Búsqueda por Ascenso de Colinas

Introducción

En este trabajo procederemos a realizar el desarrollo de la búsqueda por ascenso de colinas, para ello utilizaremos los restaurantes cercanos a un punto dentro de la ciudad de cuenca. Dentro de ello procederemos a definir un punto de partida y un nodo meta. Para la definición de la distancia utilizaremos la regla establecida en Google maps.

Desarrollo

La búsqueda por escalada o ascenso de colinas es utilizada para encontrar una solución razonable en espacios muy grandes o infinitos. Este tipo de búsqueda explora de acuerdo con la distancia que posee cada nodo determinado por una función heurística.

Para el desarrollo del algoritmo se realiza desde un punto de partida o punto actual hasta un nuevo punto donde se analiza el punto, y si el mejor que el actual, el nuevo punto pasa a ser el punto actual. Este método trata de considerar el camino con menor costo considerado entre el punto o nodo de partida hacia a un estado meta

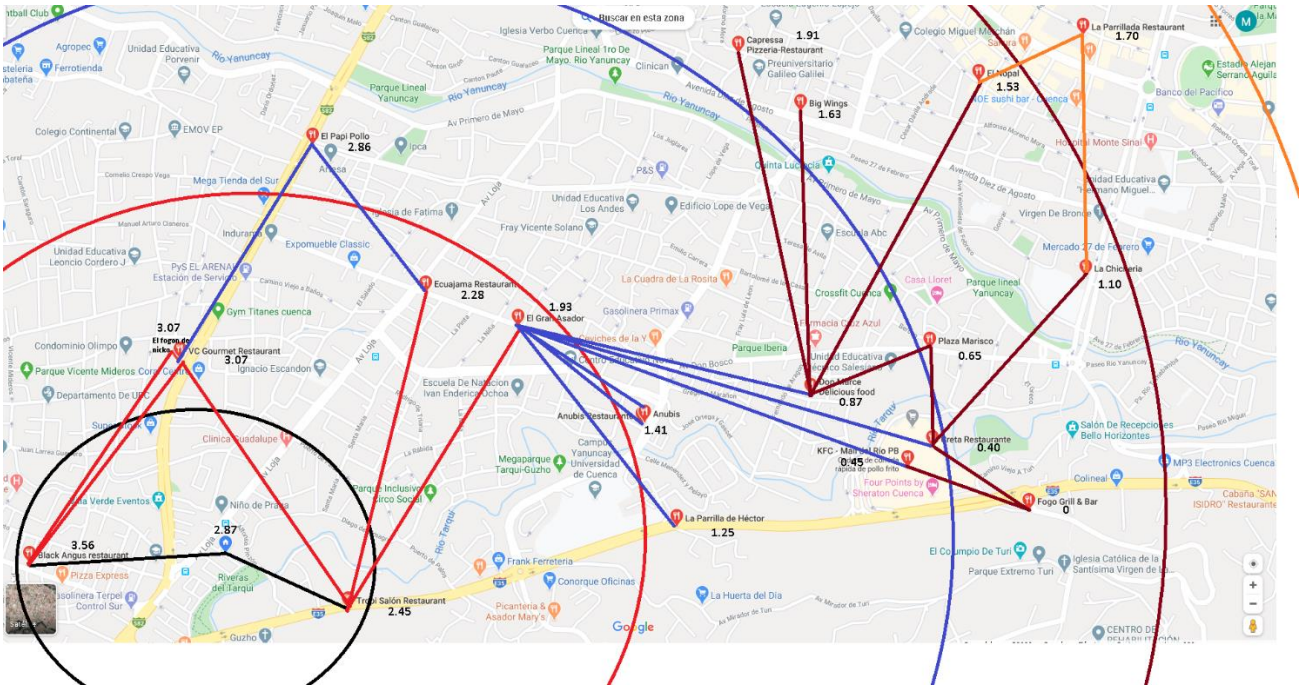
Este algoritmo es considerados locales, ya que solo consideran los nodos consecuentes inmediatos a ellos y sus sucesores. Considerado esto puede que nunca lleguen a encontrar una solución si es que no dan con el nodo o estado objetivo.

Características

- Este tipo de búsqueda reduce el numero de nodos a analizar
- Es informado ya que utiliza la información del nodo por visitar o nodo siguiente.
- Este tipo de algoritmo no realiza una búsqueda a todos los nodos comprendidos en el árbol, como máximo encuentra una solución buena pero no considera si es la mas optima.

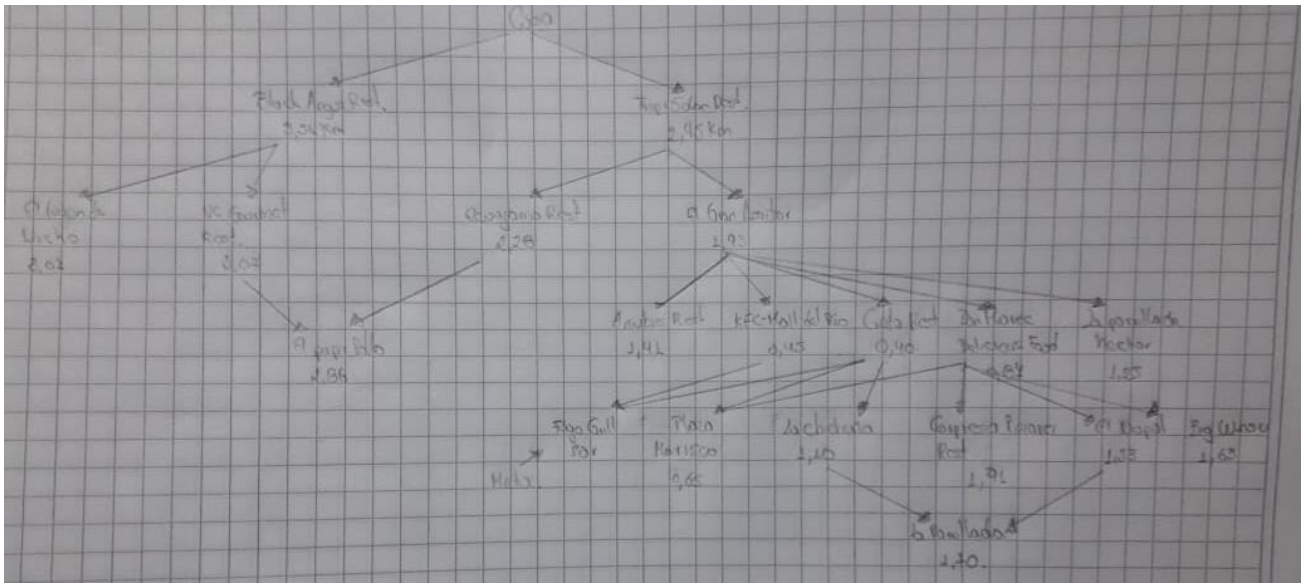
Mapa de Restaurants.

Para la búsqueda de los restaurantes utilizamos Google maps donde procedemos a realizar la búsqueda y la medida de los respectivos nodos. Para ello utilizamos la regla que dispone la herramienta de Google maps.



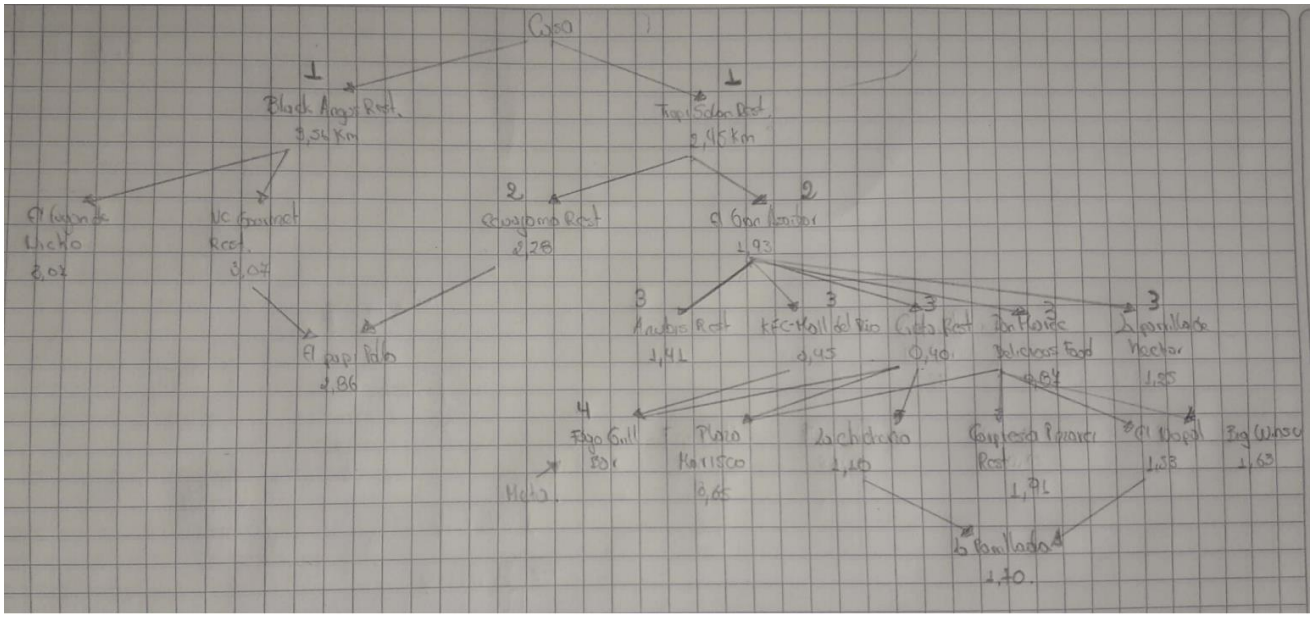
Desarrollo a mano.

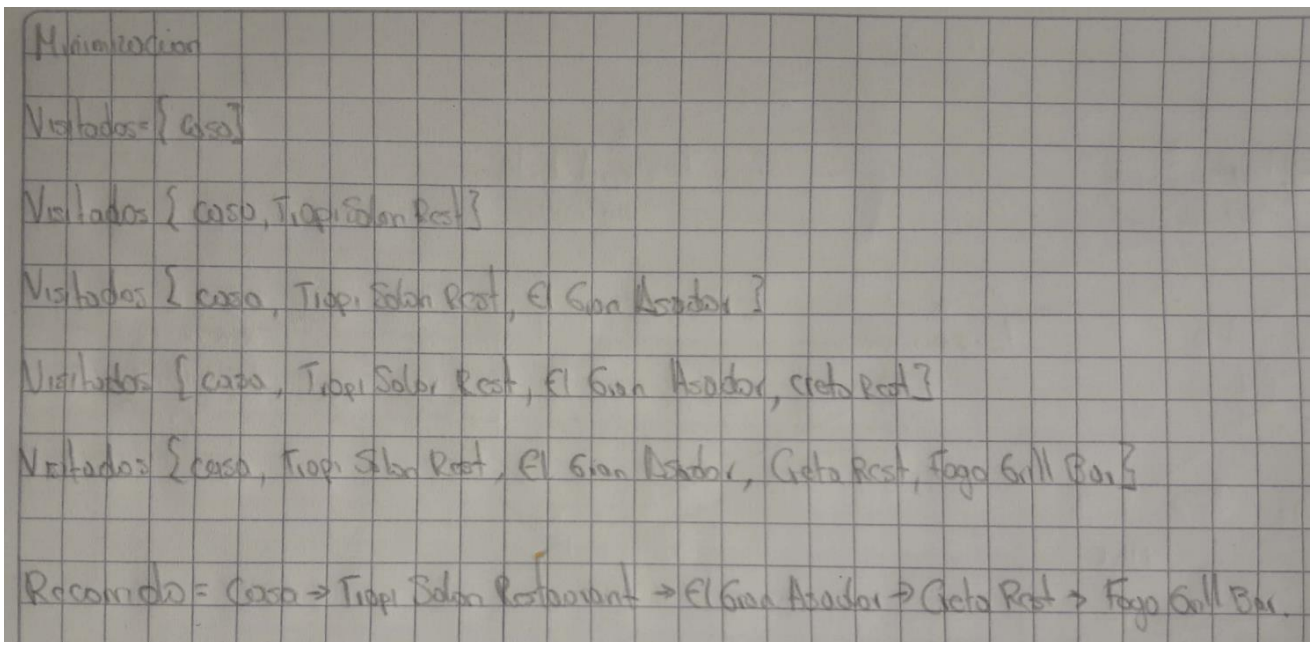
Procedemos a realizar el grafico para el desarrollo a mano donde definimos nuestro nodo de partida y nuestro nodo meta.



Resultado.

Para el desarrollo según el ascenso de colinas realizamos la búsqueda del siguiente nodo con menor coste, y asignamos el nivel 1. Luego procedemos a analizar de los siguientes nodos a partir del nodo actual los cuales son el nivel 2. De los nodos de nivel 2 buscamos el nodo de menor coste y asignamos como el nodo actual, y volvemos a realizar el análisis de este a sus nodos consecutivos para encontrar el nuevo nodo actual los cuales serán de 3 nivel y el nodo con menor coste hasta llegar al nodo meta el cual será de 4 nivel. Y retornamos la solución.



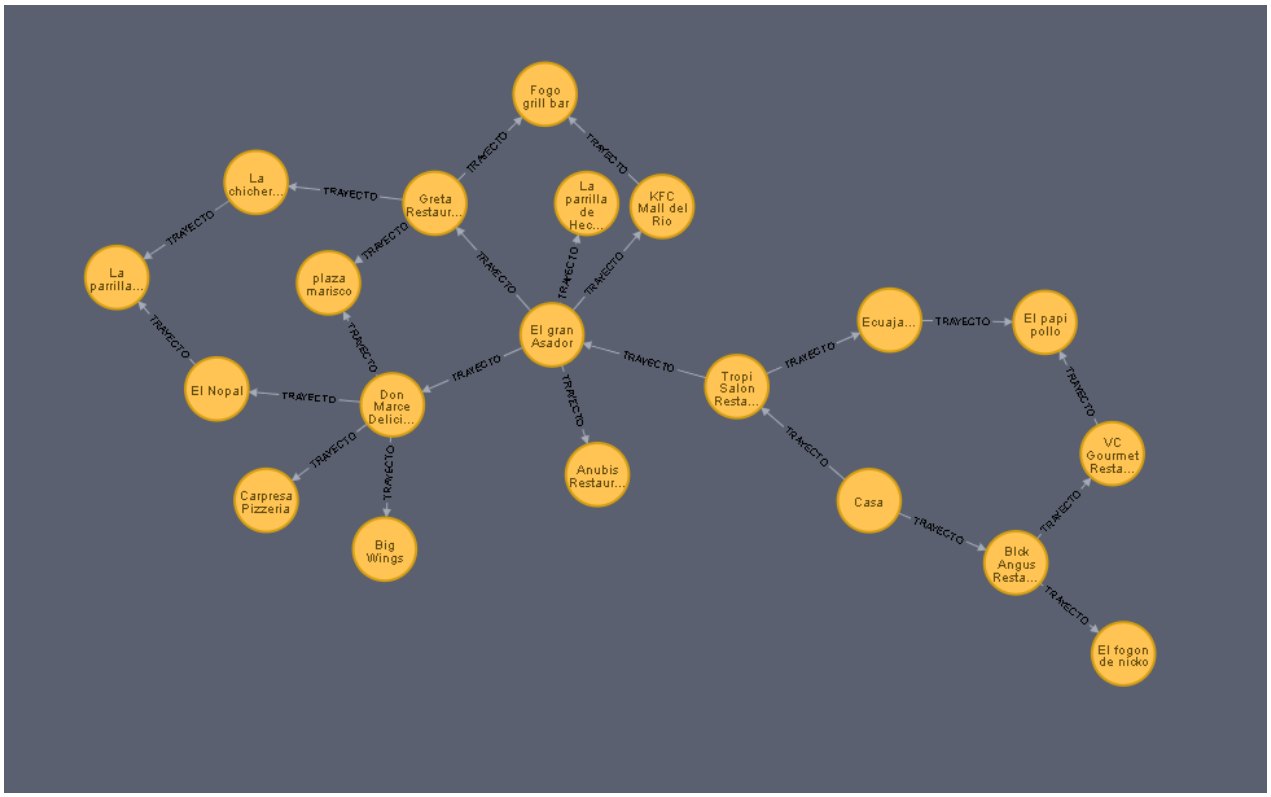


Desarrollo en Neo4j.

Para la prueba en neo4j procedemos a realizar la prueba con el algoritmo de Shortd path en donde se considera la ruta con menor coste. Primero procedemos a crear los respectivos nodos con sus relaciones y costos.

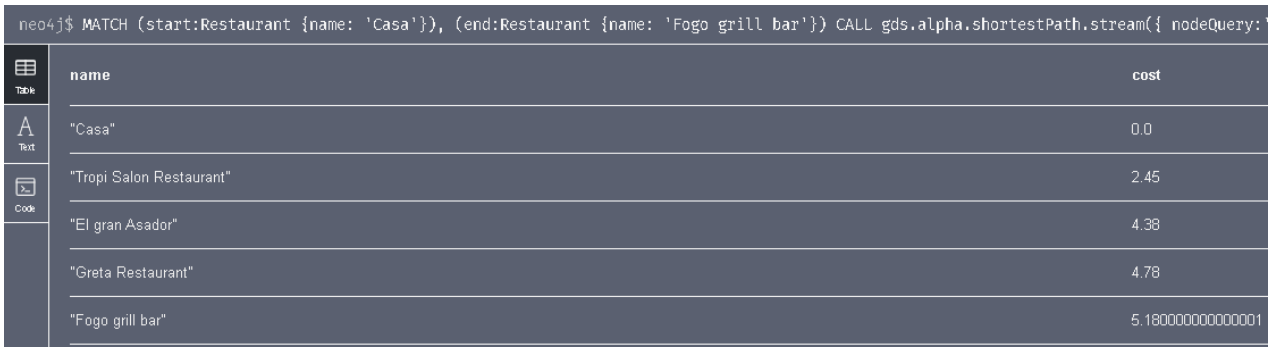
```
CREATE (a:Restaurant {name: 'Casa'}),
       (b:Restaurant {name: 'Blck Angus Restaurant'}),
       (c:Restaurant {name: 'Tropi Salon Restaurant'}),
       (d:Restaurant {name: 'El fogon de nicko'}),
       (e:Restaurant {name: 'VC Gourmet Restaurant'}),
       (f:Restaurant {name: 'Ecuajama Restaurant'}),
       (g:Restaurant {name: 'El gran Asador'}),
       (h:Restaurant {name: 'El papi pollo'}),
       (i:Restaurant {name: 'Anubis Restaurant'}),
       (j:Restaurant {name: 'KFC Mall del Rio'}),
       (k:Restaurant {name: 'Greta Restaurant'}),
       (l:Restaurant {name: 'Don Marce Delicious food'}),
       (m:Restaurant {name: 'La parrilla de Hector'}),
       (n:Restaurant {name: 'Fogo grill bar'}),
       (o:Restaurant {name: 'plaza marisco'}),
       (p:Restaurant {name: 'La chicheria'}),
       (q:Restaurant {name: 'Carpresa Pizzeria Restaurant'}),
       (r:Restaurant {name: 'El Nopal'}),
       (s:Restaurant {name: 'Big Wings'}),
       (t:Restaurant {name: 'La parrillada'}),
       (a)-[:TRAYECTO {cost: 3.56}]->(b),
       (a)-[:TRAYECTO {cost: 2.45}]->(c),
       (b)-[:TRAYECTO {cost: 3.07}]->(d),
       (b)-[:TRAYECTO {cost: 3.07}]->(e),
       (e)-[:TRAYECTO {cost: 2.86}]->(h),
       (c)-[:TRAYECTO {cost: 2.28}]->(f),
       (c)-[:TRAYECTO {cost: 1.93}]->(g),
       (f)-[:TRAYECTO {cost: 2.86}]->(h),
       (g)-[:TRAYECTO {cost: 1.41}]->(i),
       (g)-[:TRAYECTO {cost: 0.45}]->(j),
```

```
(g)-[:TRAYECTO {cost: 0.40}]->(k),
(g)-[:TRAYECTO {cost: 0.87}]->(l),
(g)-[:TRAYECTO {cost: 1.25}]->(m),
(j)-[:TRAYECTO {cost: 0.45}]->(n),
(k)-[:TRAYECTO {cost: 0.40}]->(n),
(k)-[:TRAYECTO {cost: 3.65}]->(o),
(k)-[:TRAYECTO {cost: 1.10}]->(p),
(l)-[:TRAYECTO {cost: 3.65}]->(o),
(l)-[:TRAYECTO {cost: 1.91}]->(q),
(l)-[:TRAYECTO {cost: 1.53}]->(r),
(l)-[:TRAYECTO {cost: 1.41}]->(s),
(p)-[:TRAYECTO {cost: 1.70}]->(t),
(r)-[:TRAYECTO {cost: 1.70}]->(t)
```



Para realizar la búsqueda en neo5j procedemos a realizar con el siguiente código en cypher el mismo que nos permite calcular el coste existente entre el nodo inicial y el nodo meta.

```
MATCH (start:Restaurant {name: 'Casa'}), (end:Restaurant {name:
'Fogo grill bar'})

CALL gds.alpha.shortestPath.stream({

  nodeQuery:'MATCH(n:Restaurant) WHERE NOT n.name = "c" RETURN
id(n) AS id',

  relationshipQuery:'MATCH(n:Restaurant)-[r:TRAYECTO]-
>(m:Restaurant) RETURN id(n) AS source, id(m) AS target, r.cost
AS weight',

  startNode: start,

  endNode: end,

  relationshipWeightProperty: 'weight',

  writeProperty: 'sssp'

})

YIELD nodeId, cost

RETURN  gds.util.asNode(nodeId).name AS name, cost
```



name	cost
"Casa"	0.0
"Tropi Salon Restaurant"	2.45
"El gran Asador"	4.38
"Greta Restaurant"	4.78
"Fogo grill bar"	5.180000000000001

Conclusión.

El desarrollo del método por ascenso de colinas podemos observar el calculo el cual considera la solución con el menor coste, el cual se puede ver reflejado en el algoritmo de neo4j. El algoritmo de ascenso de colina considera el menor coste, pero no el mas óptimo.