

In [9]:



```
1 from neo4j import GraphDatabase
2 import bs4
3 from bs4 import BeautifulSoup as soup
4 from urllib.request import Request, urlopen
5 from termcolor import colored
6 #pip install termcolor
```

In [10]:



```

1  #CLASE PAR CREAR NODO CENTRAR-PARQUE CENTRAL
2  class CLASE_NEO4J(object):
3      def __init__(self):
4          self._driver = GraphDatabase.driver("bolt:neo4j://localhost:7687", auth=("neo4j", "password"))
5      def close(self):
6          self._driver.close()
7      def CREAR_LUGAR(self, message, lugar, latitud, longitud):
8          with self._driver.session() as session:
9              greeting = session.write_transaction(self._VALIDAR_LUGAR, message, lugar, latitud, longitud)
10             print(greeting)
11      def CREAR_RUTA(self, origen, destino, costo, hn):
12          with self._driver.session() as session:
13              greeting2 = session.write_transaction(self._VALIDAR_RUTA, origen, destino, costo, hn)
14              print(greeting2)
15      def CREAR_LUGAR_KNN(self, message, lugar, latitud, longitud, estrellas, infectados, tipo):
16          with self._driver.session() as session:
17              greeting = session.write_transaction(self._VALIDAR_LUGAR_KNN, message, lugar, latitud, longitud, estrellas, infectados, tipo)
18              print(greeting)
19
20      #METODO PARA CREAR LOS NODOS DE LUGARES PARA ALGORITMO COSTO
21      @staticmethod
22      def _VALIDAR_LUGAR(tx, message, lugar, latitud, longitud):
23          #SE BUSCA SI EL LUGAR DEL ARREGLO EXISTE EN LA BASE NEO4J
24          result2 = tx.run("match(1:Lugares {nombre:'"+lugar+"',latitud:'"+latitud+"',longitud:'"+longitud+"'})")
25          #CONDICION PARA VERIFICAR SI EXISTE
26          if int(len(result2)) == 0:
27              print("SE CREA EL LUGAR EN LA BASE.....")
28              #SE CREA NODO LUGAR
29              result = tx.run("CREATE("+lugar+":Lugares {nombre:'"+lugar+"',latitud:'"+latitud+"',longitud:'"+longitud+"'})")
30              "SET "+lugar+".message = $message "
31              "RETURN "+lugar+".message + ', from node ' + id("+lugar+")", message)
32          elif int(len(result2)) == 1:
33              print("EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....")
34
35      #METODO PARA CREAR LOS NODOS DE LUGARES PARA EL ALGORITMO KNN
36      @staticmethod
37      def _VALIDAR_LUGAR_KNN(tx, message, lugar, latitud, longitud, estrellas, infectados, tipo):
38          #SE BUSCA SI EL LUGAR DEL ARREGLO EXISTE EN LA BASE NEO4J
39          result2 = tx.run("match(1:"+tipo+" {nombre:'"+lugar+"',latitud:'"+latitud+"',longitud:'"+longitud+"'})")
40          #CONDICION PARA VERIFICAR SI EXISTE
41          if int(len(result2)) == 0:
42              print("SE CREA EL LUGAR EN LA BASE.....")
43              #SE CREA NODO LUGAR
44              result = tx.run("CREATE("+lugar+": "+tipo+" {nombre:'"+lugar+"',latitud:'"+latitud+"',longitud:'"+longitud+"'})")
45              "SET "+lugar+".message = $message "
46              "RETURN "+lugar+".message + ', from node ' + id("+lugar+")", message)
47          elif int(len(result2)) == 1:
48              print("EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....")
49
50      #METODO PARA CREAR LAS RELACIONES CON EL COSTE Y HN PARA LA RUTA
51      @staticmethod
52      def _VALIDAR_RUTA(tx, origen, destino, costo, hn):
53          #SE BUSCA SI LA RUTA A CREAR YA DEL ARREGLO EXISTE EN LA BASE NEO4J
54          result = tx.run("match(1:Lugares {nombre:'"+origen+"'})-[r:RUTA_DE {costo:'"+costo+"',hn:'"+hn+"'}]-(2:Lugares {nombre:'"+destino+"'})")
55          if int(len(result)) == 0:
56              print("SE CREA LOS NODOS DE RELACION DE RUTAS ENTRE LOS LUGARES .....")
57              result2 = tx.run(" match('"+origen+":Lugares {nombre:'"+origen+"'}) match('"+destino+":Lugares {nombre:'"+destino+"'})")
58          elif int(len(result)) == 1:
59              print(colored("YA EXISTE LA RUTA*****", 'green'))

```

```

60 # MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n,r
61 #SE INICIALIZA LA CLASE DE LOS METODOS DE NEO4J
62 grafo=CLASE_NEO4J()

```

```

2 class CLASE_NEO4J(object):
3     def __init__(self):
----> 4         self._driver = GraphDatabase.driver("bolt:neo4j://localhost:7687", auth=("neo4j", "neo4j"), encrypted=False)
5         def close(self):
6             self._driver.close()

```

```

~\anaconda3\lib\site-packages\neo4j\__init__.py in driver(cls, uri, auth,
**config)
181
182     if driver_type == DRIVER_BOLT:
--> 183         return cls.bolt_driver(parsed.netloc, auth=auth, **con
fig)
184     elif driver_type == DRIVER_NEO4j:
185         routing_context = parse_routing_context(parsed.query)

```

```

~\anaconda3\lib\site-packages\neo4j\__init__.py in bolt_driver(cls, target,
auth, **config)
194
195     +...

```

In [8]:



```

1 #SE CREA LA LISTA DE NODOS LUGARES
2 listal = ([ "HOTEL_MONTECARLO", -79.16885, -09.17885, 2, 10],
3           [ "PARQUE_CENTRAL", -0.22214, -0.34216, 3, 20],
4           [ "RESTAURANTE_CODELL", -0.2975, -79.16784, 4, 11],
5           [ "RESTAURANTE_PICHINCHA", -0.24665, -79.19275, 1, 9],
6           [ "RESTAURANTE_CHURRERIA", -0.2975, -79.1765, 2, 2],
7           [ "RESTAURANTE_QUINTA_DORADA", -0.23421, -79.16772, 5, 23],
8           [ "RESTAURANTE_CORBAN_COFFEE", -0.27227, -79.1402, 3, 15],
9           [ "RESTAURANTE_TENTACIONES", -0.25516, -79.18057, 2, 16],
10          [ "RESTAURANTE_DELIRIO", -0.25487, -79.1854, 4, 20],
11          [ "HOTEL_IMPERIAL", -0.24886, -79.15787, 3, 25],
12          [ "HOTEL_ESTACION", -0.25307, -79.17365, 1, 25],
13          [ "RESTAURANTE_BOM", -0.24781, -79.15229, 3, 14],
14          [ "RESTAURANTE_PAOLA", -0.26004, -79.16534, 5, 19],
15          [ "RESTAURANTE_CHILENO", -0.2476, -79.15319, 3, 23],
16          [ "RESTAURANTE_NELLY", -0.24873, -79.15046, 2, 34],
17          [ "HOTEL_RIOBAMBA", -0.2473, -79.15811, 4, 22],
18          [ "HOTEL_ROCIO", -0.24528, -79.18359, 1, 26],
19          [ "HOTEL_HUASI", -0.24936, -79.16421, 4, 27],
20          [ "HOTEL_TORIL", -0.2478, -79.15211, 2, 15],
21          [ "HOTEL_PRIMAVERA", -0.24848, -79.15261, 3, 6],
22          [ "HOTEL_GABRIEL", -0.24528, -79.15811, 5, 30],
23          [ "HOTEL_SOL", -0.26004, -79.16534, 4, 25],
24          [ "HOTEL_INTERNACIONAL", -0.24525, -79.16612, 3, 25],
25          [ "HOTEL_ACACIAS", -0.24986, -79.16322, 3, 45],
26          [ "HOTEL_MAYA", -0.24525, -79.16612, 3, 21],
27          [ "HOTEL_TROJE", -0.26004, -79.17629, 2, 45],
28          [ "HOTEL_PERUANA", -0.2473, -79.15811, 1, 23],
29          [ "HOTEL_MOLINO", -0.24528, -79.16534, 5, 25],
30          [ "HOTEL_SHALOM", -0.24525, -79.16322, 2, 15],
31          [ "RESTAURANTE_TAURO", -0.24759, -79.15341, 4, 26],
32          [ "HOTEL_CHACARERO", -0.24528, -79.18359, 3, 45],
33          [ "RESTAURANTE_JEKE", -0.25337, -79.15229, 2, 32],
34          [ "RESTAURANTE_XIMENITA", -0.24759, -79.19456, 1, 32],
35          [ "RESTAURANTE_ROSITA", -0.25077, -79.16322, 3, 28],
36          [ "HOTEL_DORADO", -0.24986, -79.15261, 2, 35],
37          [ "RESTAURANTE_SABOR", -0.2473, -79.15811, 1, 32],
38          [ "RESTAURANTE_BOEMIA", -0.26004, -79.15046, 3, 24],
39          [ "RESTAURANTE_PILSENER", -0.24848, -79.16322, 4, 37],
40          [ "RESTAURANTE_ANDESBULL", -0.25578, -79.18003, 1, 45],
41          [ "RESTAURANTE_QUINCHOS", -0.25197, -79.15253, 2, 36],
42          [ "RESTAURANTE_NORTENO", -0.24936, -79.18003, 3, 56],
43          [ "RESTAURANTE_YAGUAR", -0.24886, -79.1402, 4, 23],
44          [ "RESTAURANTE_BURGER", -0.25382, -79.16772, 5, 67],
45          [ "RESTAURANTE_LINA", -0.24759, -79.13482, 3, 43],
46          [ "RESTAURANTE_COFFEREX", -0.26618, -79.11198, 4, 76],
47          [ "RESTAURANTE_BONNY", -0.25337, -79.1681, 14])
48 for ll in listal:
49     #SE INICIA EL METODO DE GENERAR NODOS LUGARES
50     print(str(ll[0]),str(ll[1]),str(ll[2]))
51     grafo.CREAR_LUGAR("NODO CREADO",str(ll[0]),str(ll[1]),str(ll[2]))
52
53 # SE GENERA LAS RELACIONES DE LOS LUGARES
54 #SE CREA LA LISTA DE LAS RELACIONES Y LOS NODOS
55 listal = ([ 'PARQUE_CENTRAL', 'HOTEL_MONTECARLO', 0.10371, 0.350],
56          [ 'PARQUE_CENTRAL', 'RESTAURANTE_CODELL', 0.9416, 0.600],
57          [ 'RESTAURANTE_CODELL', 'RESTAURANTE_PICHINCHA', 0.11894, 0.130],
58          [ 'HOTEL_MONTECARLO', 'RESTAURANTE_CHURRERIA', 0.6768, 0.9100],
59          [ 'HOTEL_MONTECARLO', 'RESTAURANTE_QUINTA_DORADA', 0.4681, 0.3500],

```

```

60 ['RESTAURANTE_CORBAN_COFFEE', 'RESTAURANTE_TENTACIONES', 0.5173, 0.7700],
61 ['RESTAURANTE_CORBAN_COFFEE', 'HOTEL_LIBERTADOR', 0.1041, 0.1100],
62 ['RESTAURANTE_CORBAN_COFFEE', 'RESTAURANTE_DELIRIO', 0.10271, 0.350],
63 ['HOTEL_LIBERTADOR', 'HOTEL_IMPERIAL', 0.37, 0.350],
64 ['HOTEL_LIBERTADOR', 'HOTEL_ESTACION', 0.121, 0.400],
65 ['RESTAURANTE_CORBAN_COFFEE', 'RESTAURANTE_BOM', 0.5184, 0.350],
66 ['RESTAURANTE_BOM', 'RESTAURANTE_PAOLA', 0.184, 0.190],
67 ['RESTAURANTE_PICHINCHA', 'RESTAURANTE_CHILENO', 0.131, 0.500],
68 ['RESTAURANTE_PICHINCHA', 'RESTAURANTE_NELLY', 0.51, 0.290],
69 ['RESTAURANTE_NELLY', 'RESTAURANTE_DELIRIO', 0.115, 0.130],
70 ['RESTAURANTE_CHILENO', 'RESTAURANTE_NELLY', 0.116, 0.500],
71 ['RESTAURANTE_DELIRIO', 'HOTEL_DORADO', 0.101, 0.140],
72 ['RESTAURANTE_DELIRIO', 'HOTEL_LIBERTADOR', 0.121, 0.550],
73 ['HOTEL_LIBERTADOR', 'HOTEL_DORADO', 0.117, 0.270],
74 ['HOTEL_DORADO', 'HOTEL_ESTACION', 0.131, 0.140],
75 ['RESTAURANTE_CHILENO', 'HOTEL_DORADO', 0.294, 0.450],
76 ['RESTAURANTE_PAOLA', 'HOTEL_LIBERTADOR', 0.150, 0.250],
77 ['RESTAURANTE_PAOLA', 'HOTEL_ESTACION', 0.191, 0.450],
78 ['HOTEL_RIOBAMBA', 'RESTAURANTE_CHURRERIA', 0.209, 0.650],
79 ['RESTAURANTE_RIOBAMBA', 'RESTAURANTE_PAOLA', 0.5300, 0.6400],
80 ['RESTAURANTE_QUINTA_DORADA', 'RESTAURANTE_PAOLA', 0.187, 0.400],
81 ['RESTAURANTE_CHURRERIA', 'HOTEL_ESTACION', 0.338, 0.400],
82 ['RESTAURANTE_CODELL', 'HOTEL_DORADO', 0.358, 0.450],
83 ['HOTEL_ESTACION', 'PARQUE_CENTRAL', 0.351, 0.450],
84 ['RESTAURANTE_NELLY', 'RESTAURANTE_TENTACIONES', 0.158, 0.170],
85 ['RESTAURANTE_DELIRIO', 'RESTAURANTE_PAOLA', 0.263, 0.500],
86 ['HOTEL_RIOBAMBA', 'RESTAURANTE_LINA', 0.205, 0.500],
87 ['RESTAURANTE_LINA', 'RESTAURANTE_PAOLA', 2.41, 3.30],
88 ['RESTAURANTE_LINA', 'RESTAURANTE_BUGER', 2.6, 2.5],
89 ['RESTAURANTE_CHURRERIA', 'RESTAURANTE_NORTENO', 1.04, 1.9],
90 ['RESTAURANTE_NORTENO', 'RESTAURANTE_LINA', 0.189, 0.260],
91 ['RESTAURANTE_QUINTA_DORADA', 'RESTAURANTE_NORTENO', 0.189, 0.260],
92 ['HOTEL_IMPERIAL', 'RESTAURANTE_TENTACIONES', 0.95, 0.400],
93 ['RESTAURANTE_CHURRERIA', 'RESTAURANTE_YAGUAR', 0.699, 0.850],
94 ['RESTAURANTE_YAGUAR', 'RESTAURANTE_BONNY', 1.33, 2.1],
95 ['RESTAURANTE_YAGUAR', 'RESTAURANTE_BURGER', 1.26, 2.0],
96 ['RESTAURANTE_PILSENER', 'RESTAURANTE_YAGUAR', 1.30, 1.5],
97 ['RESTAURANTE_PILSENER', 'RESTAURANTE_QUINCHOS', 1.23, 2.0],
98 ['HOTEL_MONTECARLO', 'RESTAURANTE_PILSENER', 0.654, 0.800],
99 ['RESTAURANTE_BURGER', 'HOTEL_RIOBAMBA', 0.6768, 0.9100],
100 ['HOTEL_MONTECARLO', 'RESTAURANTE_CHURRERIA', 0.6768, 0.9100],
101 ['RESTAURANTE_BONNY', 'RESTAURANTE_BURGER', 0.6768, 0.9100],
102 ['RESTAURANTE_BONNY', 'RESTAURANTE_QUINCHOS', 0.6768, 0.9100],
103 ['HOTEL_IMPERIAL', 'RESTAURANTE_CHILENO', 0.284, 0.500],
104 ['RESTAURANTE_HUASI', 'HOTEL_IMPERIAL', 0.676, 0.650],
105 ['RESTAURANTE_QUINCHOS', 'RESTAURANTE_BOEMIA', 0.6768, 0.9100],
106 ['RESTAURANTE_BOEMIA', 'RESTAURANTE_ANDESBULL', 0.6768, 0.9100],
107 ['RESTAURANTE_BOEMIA', 'RESTAURANTE_XIMENITA', 0.6768, 0.9100],
108 ['RESTAURANTE_SABOR', 'RESTAURANTE_PILSENER', 1.54, 2.4],
109 ['RESTAURANTE_SABOR', 'RESTAURANTE_BOEMIA', 0.6768, 0.9100],
110 ['RESTAURANTE_ROSITA', 'RESTAURANTE_SABOR', 2.26, 3.0],
111 ['HOTEL_CHACARRERO', 'RESTAURANTE_SABOR', 0.65, 0.950],
112 ['HOTEL_CHACARRERO', 'RESTAURANTE_XIMENITA', 0.688, 0.918],
113 ['RESTAURANTE_QUINTA_DORADA', 'HOTEL_CHACARRERO', 0.67, 0.9100],
114 ['RESTAURANTE_XIMENITA', 'RESTAURANTE_ROSITA', 0.67, 0.874],
115 ['RESTAURANTE_ROSITA', 'RESTAURANTE_COFFEREX', 0.54, 0.74],
116 ['RESTAURANTE_QUINTA_DORADA', 'HOTEL_CHACARRERO', 0.68, 0.470],
117 ['HOTEL_ESTACION', 'RESTAURANTE_PICHINCA', 0.57, 0.74],
118 ['RESTAURANTE_CODELL', 'HOTEL_RIOBAMBA', 0.812, 0.74],
119 ['HOTEL_RIOBAMBA', 'HOTEL_PRIVAMERA', 0.550, 0.7420],
120 ['RESTAURANTE_COFFEREX', 'HOTEL_HUASI', 0.57, 0.74],

```

```

121     ['RESTAURANTE_COFFEREX', 'HOTEL_SOL', 0.57, 0.74],
122     ['HOTEL_SOL', 'RESTAURANTE_PILSENER', 0.57, 0.74],
123     ['HOTEL_ESTACION', 'RESTAURANTE_PICHINCA', 0.57, 0.74],
124     ['HOTEL_SOL', 'HOTEL_ROCIO', 0.47, 0.78],
125     ['HOTEL_SOL', 'RESTAURANTE_CODELL', 0.74, 0.65],
126     ['RESTAURANTE_ANDESBULL', 'RESTAURANTE_XIMENITA', 0.57, 0.74],
127     ['HOTEL_ROCIO', 'RESTAURANTE_CODELL', 0.57, 0.74],
128     ['HOTEL_CHACARRERO', 'HOTEL_ROCIO', 0.57, 0.74],
129     ['HOTEL_PRIMAVERA', 'HOTEL_ROCIO', 0.57, 0.74],
130     ['RESTAURANTE_CHILENO', 'HOTEL_PRIMAVERA', 0.57, 0.74],
131     ['HOTEL_PRIMAVERA', 'RESTAURANTE_JEKE', 0.57, 0.74],
132     ['HOTEL_HUASI', 'RESTAURANTE_JEKE', 0.57, 0.74],
133     ['HOTEL_HUASI', 'RESTAURANTE_CHACARRERO', 0.57, 0.74],
134     ['HOTEL_HUASI', 'HOTEL_ROCIO', '0.57', '0.74'],
135     ['RESTAURANTE_YAGUAR', 'RESTAURANTE_QUINCHOS', 0.25, 0.50],
136     ['RESTAURANTE_JEKE', 'HOTEL_IMPERIAL', 0.30, 0.38]
137
138 )
139
140 #          ['RESTAURANTE_QUINTA_DORADA', 'RESTAURANTE_CORBAN_COFFEE', '0.6079', '0.8000']
141 for ll in listal:
142     #SE INICIA EL METODO DE GENERAR NODOS LUGARES
143     grafo.CREAR_RUTA(str(ll[0]), str(ll[1]), str(ll[2]), str(ll[3]))
144
145 #-----NODOS PARA ALGORITMO KNN-----
146 listar= (["RESTAURANTE_CODELL", -0.2975, -79.16784, 4, 11],
147          ["RESTAURANTE_PICHINCHA", -0.24665, -79.19275, 1, 9],
148          ["RESTAURANTE_CHURRERIA", -0.2975, -79.1765, 2, 2],
149          ["RESTAURANTE_QUINTA_DORADA", -0.23421, -79.16772, 5, 23],
150          ["RESTAURANTE_CORBAN_COFFEE", -0.27227, -79.1402, 3, 15],
151          ["RESTAURANTE_TENTACIONES", -0.25516, -79.18057, 2, 16],
152          ["RESTAURANTE_DELIRIO", -0.25487, -79.1854, 4, 20],
153          ["RESTAURANTE_BOM", -0.24781, -79.15229, 3, 14],
154          ["RESTAURANTE_PAOLA", -0.26004, -79.16534, 5, 19],
155          ["RESTAURANTE_CHILENO", -0.2476, -79.15319, 3, 23],
156          ["RESTAURANTE_NELLY", -0.24873, -79.15046, 2, 34],
157          ["RESTAURANTE_TAURO", -0.24759, -79.15341, 4, 26],
158          ["RESTAURANTE_SABOR", -0.2473, -79.15811, 1, 32],
159          ["RESTAURANTE_BOEMIA", -0.26004, -79.15046, 3, 24],
160          ["RESTAURANTE_PILSENER", -0.24848, -79.16322, 4, 37],
161          ["RESTAURANTE_ANDESBULL", -0.25578, -79.18003, 1, 45],
162          ["RESTAURANTE_QUINCHOS", -0.25197, -79.15253, 2, 36],
163          ["RESTAURANTE_NORTENO", -0.24936, -79.18003, 3, 56],
164          ["RESTAURANTE_YAGUAR", -0.24886, -79.1402, 4, 23],
165          ["RESTAURANTE_BURGER", -0.25382, -79.16772, 5, 67],
166          ["RESTAURANTE_LINA", -0.24759, -79.13482, 3, 43],
167          ["RESTAURANTE_COFFEREX", -0.26618, -79.11198, 4, 76],
168          ["RESTAURANTE_BONNY", -0.25337, -79.1681, 1, 14],
169          ["RESTAURANTE_JEKE", -0.25337, -79.15229, 2, 32],
170          ["RESTAURANTE_XIMENITA", -0.24759, -79.19456, 1, 32],
171          ["RESTAURANTE_ROSITA", -0.25077, -79.16322, 3, 28])
172 st = "Restaurantes"
173 for ll in listar:
174     #SE INICIA EL METODO DE GENERAR NODOS LUGARES
175     print(str(ll[0]), str(ll[1]), str(ll[2]))
176     grafo.CREAR_LUGAR_KNN("NODO CREADO", str(ll[0]), str(ll[1]), str(ll[2]), str(ll[3]), st)
177 #SE CREA LA LISTA DE NODOS LUGARES
178 listal = (["HOTEL_MONTECARLO", -79.16885, -09.17885, 2, 10],
179          ["PARQUE_CENTRAL", -0.22214, -0.34216, 3, 20],
180          ["HOTEL_IMPERIAL", -0.24886, -79.15787, 3, 25],
181          ["HOTEL_ESTACION", -0.25307, -79.17365, 1, 25],

```

```

182     ["HOTEL_RIOBAMBA", -0.2473, -79.15811, 4, 22],
183     ["HOTEL_ROCIO", -0.24528, -79.18359, 1, 26],
184     ["HOTEL_HUASI", -0.24936, -79.16421, 4, 27],
185     ["HOTEL_TORIL", -0.2478, -79.15211, 2, 15],
186     ["HOTEL_PRIMAVERA", -0.24848, -79.15261, 3, 6],
187     ["HOTEL_GABRIEL", -0.24528, -79.15811, 5, 30],
188     ["HOTEL_SOL", -0.26004, -79.16534, 4, 25],
189     ["HOTEL_INTERNACIONAL", -0.24525, -79.16612, 3, 25],
190     ["HOTEL_ACACIAS", -0.24986, -79.16322, 3, 45],
191     ["HOTEL_MAYA", -0.24525, -79.16612, 3, 21],
192     ["HOTEL_TROJE", -0.26004, -79.17629, 2, 45],
193     ["HOTEL_PERUANA", -0.2473, -79.15811, 1, 23],
194     ["HOTEL_MOLINO", -0.24528, -79.16534, 5, 25],
195     ["HOTEL_SHALOM", -0.24525, -79.16322, 2, 15],
196     ["HOTEL_CHACARERO", -0.24528, -79.18359, 3, 45],
197     ["HOTEL_DORADO", -0.24986, -79.15261, 2, 35])
198 st = "Hoteles"
199 for ll in listal:
200     #SE INICIA EL METODO DE GENERAR NODOS LUGARES
201     print(str(ll[0]),str(ll[1]),str(ll[2]))
202     grafo.CREAR_LUGAR_KNN("NODO CREADO",str(ll[0]),str(ll[1]),str(ll[2]),str(ll[3]),st

```

```

HOTEL_MONTECARLO -79.16885 -9.17885
EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....
None
PARQUE_CENTRAL -0.22214 -0.34216
EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....
None
RESTAURANTE_CODELL -0.2975 -79.16784
EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....
None
RESTAURANTE_PICHINCHA -0.24665 -79.19275
EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....
None
RESTAURANTE_CHURRERIA -0.2975 -79.1765
EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....
None
RESTAURANTE_QUINTA_DORADA -0.23421 -79.16772
EL NODO LUGAR YA EXISTE, INGRESAR OTRO LUGAR.....
None
RESTAURANTE CORBAN COFFEE -0.27227 -79.1402

```




In [5]:

```
1
2 #IMPLEMENTAR EL ALGORITMO DE BUSQUEDA POR AMPLITUD
3
4 # Creamos La clase Nodo
5 class Node:
6     def __init__(self, data, child=None): # Constructor de la clase
7         self.data = data
8         self.child = None
9         self.fathr = None
10        self.cost = None
11        self.set_child(child)
12
13    def set_child(self, child): # Agregar hijos
14        self.child = child
15        if self.child is not None:
16            for ch in self.child:
17                ch.fathr = self
18
19    def equal(self, node): # Igual al equals de Java
20        if self.data == node.data:
21            return True
22        else:
23            return False
24
25    def on_list(self, node_list): # Verificar si el nodo esta en la lista
26        listed = False
27        for n in node_list:
28            if self.equal(n):
29                listed = True
30        return listed
31
32    def __str__(self): # Igual al toString Java
33        return str(self.data)
34
35
36
37
38
39
40 # Implementacion del metodo de busqueda por amplitud
41 def search_Amplitud_solution(connections, init_state, solution):
42     solved = False # Variable para almacenar el estado de la busqueda
43     visited_nodes = [] # Nodos visitados
44     frontrs_nodes = [] # Nodos en busqueda o lista de nodos
45
46     init_node = Node(init_state) # Nodo inicial
47     frontrs_nodes.append(init_node)
48     while (not solved) and len(frontrs_nodes) != 0:
49         node = frontrs_nodes[0]
50         # extraer nodo y añadirlo a visitados
51         visited_nodes.append(frontrs_nodes.pop(0))
52         if node.data == solution: # Preguntar si el nodo obtenido es la solucion
53             solved = True
54             return node # Retornamos el nodo de la solucion
55         else:
56             # expandir nodos hijo - ciudades con conexion
57             node_data = node.data
58             child_list = []
59             for chld in connections[node_data]:
```



```

60         child = Node(chld)
61         child_list.append(child)
62         if not child.on_list(visited_nodes) and not child.on_list(frontrs_node
63             frontrs_nodes.append(child)
64         node.set_child(child_list)
65
66 if __name__ == "__main__":
67     connections = {
68         'PARQUE_CENTRAL': {'RESTAURANTE_CODELL', 'HOTEL_MONTECARLO'},
69         'HOTEL_MONTECARLO': {'RESTAURANTE_PILSENER', 'RESTAURANTE_CHURRERIA', 'RESTAURA
70         'RESTAURANTE_QUINTA_DORADA': {'RESTAURANTE_PAOLA', 'RESTAURANTE_NORTENO'},
71         'RESTAURANTE_NORTENO': {'RESTAURANTE_LINA'},
72         'RESTAURANTE_CODELL': {'HOTEL_RIOBAMBA', 'HOTEL_DORADO', 'RESTAURANTE_PICHINCH
73         'HOTEL_RIOBAMBA': {'RESTAURANTE_CHURRERIA', 'RESTAURANTE_LINA'},
74         'RESTAURANTE_LINA': {'RESTAURANTE_PAOLA'},
75         'RESTAURANTE_PILSENER': {'RESTAURANTE_YAGUAR', 'RESTAURANTE_QUINCHOS'},
76         'HOTEL_DORADO': {'HOTEL_ESTACION', ''},
77         'HOTEL_ESTACION': {'PARQUE_CENTRAL'},
78         'RESTAURANTE_PAOLA': {'HOTEL_ESTACION'},
79         'RESTAURANTE_PICHINCHA': {'RESTAURANTE_NELLY', 'RESTAURANTE_CHILENO'},
80         'RESTAURANTE_NELLY': {'RESTAURANTE_DELIRIO', 'RESTAURANTE_TENTACIONES'},
81         'RESTAURANTE_CHILENO': {'RESTAURANTE_NELLY', 'HOTEL_PRIMAVERA'},
82         'HOTEL_IMPERIAL': {'RESTAURANTE_CHILENO', 'RESTAURANTE_TENTACIONES'},
83         'RESTAURANTE_CORBAN_COFFEE': {'RESTAURANTE_BOM', 'RESTAURANTE_DELIRIO'},
84         'RESTAURANTE_BOM': {'RESTAURANTE_PAOLA'},
85         'RESTAURANTE_DELIRIO': {'RESTAURANTE_PAOLA', 'HOTEL_DORADO'},
86         'RESTAURANTE_CHURRERIA': {'RESTAURANTE_YAGUAR', 'RESTAURANTE_NORTENO'},
87         'RESTAURANTE_QUINCHOS': {'RESTAURANTE_BOEMIA'},
88         'RESTAURANTE_SABOR': {'RESTAURANTE_BOEMIA', 'RESTAURANTE_PILSENER'},
89         'RESTAURANTE_BOEMIA': {'RESTAURANTE_ANDESBULL', 'RESTAURANTE_XIMENITA'},
90         'RESTAURANTE_ANDESBULL': {'RESTAURANTE_XIMENITA'},
91         'RESTAURANTE_XIMENITA': {'RESTAURANTE_ROSITA'},
92         'RESTAURANTE_ROSITA': {'RESTAURANTE_SABOR', 'RESTAURANTE_COFFEREX'},
93         'HOTEL_SOL': {'RESTAURANTE_CODELL', 'HOTEL_ROCIO'},
94         'RESTAURANTE_COFFEREX': {'HOTEL_HUASI', 'HOTEL_SOL'},
95         'HOTEL_ROCIO': {'RESTAURANTE_CODELL'},
96         'HOTEL_HUASI': {'HOTEL_ROCIO', 'RESTAURANTE_JEKE'},
97         'HOTEL_PRIMAVERA': {'RESTAURANTE_JEKE', 'HOTEL_ROCIO'},
98         'RESTAURANTE_YAGUAR': {'RESTAURANTE_QUINCHOS'},
99         'RESTAURANTE_JEKE': {'HOTEL_IMPERIAL'}
100     }
101
102     init_state = 'HOTEL_MONTECARLO'
103     solution = 'RESTAURANTE_PAOLA'
104     solution_node = search_Amplitud_solution(connections, init_state, solution)
105     # mostrar resultado
106     result = []
107     node = solution_node
108     if node is not None:
109         while node.fathr is not None:
110             result.append(node.data)
111             node = node.fathr
112         result.append(init_state)
113         result.reverse() # Reverso el resultado (Solo para presentar)
114         print(result)
115     else:
116         print("No hay solucion !!!!")
117
118 #SE CREA EL GRAFO CON LAS RUTAS COMO LAS RELACIONES
119 #CALL gds.graph.create('migrafo', 'Lugares', 'RUTA_DE', { relationshipProperties: 'cos
120

```

```
121 # MATCH (HOTEL_MONTECARLO:Lugares{nombre:'HOTEL_MONTECARLO'}), (RESTAURANTE_PAOLA:Luga
122 # WITH id(HOTEL_MONTECARLO) AS startNode, [id(RESTAURANTE_PAOLA)] AS targetNodes
123 # CALL gds.alpha.bfs.stream('migrafo', {startNode: startNode, targetNodes: targetNodes
124 # YIELD path UNWIND [ n in nodes(path) | n.nombre ] AS tags RETURN tags
125
```

```
[ 'HOTEL_MONTECARLO', 'RESTAURANTE_QUINTA_DORADA', 'RESTAURANTE_PAOLA' ]
```

In [17]:



```

1  #SE IMPLEMENTE ALGORITMO BUSQUEDA POR PROFUNDIDA
2  # Busqueda en Profundidad
3
4  # Creamos la clase Nodo
5  class Node:
6      def __init__(self, data, child=None): # Constructor de la clase
7          self.data = data
8          self.child = None
9          self.fathr = None
10         self.cost = None
11         self.set_child(child)
12
13     def set_child(self, child): # Agregar hijos
14         self.child = child
15         if self.child is not None:
16             for ch in self.child:
17                 ch.fathr = self
18
19     def equal(self, node): # Igual al equals de Java
20         if self.data == node.data:
21             return True
22
23         else:
24             return False
25
26     def on_list(self, node_list): # Verfiicar su el nodo esta en la lista
27         listed = False
28         for n in node_list:
29             if self.equal(n):
30                 listed = True
31         return listed
32
33     def __str__(self): # Igual al toString Java
34         return str(self.data)
35
36 # Implementacion del metodo de busqueda por profundidad
37 def search_profundidad(init_node, solution, visited):
38     visited.append(init_node.data) #Lista de visitados
39     if init_node.data == solution: # Condicion de salida recursividad (Encontro la solu
40         return init_node # Retorno el nodo resultado
41     else:
42         # Expandir nodos sucesores (hijos)
43         node_data = init_node.data
44         son = [node_data[1], node_data[0], node_data[2], node_data[3]]
45         izq = Node(son)
46         son = [node_data[0], node_data[2], node_data[1], node_data[3]]
47         centro = Node(son)
48         son = [node_data[0], node_data[1], node_data[3], node_data[2]]
49         der = Node(son)
50         init_node.set_child([izq, centro, der])
51
52     for node_son in init_node.child: # Recorrer Los nodos hijos
53         if not node_son.data in visited: # No deben estar en los nodos visitados
54             # Llamada Recursiva
55             Solution = search_profundidad(node_son, solution, visited)
56             if Solution is not None: # Cuando encuentra una solucion
57                 return Solution # Retornamos la solucion encontrada
58     return None
59

```

```

60 init_state = [4, 2, 3, 1] # Creamos un estado inicial
61 solution = [1, 2, 3, 4] # La solucion que debe buscar
62 #Inicializamos Las variables
63 solution_node = None
64 visited = []
65 init_node = Node(init_state)
66 node = search_profundidad(init_node, solution, visited) # Llamamos La metodo de busqueda
67
68 # Mostrar Resultado
69 result = []
70 while node.fathr is not None:
71     result.append(node.data)
72     node = node.fathr
73 result.append(init_state)
74 result.reverse() # Reverso el resultado (Solo para presentar)
75 print(result)
76
77
78 #
79 # CALL gds.graph.create('myGraph', 'Lugares', 'RUTA_DE', { relationshipProperties: 'cos
80
81 # ALGORITMO DE BUSQUEDA POR PROFUNDIDAD
82 # MATCH (HOTEL_MONTECARLO:Lugares{nombre:'HOTEL_MONTECARLO'}), (RESTAURANTE_PAOLA:Lugar
83 # WITH id(HOTEL_MONTECARLO) AS startNode, [id(RESTAURANTE_PAOLA)] AS targetNodes
84 # CALL gds.alpha.dfs.stream('myGraph', {startNode: startNode, targetNodes: targetNodes}
85 # YIELD path
86 # UNWIND [ n in nodes(path) | n.nombre ] AS tags
87 # RETURN tags
88

```

```

[[4, 2, 3, 1], [2, 4, 3, 1], [2, 3, 4, 1], [3, 2, 4, 1], [3, 4, 2, 1], [4,
3, 2, 1], [4, 3, 1, 2], [3, 4, 1, 2], [3, 1, 4, 2], [1, 3, 4, 2], [1, 4, 3,
2], [4, 1, 3, 2], [4, 1, 2, 3], [1, 4, 2, 3], [1, 2, 4, 3], [2, 1, 4, 3],
[2, 1, 3, 4], [1, 2, 3, 4]]

```

In []:



```
1 #SE IMPLEMENTE ALGORITMO BUSQUEDAD POR COSTO DE A*
2 # ALGORITMO DE BUSQUEDA A*
3 # MATCH (start:Lugares {nombre:
4 # 'HOTEL_MONTECARLO'}), (end:Lugares {nombre: 'RESTAURANTE_PAOLA'})
5 # CALL gds.alpha.shortestPath.astar.stream({
6 #   nodeProjection: {
7 #     Lugares: {
8 #       properties: ['Longitud', 'Latitud']
9 #     }
10 #   },
11 #   relationshipProjection: {
12 #     CONNECTION: {
13 #       type: 'RUTA_DE',
14 #       orientation: 'UNDIRECTED',
15 #       properties: 'costo'
16 #     }
17 #   },
18 #   startNode: start,
19 #   endNode: end,
20 #   propertyKeyLat: 'Latitud',
21 #   propertyKeyLon: 'Longitud'
22 # })
23 # YIELD nodeId, cost
24 # RETURN gds.util.asNode(nodeId).nombre AS Lugares, cost
25
```

In []:



1