ara', 'Sib', 'Rim', 'Cra'], 366]]

Paso:  8
[[['Ara', 'Tim', 'Lug', 'Meh'], 299], [['Ara', 'Sib', 'Rim', 'Pit'], 317], [['Ara', 'Sib', 'Rim', 'Cra'], 366], [['Ara', 'Sib', 'Fag', 'Buc'], 450]]

Paso:  9
[[['Ara', 'Sib', 'Rim', 'Pit'], 317], [['Ara', 'Sib', 'Rim', 'Cra'], 366], [['Ara', 'Tim', 'Lug', 'Meh', 'Dob'], 374], [['Ara', 'Sib', 'Fag', 'Buc'], 450]]

Paso:  10
[[['Ara', 'Sib', 'Rim', 'Cra'], 366], [['Ara', 'Tim', 'Lug', 'Meh', 'Dob'], 374], [['Ara', 'Sib', 'Rim', 'Pit', 'Buc'], 418]]

Paso:  11
[[['Ara', 'Tim', 'Lug', 'Meh', 'Dob'], 374], [['Ara', 'Sib', 'Rim', 'Pit', 'Buc'], 418]]

Paso:  12
[[['Ara', 'Sib', 'Rim', 'Pit', 'Buc'], 418]]
--------------------------------------------------------------------------------
```



Ej. 2 de Mehadia a Hirvosa.

```
Ingresar otra trayectoria? SI/NO
->si
Ciudad de inicio:
->meh
Ciudad de fin:
->hir

Paso:  1
[[['Meh', 'Lug'], 70], [['Meh', 'Dob'], 75]]

Paso:  2
[[['Meh', 'Dob'], 75], [['Meh', 'Lug', 'Tim'], 181]]

Paso:  3
[[['Meh', 'Lug', 'Tim'], 181], [['Meh', 'Dob', 'Cra'], 195]]

Paso:  4
[[['Meh', 'Dob', 'Cra'], 195], [['Meh', 'Lug', 'Tim', 'Ara'], 299]]

Paso:  5
[[['Meh', 'Lug', 'Tim', 'Ara'], 299], [['Meh', 'Dob', 'Cra', 'Pit'], 333], [['Meh', 'Dob', 'Cra', 'Rim'], 341]]

Paso:  6
[[['Meh', 'Dob', 'Cra', 'Pit'], 333], [['Meh', 'Dob', 'Cra', 'Rim'], 341], [['Meh', 'Lug', 'Tim', 'Ara', 'Zer'], 374], [['Meh', 'Lug', 'Tim', 'Ara', 'Sib'], 439]]

Paso:  7
[[['Meh', 'Dob', 'Cra', 'Rim'], 341], [['Meh', 'Lug', 'Tim', 'Ara', 'Zer'], 374], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc'], 434], [['Meh', 'Lug', 'Tim', 'Ara', 'Sib'], 439]]

Paso:  8
[[['Meh', 'Lug', 'Tim', 'Ara', 'Zer'], 374], [['Meh', 'Dob', 'Cra', 'Rim', 'Sib'], 421], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc'], 434]]

Paso:  9
[[['Meh', 'Dob', 'Cra', 'Rim', 'Sib'], 421], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc'], 434], [['Meh', 'Lug', 'Tim', 'Ara', 'Zer', 'Ora'], 445]]

Paso:  10
[[['Meh', 'Dob', 'Cra', 'Pit', 'Buc'], 434], [['Meh', 'Lug', 'Tim', 'Ara', 'Zer', 'Ora'], 445], [['Meh', 'Dob', 'Cra', 'Rim', 'Sib', 'Fag'], 520]]

Paso:  11
[[['Meh', 'Lug', 'Tim', 'Ara', 'Zer', 'Ora'], 445], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz'], 519], [['Meh', 'Dob', 'Cra', 'Rim', 'Sib', 'Fag'], 520], [['Meh', 'Dob', 'Cra', 'Pit',
'Buc', 'Giu'], 524]]

Paso:  12
[[['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz'], 519], [['Meh', 'Dob', 'Cra', 'Rim', 'Sib', 'Fag'], 520], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Giu'], 524]]

Paso:  13
[[['Meh', 'Dob', 'Cra', 'Rim', 'Sib', 'Fag'], 520], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Giu'], 524], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz', 'Hir'], 617], [['Meh', 'Dob', 'Cra',
'Pit', 'Buc', 'Urz', 'Vas'], 661]]

Paso:  14
[[['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Giu'], 524], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz', 'Hir'], 617], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz', 'Vas'], 661]]

Paso:  15
[[['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz', 'Hir'], 617], [['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz', 'Vas'], 661]]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
La trayectoria óptima es:  ['Meh', 'Dob', 'Cra', 'Pit', 'Buc', 'Urz', 'Hir']

El costo total es:  617
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

Ingresar otra trayectoria? SI/NO
->
```