Busqueda por Amplitud

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

In [10]:

```
# Busqueda en Amplitud - Breadth First Search

# 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
        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): # Igual al equals de Java
        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 [19]:

```
# Implementacion del metodo de busqueda por amplitud
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
Grafica = nx.Graph()


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

    init_node = Node(init_state) # Nodo inicial
    frontrs_nodes.append(init_node)
    while (not solved) and len(frontrs_nodes) != 0:
        node = frontrs_nodes[0]
        # extraer nodo y añadirlo a visitados
        visited_nodes.append(frontrs_nodes.pop(0))
        if node.data == solution: # Preguntar se el nodo obtenido es la solucion
            solved = True
            return node # Retornamos el nodo de la solucion
        else:
            # expandir nodos hijo - ciudades con conexion
            node_data = node.data
            child_list = []
            for chld in connections[node_data]:
                child = Node(chld)
                child_list.append(child)
                if not child.on_list(visited_nodes) and not child.on_list(frontrs_nodes):
                    frontrs_nodes.append(child)
            node.set_child(child_list)

if __name__ == "__main__":
    connections = {
        'Cuenca': {'Riobamba', 'Quito', 'Guayaquil'},
        'Latacunga': {'Ambato', 'Quito'},
        'Esmeraldas': {'Manta'},
        'Manta': {'Guayaquil'},
        'Quito': {'Riobamba', 'Latacunga', 'Cuenca', 'Guayaquil', 'Puyo'},
        'Riobamba': {'Cuenca', 'Quito'},
        'Ambato': {'Latacunga', 'Puyo', 'Guayaquil'},
        'Puyo': {'Ambato', 'Quito'},
        'Machala': {'Guayaquil'},
        'Guayaquil': {'Machala', 'Ambato', 'Quito', 'Cuenca', 'Manta'}
    }

    init_state = 'Cuenca'
    solution = 'Ambato'
    solution_node = search_Amplitud_solution(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)
```

```
    else:
        print("No hay solucion !!!!")
```

```
['Cuenca', 'Guayaquil', 'Ambato']
```

Tarea: Cálculo del factor de ramificación

Realice el cálculo del factor de ramificación del problema de las N reinas (con N = 4). Para ello deberá realizar las siguientes actividades:

Asumir que el factor de ramificación es constante. Despejar el valor de b Consultar sitios externos sobre cómo realizar el cálculo

#SOLUCION

#n = numero de nodos

#d = profundidad

#b = factor de ramificacion

n = 4
p = 16

formula:

b = Tn/p
b = 4/16 b = 0.25

Practica

1- Implementar un algoritmo que me permita dibujar las conexiones y los resultados del grafo.

2- 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 Hospitales, para ello se debe tener como primer nivel los mas cercanos y a continuacion los demas.

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

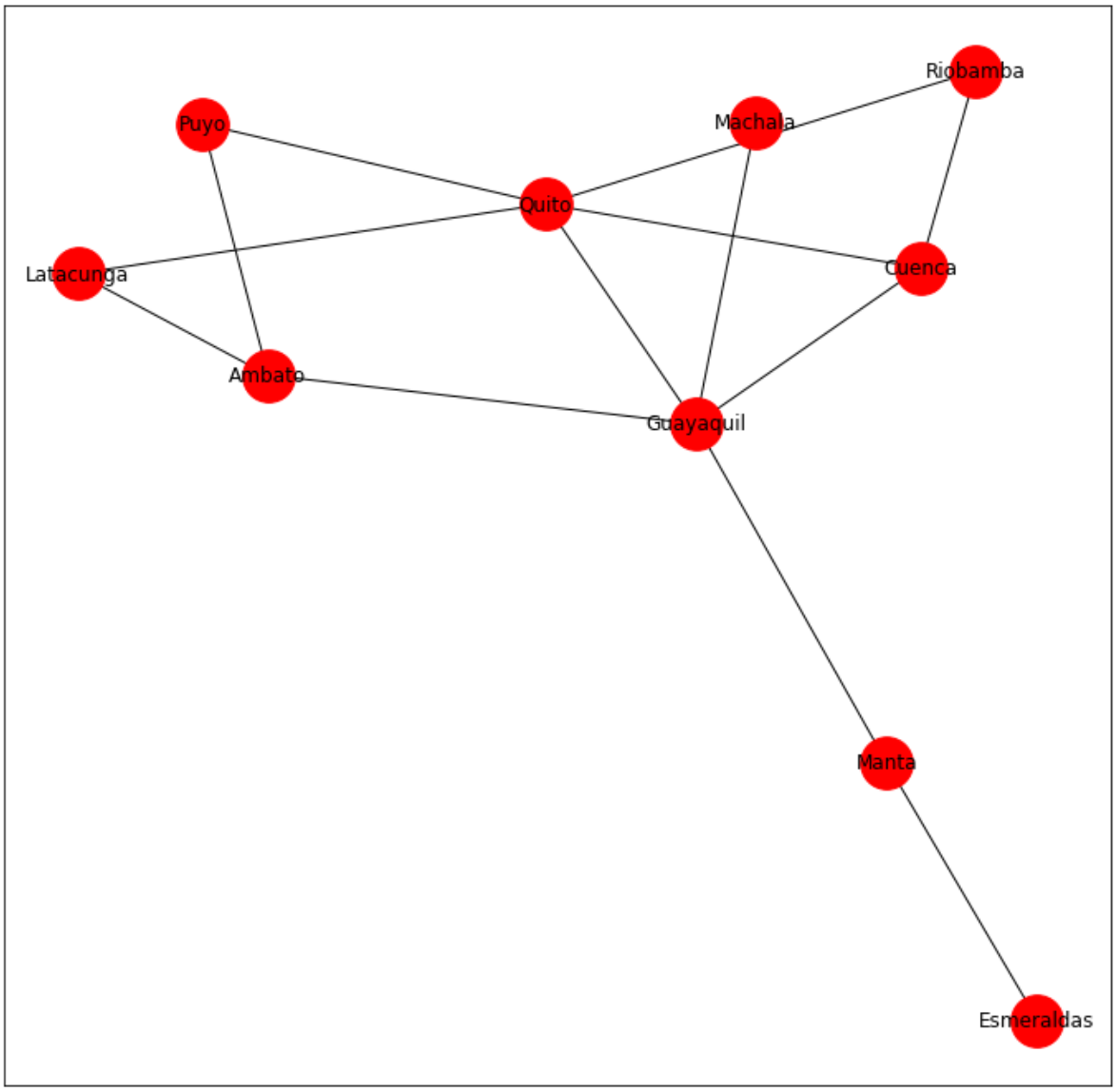
Subir el cuaderno con la resolucion

In [ ]:

```
1.Implementar un algoritmo que me permita dibujar las conexiones y los resultados del grafo
```

In [25]:

```
connections = {
        'Cuenca': {'Riobamba', 'Quito', 'Guayaquil'},
        'Latacunga': {'Ambato', 'Quito'},
        'Esmeraldas': {'Manta'},
        'Manta': {'Guayaquil'},
        'Quito': {'Riobamba', 'Latacunga', 'Cuenca', 'Guayaquil', 'Puyo'},
        'Riobamba': {'Cuenca', 'Quito'},
        'Ambato': {'Latacunga', 'Puyo', 'Guayaquil'},
        'Puyo': {'Ambato', 'Quito'},
        'Machala': {'Guayaquil'},
        'Guayaquil': {'Machala', 'Ambato', 'Quito', 'Cuenca', 'Manta'}
}

Grafica.add_nodes_from(connections)
for ciudad, listaCiudades in connections.items():
        for a in listaCiudades:
            Grafica.add_edge(ciudad,a,size=250)

plt.figure(3,figsize=(12,12))
nx.draw_networkx(Grafica, node_color = 'red', with_label = True, node_size=1000)
plt.show()
```

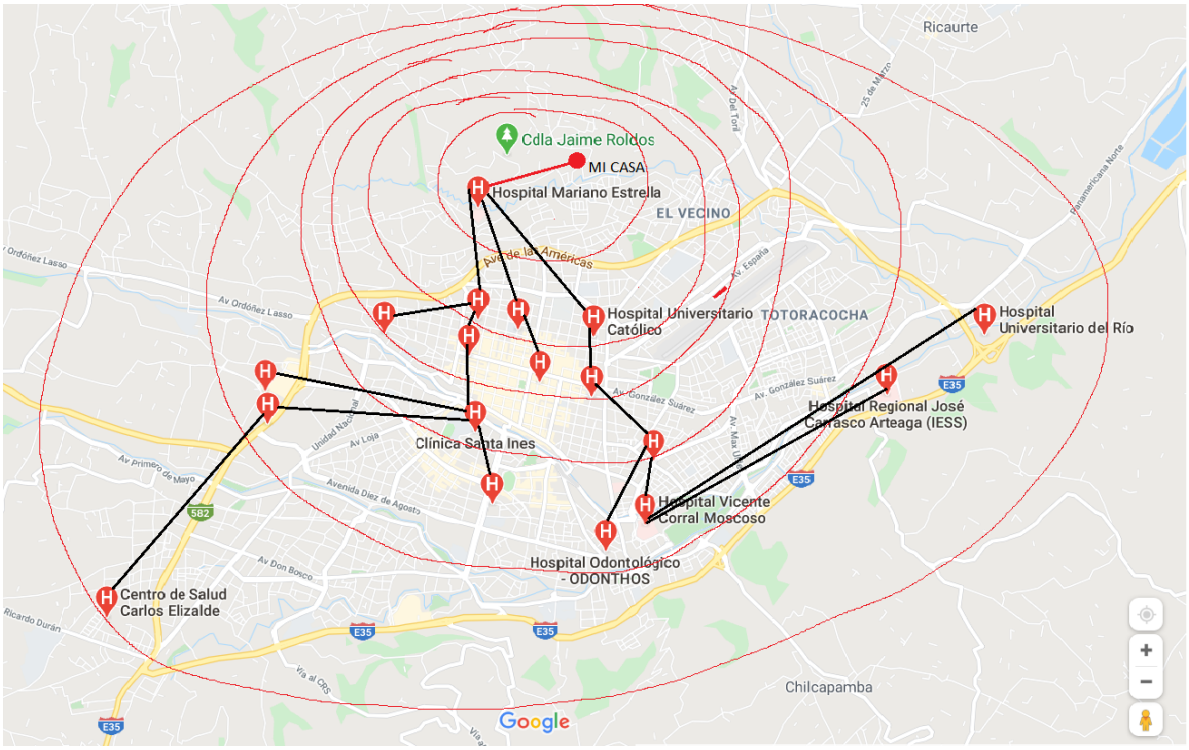


2- Mediante el uso de la herramienta de Google Maps tomar al su direccion domiciliaria como punto de partida y generar un arbol jerarquico con todos los posibles Hospitales, para ello se debe tener como primer nivel los mas cercanos y a continuacion los demas.

In [36]:

```
from IPython.display import Image
Image(filename="hospitales.png")
```

Out[36]:

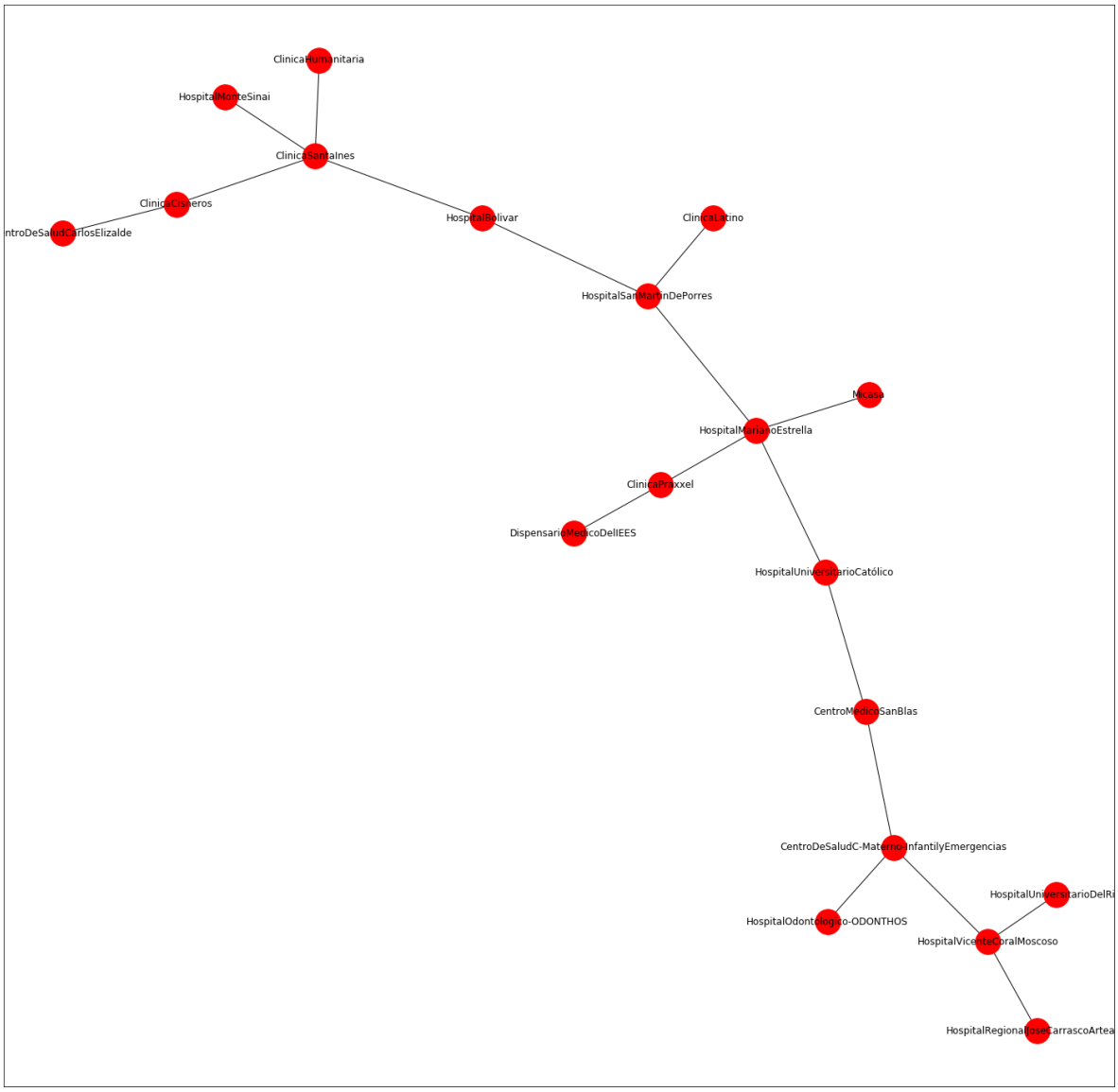


In [30]:

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
if __name__ == "__main__":
    connections = {
        'Micasa': {'HospitalMarianoEstrella'},
        'HospitalMarianoEstrella': {'Micasa','HospitalSanMartinDePorres', 'ClinicaPraxxel',
        'HospitalSanMartinDePorres': {'HospitalMarianoEstrella','ClinicaLatino','HospitalBo
        'ClinicaPraxxel':{'HospitalMarianoEstrella','DispensarioMedicoDelIEES'},
        'HospitalUniversitarioCatólico':{'HospitalMarianoEstrella','CentroMedicoSanBlas'},
        'ClinicaLatino': {'HospitalSanMartinDePorres'},
        'HospitalBolivar': {'HospitalSanMartinDePorres','ClinicaSantaInes'},
        'DispensarioMedicoDelIEES': {'ClinicaPraxxel'},
        'CentroMedicoSanBlas': { 'HospitalUniversitarioCatólico', 'CentroDeSaludC-Materno-I
        'ClinicaSantaInes':{'HospitalBolivar','ClinicaHumanitaria','ClinicaCisneros','Hospi
        'CentroDeSaludC-Materno-InfantilyEmergencias':{'CentroMedicoSanBlas','HospitalOdont
        'ClinicaHumanitaria':{'ClinicaSantaInes'},
        'ClinicaCisneros':{'ClinicaSantaInes','CentroDeSaludCarlosElizalde'},
        'HospitalMonteSinai':{'ClinicaSantaInes'},
        'HospitalOdontologico-ODONTHOS':{'CentroDeSaludC-Materno-InfantilyEmergencias'},
        'HospitalVicenteCoralMoscoso':{'CentroDeSaludC-Materno-InfantilyEmergencias','Hospi
        'CentroDeSaludCarlosElizalde':{'ClinicaCisneros'},
        'HospitalRegionalJoseCarrascoArteaga':{'HospitalVicenteCoralMoscoso'},
        'HospitalUniversitarioDelRio':{'HospitalVicenteCoralMoscoso'}

    }


    init_state = 'Micasa'
    solution = 'HospitalRegionalJoseCarrascoArteaga'
    solution_node = search_Amplitud_solution(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("LA SOLUCION ES : \n")
        print(result)
    else:
        print("No hay solucion !!!!")


    Grafica = nx.Graph()
    Grafica.add_nodes_from(connections)
    for ciudad, listaCiudades in connections.items():
            for a in listaCiudades:
                Grafica.add_edge(ciudad,a,size=250)

    plt.figure(3,figsize=(25,25))
    nx.draw_networkx(Grafica, node_color = 'red', with_label = True, node_size=1000)
    plt.show()
```

LA SOLUCION ES :

['Micasa', 'HospitalMarianoEstrella', 'HospitalUniversitarioCatólico', 'CentroMedicoSanBlas', 'CentroDeSaludC-Materno-InfantilyEmergencias', 'HospitalVicenteCoralMoscoso', 'HospitalRegionalJoseCarrascoArteaga']



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

1.FACTOR DE RAMIFICACION

Tn = 18

d = 6

Formula: b = n/d

b = 18/6

Factor de ramificacion = 3

2.COMPLEJIDAD TEMPORAL

Formula = O(b^n)

Complejidad Temporal = 3^6

Complejidad Temporal = 729

3.COMPLEJIDAD ESPACIAL: Formula = O(nb)

Complejidad Espacial = 6*3

Complejidad Espacial = 18

Conclusiones

- Se puede concluir que se ha llevado a cabo de manera satisfactoria esta practica de busqueda por amplitud, donde hemos podido obtener de manera correcta, las rutas apropadas para llegar de una ciudad a otra.
- Hemos obtenido la ruta apropiada para llegar desde mi vivienda a un hospital utilizando tambien busqueda por amplitud.
- Se ha realizado los graficos correspondientes de las cuidades del ecuador y tambien de las rutas de mi vivienda hacia los hospitales mas cercanos