Busqueda por Costo

A continuacion se ejemplifica la busqueda por costo revisada en clase. Para ello se tiene un ejemplo de las ciudades del territorio Ecuatoriano y su distancia.

In [6]:

```
# Busqueda por costo.

# Creamos la clase Nodo
class Node:
    def __init__(self, data, child=None): # Constructor de la clase
        self.data = data
        self.child = None
        self.fathr = None
        self.cost = None # Importante tener el costo de recorer el nodo
        self.set_child(child)

    def set_child(self, child): # Agregar hijos
        self.child = child
        if self.child is not None:
            for ch in self.child:
                ch.fathr = self

    def equal(self, node):
        if self.data == node.data:
            return True
        else:
            return False

    def on_list(self, node_list): # Verfiicar su el nodo esta en la lista
        listed = False
        for n in node_list:
            if self.equal(n):
                listed = True
        return listed

    def __str__(self): # Igual al toString Java
        return str(self.data)
```

In [7]:

```
#Definimos una funcion para obtener el costo - CompareTo (Java)
def Compare(node):
    return node.cost
```

In [10]:

```
# Implementacion del metodo de busqueda por costo
def search_costo_solucion(connections, init_state, solution):
    solved = False # Variable para almacenar el estado de la busqueda
    visited_nodes = [] # Nodos visitados
    frontier_nodes = [] # Nodos en busqueda o lista nodos o nodos por visitar

    init_node = Node(init_state) # Nodo inicial
    init_node.cost =0 # Agregar costo inicial
    frontier_nodes.append(init_node)
    while (not solved) and len(frontier_nodes) != 0:
        frontier_nodes = sorted(frontier_nodes, key=Compare) # Ordenar lista de nodos
        node = frontier_nodes[0]
        visited_nodes.append(frontier_nodes.pop(0)) # Extraer nodo y añadirlo a visitad
os
        if node.data == solution:# Solucion encontrada
            solved = True
            return node
        else:
            node_data = node.data# Expandir nodos hijo (ciudades con conexion)
            child_list = []
            for achild in connections[node_data]: # Recorrera cada uno de los nodos hij
os
                child = Node(achild)
                cost = connections[node_data][achild] # Obtener el costo del nodo
                child.cost = node.cost + cost # Agregamos el costo actual del nodo + el
historial
                child_list.append(child)
                if not child.on_list(visited_nodes):
                    if child.on_list(frontier_nodes): # Si está en la lista lo sustitui
mos con el nuevo valor de coste si es menor
                        for n in frontier_nodes:
                            if n.equal(child) and n.cost > child.cost:
                                frontier_nodes.remove(n)
                                frontier_nodes.append(child)
                    else:
                        frontier_nodes.append(child)
            node.set_child(child_list)

if __name__ == "__main__":
    connections = {
        'Cuenca': {'Riobamba':190, 'Quito':280, 'Guayaquil':170},
        'Latacunga': {'Ambato':50, 'Quito':30},
        'Esmeraldas': {'Manta':80},
        'Manta': {'Guayaquil':60},
        'Quito': {'Riobamba':110, 'Latacunga':30, 'Cuenca':280, 'Guayaquil':190, 'Puyo'
:170},
        'Riobamba': {'Cuenca':190, 'Quito':110},
        'Ambato': {'Latacunga':50, 'Puyo':80, 'Guayaquil':230},
        'Puyo': {'Ambato':60, 'Quito':170},
        'Machala': {'Guayaquil':80},
        'Guayaquil': {'Machala':80, 'Ambato':230, 'Quito':190, 'Cuenca':170, 'Manta':60
}
    }

    init_state = 'Guayaquil'
    solution = 'Puyo'
    solution_node = search_costo_solucion(connections, init_state, solution)
    # mostrar resultado
    result = []
```

```
    node = solution_node
    if node is not None:
        while node.fathr is not None:
            result.append(node.data)
            node = node.fathr
        result.append(init_state)
        result.reverse() # Reverso el resultado (Solo para presentar)
        print(result)
        print("Costo total: %s" % str(solution_node.cost)) # Imprimir el costo total de
llegar al nodo
    else:
        print("No hay solucion !!!!")
```

```
['Guayaquil', 'Ambato', 'Puyo']
Costo total: 310
```

Practica

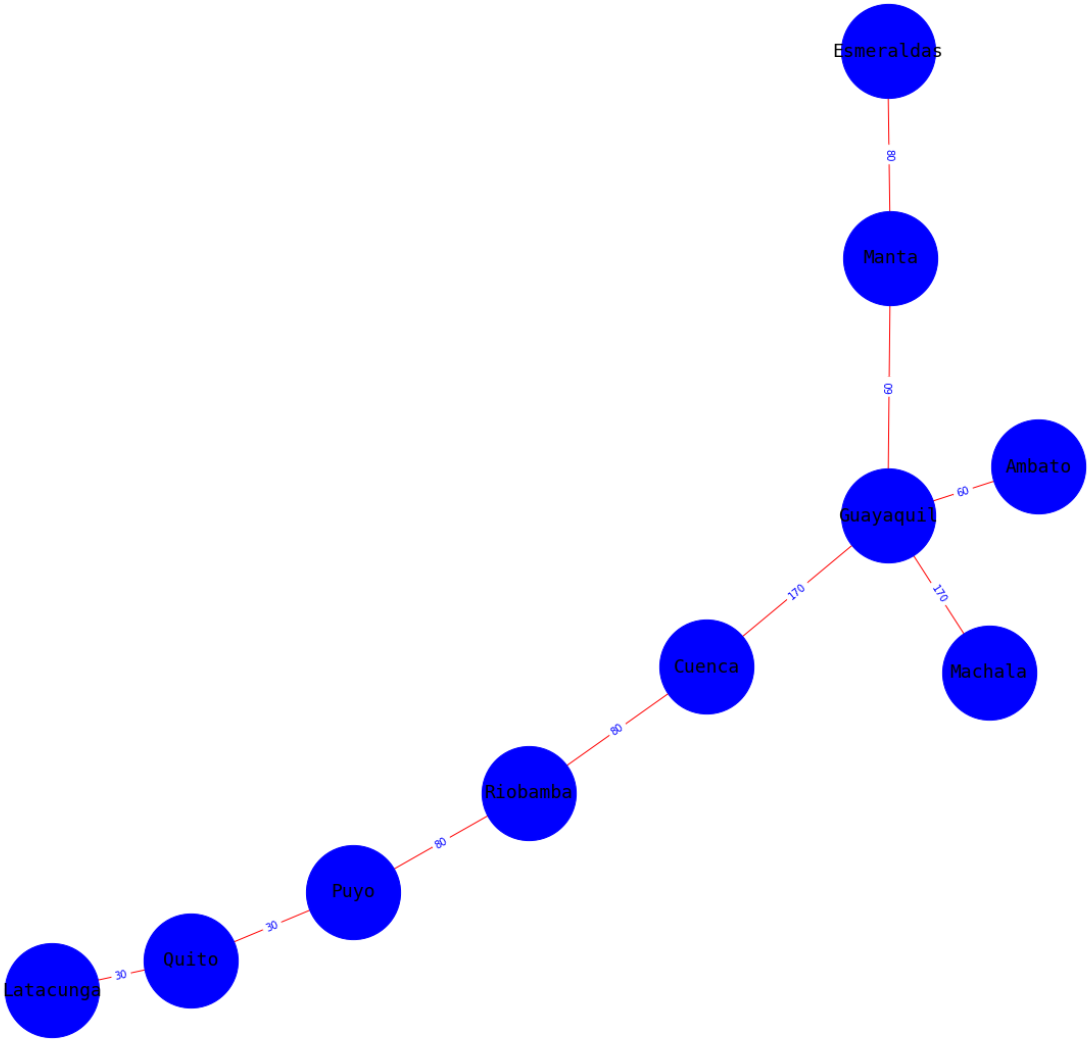
Implementar un algoritmo que me permita dibujar las conexiones con los costos y los resultados del grafo

In [32]:

```
if __name__ == "__main__":
    connections = {
        'Cuenca': {'Riobamba':190, 'Quito':280, 'Guayaquil':170},
        'Latacunga': {'Ambato':50, 'Quito':30},
        'Esmeraldas': {'Manta':80},
        'Manta': {'Guayaquil':60},
        'Quito': {'Riobamba':110, 'Latacunga':30, 'Cuenca':280, 'Guayaquil':190, 'Puyo'
:170},
        'Riobamba': {'Cuenca':190, 'Quito':110},
        'Ambato': {'Latacunga':50, 'Puyo':80, 'Guayaquil':230},
        'Puyo': {'Ambato':60, 'Quito':170},
        'Machala': {'Guayaquil':80},
        'Guayaquil': {'Machala':80, 'Ambato':230, 'Quito':190, 'Cuenca':170, 'Manta':60
}
    }

    init_state = 'Guayaquil'
    solution = 'Puyo'
    solution_node = search_costo_solucion(connections, init_state, solution)
        # mostrar resultado
    result = []
    node = solution_node
    if node is not None:
        while node.fathr is not None:
            result.append(node.data)
            node = node.fathr
        result.append(init_state)
        result.reverse() # Reverso el resultado (Solo para presentar)
        print(result)
        print("Costo total: %s" % str(solution_node.cost)) # Imprimir el costo total de
llegar al nodo
    else:
        print("No hay solucion !!!!")

import matplotlib.pyplot as plt
import networkx as nx
import warnings
warnings.filterwarnings('ignore')

gf = nx.Graph()
gf.add_edge("Cuenca","Guayaquil", label = '170')
gf.add_edge("Latacunga","Quito", label = '30')
gf.add_edge("Esmeraldas","Manta", label = '80')
gf.add_edge("Manta","Guayaquil", label = '60')
gf.add_edge("Quito","Latacunga", label = '30')
gf.add_edge("Cuenca","Guayaquil", label = '170')
gf.add_edge("Riobamba","Cuenca", label = '80')
gf.add_edge("Puyo","Riobamba", label = '80')
gf.add_edge("Ambato","Guayaquil", label = '60')
gf.add_edge("Puyo","Quito", label = '30')
gf.add_edge("Machala","Guayaquil", label = '170')

etiquetas = [gf[u][v]['label'] for u,v in gf.edges()]
plt.figure(4,figsize=(20,20))
pos =nx.spring_layout(gf)
nx.draw_networkx_nodes(gf, pos, node_size=8500, node_color='blue')
nx.draw_networkx_edges(gf, pos, edge_color='red')
nx.draw_networkx_edge_labels(gf,pos,edge_labels={(u,v):gf[u][v]['label'] for u,v in gf.
edges()} ,font_color='blue')
```

```
nx.draw_networkx_labels(gf, pos, font_family='monospace',
       node_color="blue",
                  edge_color="red",
                  font_size=18,
                  width=4,
                  with_labels=True,
                  node_size=8500)
plt.axis('off')

plt.show()
```

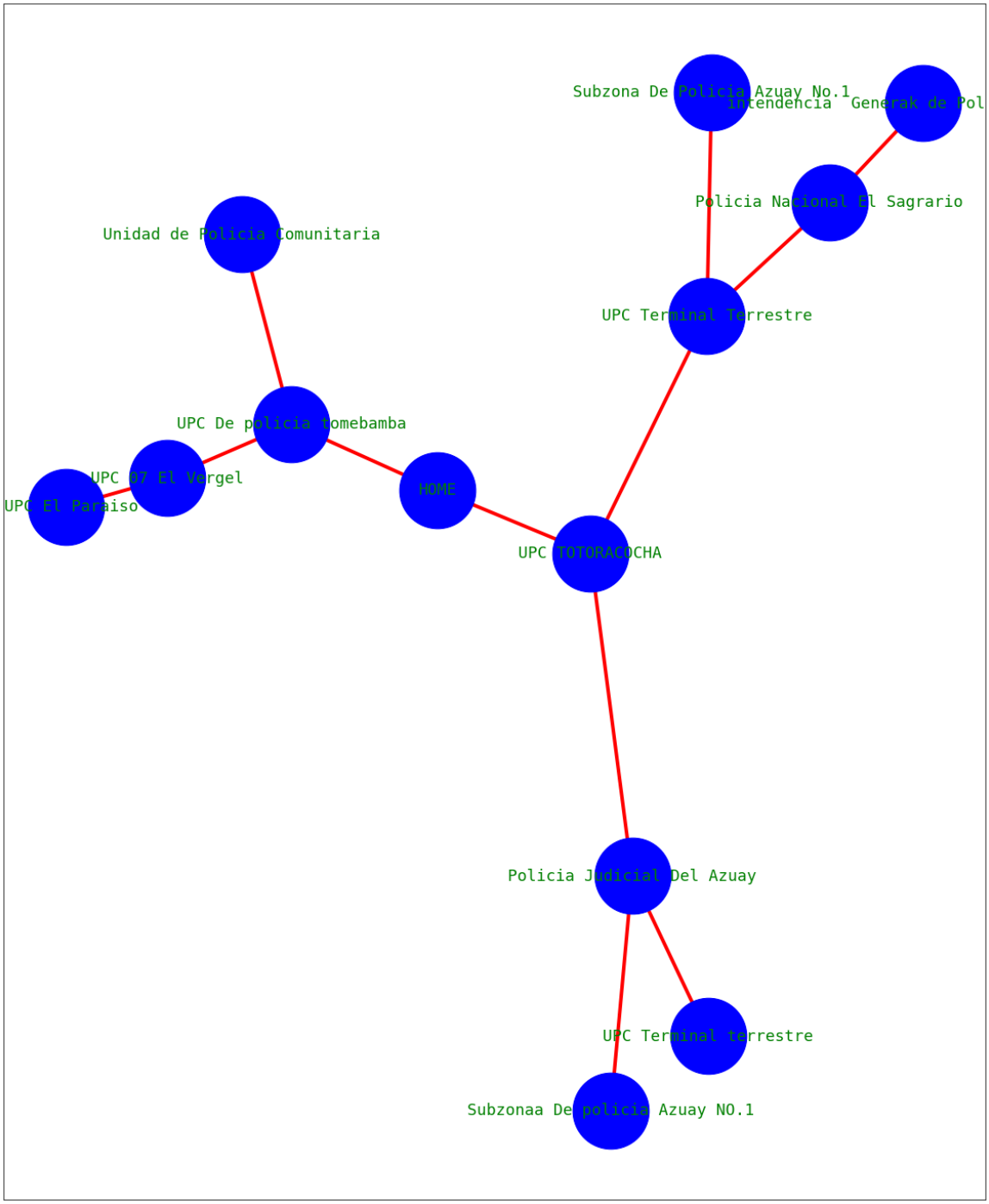
```
['Guayaquil', 'Ambato', 'Puyo']
Costo total: 310
```


Esmeraldas
80
Manta
60
Ambato
60
Guayaquil
170
170
Cuenca
Machala
80
Riobamba
80
Puyo
30
Quito
30
Latacunga

Mediante el uso de la herramienta de Google Maps tomar al su direccion domiciliaria como punto de partida y generar un arbol jerarquico con todos los posibles Policia/UPC/Funcion Judicial, para ello se debe tener como primer nivel los mas cercanos y a continuacion los demas generando un arbol jerarquico.

In [20]:

```
policia = {
    'HOME': {'UPC TOTORACOCHA','UPC De policia tomebamba'},
    'UPC TOTORACOCHA': {'UPC Terminal Terrestre', 'Policia Judicial Del Azuay'},
    'UPC De policia tomebamba': {'UPC 07 El Vergel', 'Unidad de Policia Comunitaria'},
    'UPC Terminal Terrestre':{'Subzona De Policia Azuay No.1','Policia Nacional El Sagr
ario'},
    'Policia Judicial Del Azuay':{'UPC Terminal terrestre','Subzonaa De policia Azuay N
O.1'},
    'UPC 07 El Vergel':{' UPC El Paraiso'},
    'Policia Nacional El Sagrario':{'intendencia  Generak de Policia del azuay'}
}

g = nx.Graph(policia)

for key, valor in policia.items():
        for i in valor:
            g.add_edge(key,i)

plt.figure(4,figsize=(20,25))
nx.draw_networkx(g,font_color='green',
        font_family='monospace',
        node_color="blue",
        edge_color="red",
        font_size=18,
        width=4,
        with_labels=True,
        node_size=7500)
plt.show()
```


Subzona De Policia Azuay No.1
intendencia
Generak de Pol
Policia Nacional El Sagrario
Unidad de Policia Comunitaria
UPC Terminal Terrestre
UPC De policia tomebamba
UPC 67 El Vergel
UPC El Paraiso
HOME
UPC TOTORACOCHA
Policia Judicial Del Azuay
UPC Terminal terrestre
Subzonaa De policia Azuay NO.1

Realizar los calculos para obtener el factor de ramificacion, análisis del algoritmo en términos de completitud, optimalidad, complejidad temporal y complejidad espacial.

In [27]:

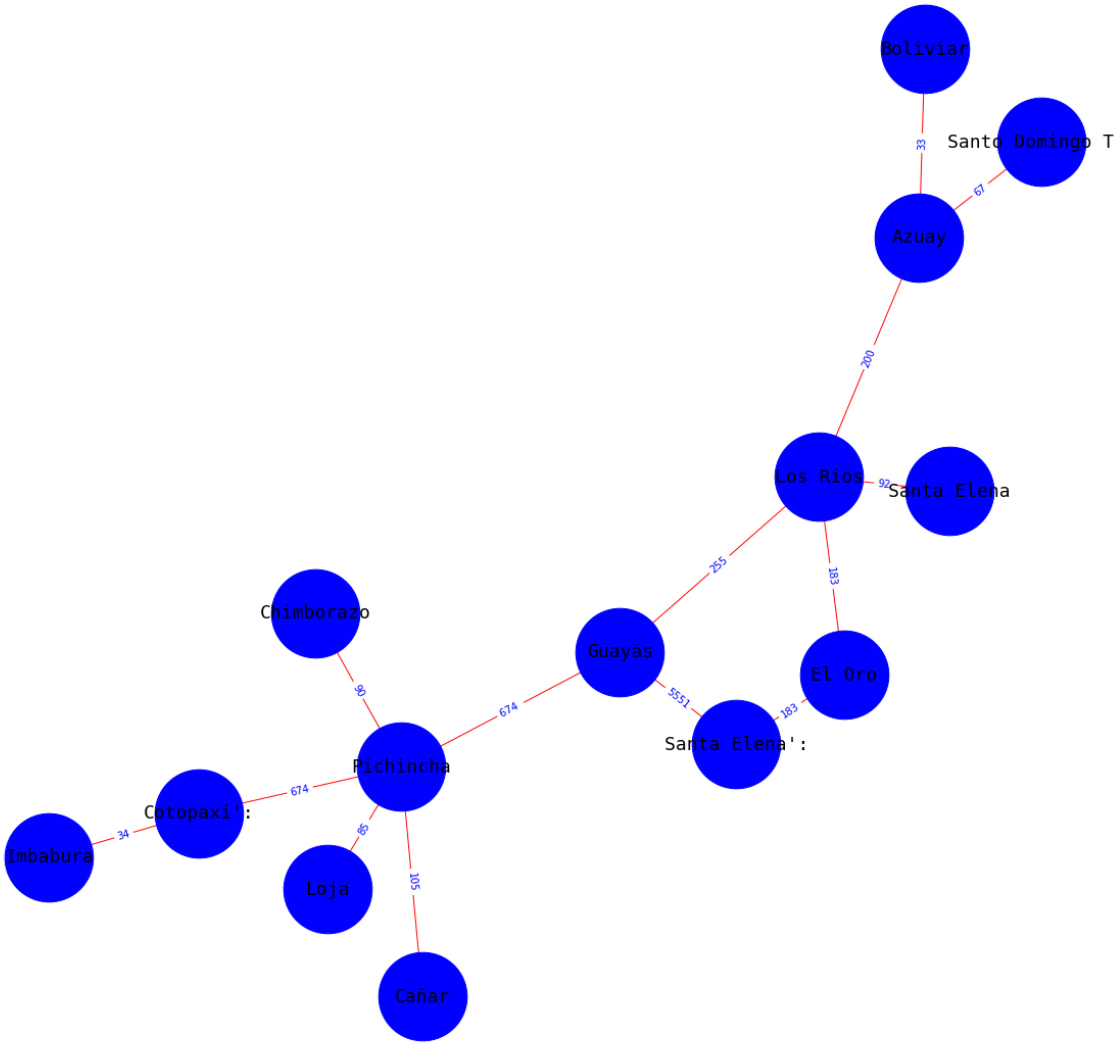
```
n= 3;
d= 13
r = (13/3)
print('factor de ramificacion')
print(r)

print('Complejidad temporal')
o=(4.6**3)
print(o)

print('Complejidad espacial')
o=(4.6*3)
print(o)
```

```
factor de ramificacion
4.333333333333333
Complejidad temporal
97.33599999999998
Complejidad espacial
13.799999999999999
```

Generar un arbol de expansion del COVID-19 en el Ecuador y agregarle al metodo de costo para obtener la ruta de contagio.

In [30]:

```
gr= nx.Graph()

gr.add_edge("Guayas", "Pichincha", weight = 3,label='674 ')
gr.add_edge("Guayas", "Los Rios", weight = 2.5,label ='255')
gr.add_edge("Los Rios", "El Oro", weight = 2.0,label ='183')
gr.add_edge("Los Rios", "Santa Elena", weight = 3.2,label ='92')
gr.add_edge("Los Rios", "Azuay", weight = 3,label ='200')
gr.add_edge("Pichincha", "Chimborazo", weight = 2.5,label ='90')
gr.add_edge("Pichincha", "Cañar", weight = 2.0, label ='105')
gr.add_edge("Pichincha", "Loja", weight = 3.2, label ='85')
gr.add_edge("Azuay", "Boliviar", weight = 3,label ='33')
gr.add_edge("Azuay", "Santo Domingo TsA", weight = 3,label ='67')
gr.add_edge("Cotopaxi':", "Imbabura", weight =2.5,label ='34')
gr.add_edge("Cotopaxi':", "Pichincha", weight =2.5,label ='674')
gr.add_edge("Santa Elena':", "Guayas", weight =2.5,label ='5551')
gr.add_edge("Santa Elena':", "El Oro", weight =2.5,label ='183')

weights = [gr[u][v]['weight'] for u,v in gr.edges()]
etiquetas = [gr[u][v]['label'] for u,v in gr.edges()]
plt.figure(4,figsize=(20,20))
pos =nx.spring_layout(gr)
nx.draw_networkx_nodes(gr, pos, node_size=7500, node_color='blue') #nodos
nx.draw_networkx_edges(gr, pos, edge_color='red')
nx.draw_networkx_edge_labels(gr,pos,edge_labels={(u,v):gr[u][v]['label'] for u,v in gr.
edges()} ,font_color='Blue')

nx.draw_networkx_labels(gr, pos, font_family='monospace',
        node_color="blue",
        edge_color="red",
        font_size=18,
        width=4,
        with_labels=True,
        node_size=7500)
plt.axis('off')

plt.show()
```


Boliviar
33
Santo Domingo T
67
Azuay
200
Los Rios
92
Santa Elena
255
183
Chimborazo
Guayas
El Oro
90
674
5551
183
Santa Elena':
Pichincha
674
Cotopaxi':
34
Imbabura
85
Loja
105
Cañar

In [33]:

```
#metodo de busqueda por costo
import networkx as nx
import matplotlib.pyplot as plt

def search_costo_solucion(connections, init_state, solution):
    solved = False # Variable para almacenar el estado de la busqueda
    visited_nodes = [] # Nodos visitados
    frontier_nodes = [] # Nodos en busqueda o lista nodos o nodos por visitar

    init_node = Node(init_state) # Nodo inicial
    init_node.cost =0 # Agregar costo inicial
    frontier_nodes.append(init_node)
    while (not solved) and len(frontier_nodes) != 0:
        frontier_nodes = sorted(frontier_nodes, key=Compare) # Ordenar lista de nodos
        node = frontier_nodes[0]
        visited_nodes.append(frontier_nodes.pop(0)) # Extraer nodo y a□0Š9adirlo a visi
tados
        if node.data == solution:# Solucion encontrada
            solved = True
            return node
        else:
            node_data = node.data# Expandir nodos hijo (ciudades con conexion)
            child_list = []
            for achild in connections[node_data]: # Recorrera cada uno de los nodos hij
os
                child = Node(achild)
                cost = connections[node_data][achild] # Obtener el costo del nodo
                child.cost = node.cost + cost # Agregamos el costo actual del nodo + el
historial
                child_list.append(child)
                if not child.on_list(visited_nodes):
                    if child.on_list(frontier_nodes): # Si est¨¢ en la lista lo sustitu
imos con el nuevo valor de coste si es menor
                        for n in frontier_nodes:
                            if n.equal(child) and n.cost > child.cost:
                                frontier_nodes.remove(n)
                                frontier_nodes.append(child)
                    else:
                        frontier_nodes.append(child)
            node.set_child(child_list)

if __name__ == "__main__":
    connections = {
        'Guayas':{'Pichincha':868,'Los Rios':382},
        'Los Rios':{'El Oro':276,'Santa Elena':163,'Azuay':247},
        'Pichincha':{'Chimborazo':121,'Cañar':149,'Loja':132},
        'Azuay':{'Bolivar':55,'Sto.Domingo TsA':94},
        'Cotopaxi':{'Imbabura':51,'Pichincha':868},
        'Santa Elena':{'Guayas':7108,'El Oro':276},
        'El Oro':{'Los Rios':382},
        'Bolivar':{'Azuay':249},
        'Sto.Domingo TsA':{'Bolivar':55},
        'Loja':{'Pichincha':868},
        'Chimborazo':{'Cañar':149},
        'Cañar':{'Chimborazo':121}
    }

    init_state = 'Guayas'
    solution = 'Bolivar'
```

```
    solution_node = search_costo_solucion(connections, init_state, solution)
    # mostrar resultado
    result = []
    node = solution_node
    if node is not None:
        while node.fathr is not None:
            result.append(node.data)
            node = node.fathr
        result.append(init_state)
        result.reverse() # Reverso el resultado (Solo para presentar)
        print(result)
        print("Costo total por provincia: %s" % str(solution_node.cost)) # Imprimir el
 costo total de llegar al nodo
    else:
        print("No hay solucion.....!")


print()
```

```
['Guayas', 'Los Rios', 'Azuay', 'Bolivar']
Costo total por provincia: 684
```

Conclusiones

la buqueda por costo nos permite encontar el camino mas corto para llegar al nodo objetivo

In [ ]: