Sistema de Recomendación

Malki Yupanki, Alex Benavidez
Sistemas, Universidad Politécnica Salesiana, UPS
Cuenca, Ecuador
abenavideza@est.ups.edu.ec
myupanki@est.ups.edu.ec

The present work is to develop an application of recommendation of places in the city of Quito. Various search algorithms and recommendations are used to have a greater degree of acceptance of the recommendation..

***Index Terms*—Neo4j, Algoritmos, Busqueda.**

I. Introducción

EL ecuador es un país que tiene varias ciudades principales y diferentes lugares turísticos, Hoy en día se puede acceder a diferentes sitios web que nos muestran diferentes lugares, pero es difícil poder tener una certeza de la distancia entre los diferentes lugares que se quiere visitar o se tienen disponibles. Para ello diseñamos un sistema de recomendación de parques, iglesias, hospitales, bomberos, policía, escuelas, centros médicos, lugares turísticos, centros de estimularon temprana y centros educativos para el desarrollo de niños. Este sistema nos permitirá devolver los lugares con mas accesos posibles, al igual que la ruta mas corta entre ellos. Este sistema nos permitirá encontrar los lugares mas cercanos y posibles lugares que nos podrían interesar por visitar.

II. Desarrollo

A. Metodología

Para el desarrollo del sistema de recomendación se procedió a realizar un estudio e investigación de los algoritmos que se implementaran para brindar una recomendación en base a la información de los diferentes lugares de la ciudad de Quito. Los algoritmos que se van a utilizar los podemos encontrar en la pagina oficial de neo4j. Al igual que que es el lugar de donde se procederá a descargar la base orientada a grafos neo4j la cual nos permitirá alcacer la información de los lugares de la ciudad para poder luego a traves de diferentes algoritmos acceder a ella desde Java. El uso básico de neo4j se puede encontrar en el siguiente link https://printregrator2020.blogspot.com/p/planteamiento-y-descripcion-del-problema.html?fbclid=IwAR153eqpU6g64T-QIpqrhC-fLLyxZKx1GrYA5waDbgyqUDgVOlQtqdfnyQg.
Despues de ver el uso de neo4j procedemos a investigar acerca de los 5 algoritmos que nos permitirán desarrollar nuestro sistema de recomendacion.

1) Cercanía Centralidad

Estos tipos de algoritmo hacen referencia a aquellos nodos con una posición mas central, es decir que tienen una acceso mas fácil dentro del árbol de nodos o grafo. Este tipo de algoritmo sirve para tener un acceso mas rapito y fácil en cuestión de flujo dentro del árbol de nodos. Cercanía (closeness) es el promedio de las distancias del vértice a todos los demás nodos. La medida de centralidad por cercanía debe tener mayor valor para nodos más centrales y por esto se considerarán los inversos del promedio de las distancias para su definición, como se observa en la ecuación

CC(x) =N-1n / d(y,x)

Esta ecuación representa el calculo de la cercanía centralidad donde N es el numero de nodos y se divide para la Sumatoria de la distancia entre los demás nodos .

2) Ruta mas corta

Los algoritmos de Búsqueda de ruta son aquellos que implementan un busque da un camino, entre un nodo inicial y un nodo final o destino. Hay varios algoritmos de búsqueda de ruta entre ellos tenemos al algoritmo de búsqueda de ruta mas corta, este algoritmo implementa una búsqueda para encontrar un camino con el menor coste en distancia entre su nodo origen y final.

El algoritmo de búsqueda de la ruta mas corta tiene una gran historia ya que en el siglo XIX se desarrollaron algoritmos para encontrar un ruta eficiente para aplicarlos en diferentes casos. En 1956 el científico Dijkstra se encontraba en la necesidad de realizar la demostración de las nuevas computadoras ARMAC. El decidió establacerse una problema para realizar la demostración de las computadoras hacia las personas comunes, que no tenian un grado de conociminiento sobre la Informática. Para ello se estableció el algoritmo de Dijkstra en cual pudo implementar en un mapa de transportes de 64 ciudades.

3) Vecinos Comunes

Este tipo de algoritmo nos ayuda a determinar la cercanía que existe entre 2 nodos de un árbol o grafo. Dentro de ellos existen varios algoritmos uno de ellos es el Algoritmo de vecinos comunes el cual nos permite determinar elementos en comunes entre dos nodos, Esto expresa entre cuanta posibilidad hay que un nodo pueda comunicase con otro de acuerdo a los nodos hijos que tenga.

4) Algoritmo de vecinos mas cercanos aproximados (ANN)

El Algoritmo Approximate Nearest Neighbors construye un gráfico de vecinos más cercanos para un conjunto de objetos basado en un algoritmo de similitud proporcionado. La similitud de los elementos se calcula en función de la similitud de Jaccard,la similitud de coseno, la distancia euclidiana la similitud de Persona.

Podemos usar el algoritmo Approximate Nearest Nearest Neighbors para averiguar los elementos k más similares entre sí. El gráfico de vecinos más cercanos se puede utilizar como parte de las consultas de recomendación.

5) Algoritmo de Propagación de etiquetas

El algoritmo de propagación de etiquetas (LPA) es un algoritmo rápido para encontrar comunidades en un gráfico. Detecta estas comunidades utilizando la estructura de la red por sí sola como su guía, y no requiere una función objetiva predefinida o información previa sobre las comunidades. Una característica interesante de LPA es que a los nodos se les pueden asignar etiquetas preliminares para reducir la gama de soluciones generadas. Esto significa que se puede utilizar como forma semi-supervisada de encontrar comunidades donde elegimos a mano algunas comunidades iniciales.

B. Descripción del problema

Para una persona que llega de turista hacia la ciudad de quito, por vía terrestres, en ocasiones se le dificulta tener acceso a información de los lugares mas cercanos que podría visitar o a la distancia que se encuentran los mismos. Para brindar una recomendación se implemento un sistema que nos permite brindar información acerca de los lugares a los que podría acudir o visitar. La ciudad no cuenta con personas que puedan brindar una ascesoria acerca de los lugares que tiene la ciudad. Las personas aveces no cuentan con un grado de explicación aceptable de la distancia a la que se encuentran los lugares o cuales son aquellos que se encuentran en una ubicación de fácil acceso.

C. Propuesta de solución

Para poder brindar una recomendación acertada en base a datos específicos y reales se procedió a desarrollar un Sistema en Java implementando la base de datos orientada a grafos neo4j. Este sistema nos permitirá recomendar diferentes lugares de la ciudad de quito en base a los 5 algoritmos anteriormente descritos. Para poder construir el sistema se procedió a recopilar información de los lugares que mas pueden visitar una persona o necesitar. Estos lugares son:

- parques,
- iglesias,
- hospitales,
- bomberos,
- policía,
- escuelas,
- centros médicos,
- lugares turísticos,
- centros de estimularon temprana y
- centros educativos para el desarrollo de niños

Para obtener información de los diferentes lugares de la ciudad procedemos a utilizar la herramienta My maps, esta herramienta nos permite diseñar mapas y establecer puntos que nos permitirán graficar las relaciones entre los diferentes lugares. La información recopilada son registradas en archivos de datos .csv, estos archivos seran cargados en nuestra base de datos neo4j para poder luego en base a los algoritmos procesar los datos y obtener datos para brindar información y recomendaciones.

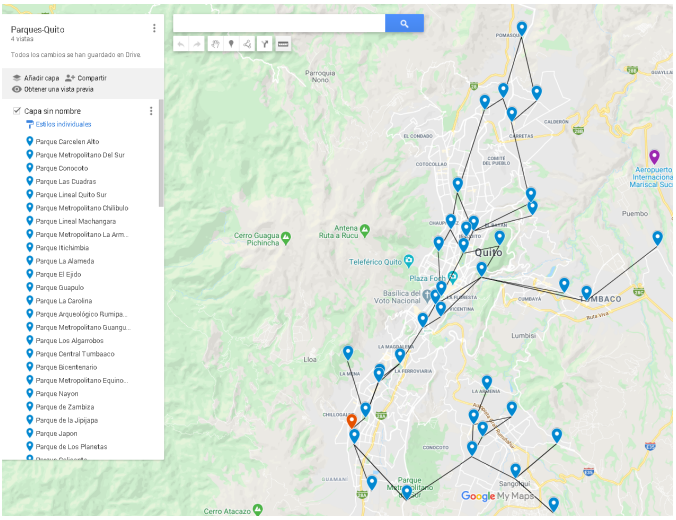

Fig. 1. Mapa de parques quito y sus relaciones

1) Diagrama esquemático

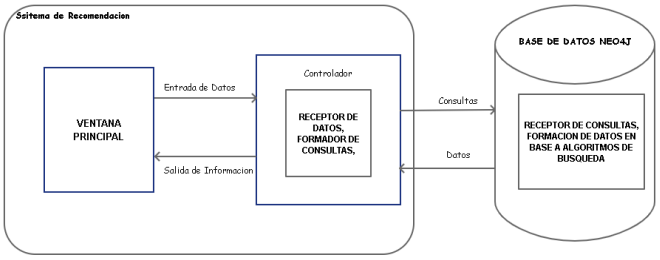

Fig. 2. Diagrama Esquemático del Sistema

Nuestro sistema cuenta con una ventana principal encargada de recopilar las datos seleccionados por el usuario, se envían los datos hacia el controlador para que se formen las consultas respectivas. El controlador realiza una conexión a la base de datos neo4j y realiza las consultas implementado algoritmos específicos para cada caso. La base de datos neo4j se encarga de realizar las consultas y devolver los datos obtenidos. El controlador se encarga de recopilar los datos y presentarlos en la ventana para el usuario.

D. Arquitectura del sistema de simulación

Los datos recopilados serán procesado en la base de datos neo4j. En esta base se procederán a ingresar todos los datos recopilados en nuestros archivos .csv. Estos datos serán procesados por la Base de datos de a cuerdo a las diferentes consulta que se realizan. Para ello las consultas enviados a la base deberán contar con un formato especifico de acuerdo al algoritmo a implementarse. El ingreso de los nodos desde los archivos .csv se realizaran a traves de comandos de carga de archivos. Una vez subido los datos a nuestra base de datos

con las respectivas relaciones y atributos asignados a cada nodo, se podra visualizar una base que contiene grafos con las propiedades de cada lugar del cual se obtuvo información.

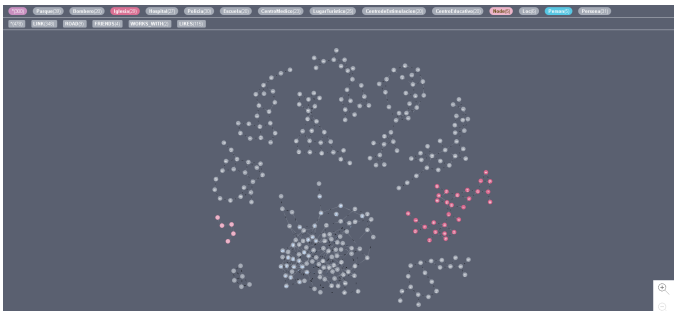

Fig. 3. Grafos en la Base Neo4j

La Base de datos orienta a grafos de neo4j almacena la información de los diferentes lugares de la ciudad de quito para poder ser analizados y procesados en base a consultas enviadas a través de una conexión establecida entre la aplicación y la base de datos. Cada consulta permite implementar un algoritmo que permite resolver diferentes problemas como ver el lugar central mejor conectado entre los nodos, devolver una ruta con un costo menor para poder desplasarce de un punto hacia otro punto, ver los lugares en común conectados entre si, Determinar los lugares que le pueden gustar a la persona en base a sus gusto y observar una comunidad de lugares que se encuentran relacionadas entre ellos.

1) Descripción de la solución

Para poder dar una recomendación en función a los datos recopilados y almacenados en nuestra base de datos primero procedemos a diseñar una ventana que nos permita realizar la recopilación de la Información respeto a los lugares que pueden ser elegidos por el usuario en base a sus gustos. Estos datos previamente almacenados servirán para poder realizar recomendaciones y cálculos de ruta en función de la elección dele usuario. Un turista que se desplaza por vía terrestre, que es lo mas usual en el ecuador divido al bajo costo de transporte, El sistema recomendara los lugares tomando como punto de partida el Terminal terrestre de Quitumbe. Desde el lugar de inicio se podrá desplazar a diferentes lugares de la ciudad y que se encuentran almacenados en nuestra base de Datos neo4j. Para el sistema se procederá a realizar en un proyecto de tipo Maven que nos permitirá descargar las respectivas librerías para poder construir las sentencias y métodos de consulta hacia la base neo4j.

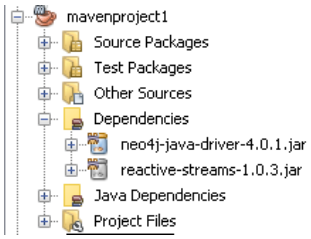

Fig. 4. Proyecto tipo Maven

Para recomendar un lugar central y el mejor conectado se utilizara el algoritmo de cercanía centralidad para poder devolver el nodo que se encuentra mejor ubicado. El algoritmo nos devuelve valores entre 0 y 1, donde mientras mas se aproxime a 1, es el lugar que mejor centrado se encuentra en el grafo construido. El algoritmo nos retorna una lista que contienen todos aquellos nodos y su centralidad dentro del grafo, este valor mientras mas cercano a uno es, mejor ubicado estará el nodo. Gracias a este calculo de cercanía centralidad podemos recomendar los 4 mejores lugares con mayor centralidad.

```
public List<Lugar> cercaniaCentralidadLugar(String lugar) {
    List<Lugar> list = new ArrayList<>();
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                Result res = tx.run("CALL gds.alpha.closeness.stream"
                        + "({nodeProjection: '" + lugar + "',relationshipProjection: 'LINK'})YIELD nodeId, centrality "
                        + "RETURN gds.util.asNode(nodeId).name AS user, centrality");
                for (Record r : res.list()) {
                    Lugar p = new Lugar();
                    p.setNombre(r.get(0).asString());
                    p.setCentralidad(r.get("centrality").asDouble());
                    list.add(p);
                }
                return null;
            }
        });
    }

    return list;
}
```

Fig. 5. Algoritmo de Cercanía Centralidad

Nuestro sistema nos devuelve una recomendación de un lugar el cual se encuentra mejor ubicado, Desde el lugar recomendado podemos calcular la ruta existente entre los el lugar de origen y el lugar destino, el cual sera nuestro lugar recomendado. Nuestro sistema permitirá calcular la ruta existente entre los diferentes que tenemos en nuestra base de datos neo4j. Para poder realizar el calculo de una ruta eficiente utilizamos el Algoritmo de la Ruta mas Corta el cual nos permitirá obtener la distancia entre los nodos origen y destino. Nuestro algoritmo en el sistema de recomendación nos pide el nodo origen y nodo destino al igual que el tipo de lugar que vamos a realizar el calculo.

```
public List<Lugar> costoUniformeLugar(String clase, String lugar1, String lugar2) {
    List<Lugar> list = new ArrayList<>();
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                Result res = tx.run("MATCH (start:" + clase + " {name: '" + lugar1 + "'}), "
                        + "(end:" + clase + " {name: '" + lugar2 + "'})"
                        + "CALL gds.alpha.shortestPath.stream({"
                        + "  nodeProjection: '" + clase + "',"
                        + "  relationshipProjection: {"
                        + "    ROAD: {"
                        + "      type: 'LINK',"
                        + "      properties: 'cost',"
                        + "      orientation: 'UNDIRECTED'"
                        + "    }"
                        + "  },"
                        + "  startNode: start,"
                        + "  endNode: end,"
                        + "  relationshipWeightProperty: 'cost'"
                        + "})"
                        + "YIELD nodeId, cost "
                        + "RETURN gds.util.asNode(nodeId).name AS name, cost");
                for (Record r : res.list()) {
                    Lugar p = new Lugar();
                    p.setNombre(r.get(0).asString());
                    p.setCosto(r.get("cost").asDouble());
                    list.add(p);
                }
                return null;
            }
        });
    }

    return list;
}
```

Fig. 6. Algoritmo de Costo Uniforme

Gracias al lugar mejor ubicado podremos saber cuales son aquellos nodos con los que tiene alguna similaridad gracias a sus nodos hijos. Para determinar aquellos nodos cercanos utilizamos el algoritmo de Vecinos Comunes, este algoritmo nos permite obtener aquellos nodos comunes entre un nodo en especifico. Nuestro algoritmo nos permite calcular la similaridad de los diferentes lugares almacenados de nuestra base de datos y un lugar seleccionado por el usuario, el algoritmo nos pide como parámetro el lugar del cual se desea saber los vecinos comunes y el tipo de lugar del cual se desea analizar de nuestra base de datos.

```
public Lugar vecinosComunes(String clase, String lugar1, String lugar2) {
    Lugar lugar= new Lugar();
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                Result res = tx.run("MATCH (p1:"+clase+" {name: '"+lugar1+"'})"
                        + " MATCH (p2:"+clase+" {name: '"+lugar2+"'})"
                        + " RETURN gds.alpha.linkprediction.commonNeighbors(p1, p2) AS score");
                int num = res.single().get(0).asInt();
                lugar.setVecino(num);


                return null;
            }
        });
    }
    return lugar;
}
```

Fig. 7. Algoritmo Vecinos Comunes

Con la utilización del algoritmo de propagación de etiquetas podremos saber como están formados los nodos en base a comunidades que este algoritmo nos permite crear, con la utilización de este algoritmo se podrá visualizar los nodos que pertenecen a la misma comunidad o que se encuentren mas cercanos. Como podremos observar el algoritmo nos pide como parámetro, el cual es el nombre del lugar del cual queremos obtener los lugares que estén asignados a la misma comunidad.

```
public List<LugarComunidad> propagacionDeEtiquetas(String grafico) {
    List<LugarComunidad> lugaresComunidad = new ArrayList<>();
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                //Result result = tx.run("MATCH(n:Node)-[r:REL]->(d:Node) return n.tag AS m, c
                Result res = tx.run("CALL gds.labelPropagation.stream('" + grafico + "') "
                        + "YIELD nodeId, communityId AS Community "
                        + "RETURN gds.util.asNode(nodeId).name AS name, Community ");

                for (Record r : res.list()) {
                    LugarComunidad lugarComunidad = new LugarComunidad();
                    lugarComunidad.setNombre(r.get(0).asString());
                    lugarComunidad.setCominidad(r.get("Community").asInt());
                    lugaresComunidad.add(lugarComunidad);
                }
                return null;
            }

            // System.out.println(greeting);
        });
    }
    return lugaresComunidad;
}
```

Fig. 8. Algoritmo de Propagación de Etiquetas

Con la utilización del algoritmo de vecino mas cercanos aproximados se podrá realizar una sugerencia al usuario de los lugares turísticos que le puede gustar en base a los gustos de otros usuarios, en donde se vera la similaridad que tienen en los gustos los usuarios y en base a esa similitud se procederá a sugerir lugares a nuevos usuarios. Con el siguiente método implementado me permitirá crear las relaciones de los usuarios con los lugares turísticos que sean de su agrado, en donde nos pide como parámetro el nombre de la persona y el tipo de lugar que en este caso será un lugar turístico.

```
public List<String> comprovarANN(String nombrePersona, String tipo) {
    List<String> lugares = new ArrayList<>();
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                Result res2 = tx.run("match (n:Persona{name:'" + nombrePersona + "'})-[:LIKES]->(p:" + tipo + ") return p.name AS nombre");
                if (res2.list().isEmpty()) {
                    System.out.println("NO TIENE RELACIONES ESTA PERSONA");
                } else {
                    for (Record r : res2.list()) {
                        String nombre = r.get("nombre").asString();
                        lugares.add(nombre);
                    }
                }
                return null;
            }
            // System.out.println(greeting);
        });
    }
    return lugares;
}
```

Fig. 9. Algoritmo ANN

Con el siguiente método me permitirá buscar los usuarios mas similares de la persona que se ingresa a la base, para asi poder obtener los lugares turísticos que se le recomendara al usuario

```
public void similarity(){
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                Result res2 = tx.run("MATCH (p:Persona)-[:LIKES]->(lugaresturisticos)\n" +
                                     " WITH {itemid:id(p), categories: collect(id(lugaresturisticos))} AS userData\n" +
                                     " WITH collect(userData) AS data\n" +
                                     " CALL gds.alpha.ml.ann.write({\n" +
                                     "  nodeProjection: '*',\n" +
                                     "  relationshipProjection: '*',\n" +
                                     "  algorithm: 'jaccard',\n" +
                                     "  data: data,\n" +
                                     "  similarityCutoff: 0.1,\n" +
                                     "  showComputations: true,\n" +
                                     "  concurrency: 1\n" +
                                     " })\n" +
                                     " YIELD nodes, similarityPairs, writeRelationshipType, writeProperty, min, max, mean, p95\n" +
                                     " RETURN nodes, similarityPairs, writeRelationshipType, writeProperty, min, max, mean, p95");
                return null;

            }
            // System.out.println(greeting);
        });
    }
}
```

Fig. 10. Algoritmo ANN

Finalmente con el método siguiente obtener las lugares turísticos que se le recomendara al usuario, en donde pasamos como parámetro el nombre de la persona que se le recomendara los lugares turísticos.

```
public List<String> recomendaciones(String nombrePersona){
    List<String> lugares = new ArrayList<>();
    try (Session session = driver.session()) {
        String greeting = session.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                Result res2 = tx.run("MATCH (p:Persona {name:'"+nombrePersona+"'})-[:SIMILAR]->(other),\n" +
                                     "      (other)-[:LIKES]->(lugarturistico)\n" +
                                     " WHERE not((p)-[:LIKES]->(lugarturistico))\n" +
                                     " RETURN lugarturistico.name AS lugarturistico, count(*) AS count\n" +
                                     " ORDER BY count DESC ");

                    for (Record r : res2.list()) {
                        String nombre = r.get("lugarturistico").asString();
                        System.out.println(nombre);
                        lugares.add(nombre);
                    }

                return null;

            }
            // System.out.println(greeting);
        });
    }
    return lugares;
```

Fig. 11. Algoritmo ANN

E. Resultados

En el menú de recomendación tenemos la ventan de ver el lugar con mejor acceso, esta ventana nos permite seleccionar el lugar que se desea buscar entre los 10 diferentes sitios que se tiene. El resultado que nos devuelve es el calculo realizado por el algoritmo de cercanía centralidad en donde nos devuelve los lugares mejor ubicados dentro del grafo. En la ventana se puede visualizar los 4 mejores lugares donde el primero es aquel lugar con mayor centralidad calculado por el algoritmo y el que se recomienda visitar.

Fig. 12. Ventana de Recomendación

La ventana de la ruta mas corta nos devuelve los lugares que se encuentran cerca y por los que se puede pasar o fijar una ruta con una distancia óptima hacia el destino, para determinar el camino mas óptimo empleamos el algoritmo de costo uniforme el cual nos devuelve el resultado entre los lugares que ingresemos como origen y destino.

Fig. 13. Ventana de Recorrido

En la ventana de recomendación del lugar con mejor acceso podemos ir hacia los lugares mas cercanos que se pueden encontrar al lugar con mejor acceso, para ello utilizamos el algoritmo de Vecinos comunes el cual nos permite determinar aquellos lugares que tienen nodos en común entre el lugar seleccionado o recomendado. Al usuario se le presenta los lugares recomendados cercanos a lugar en el que se encuentra.

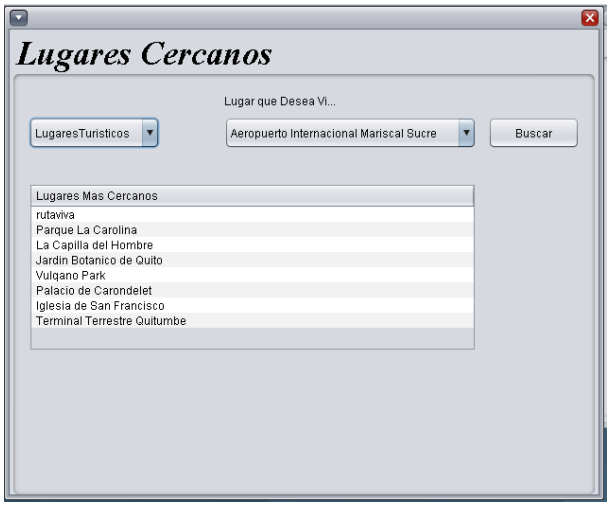

Fig. 14. Ventana de Lugares Cercanos

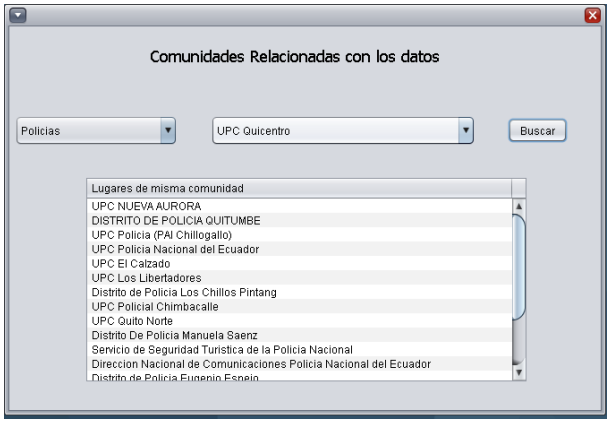

Fig. 15. Ventana Comunidades

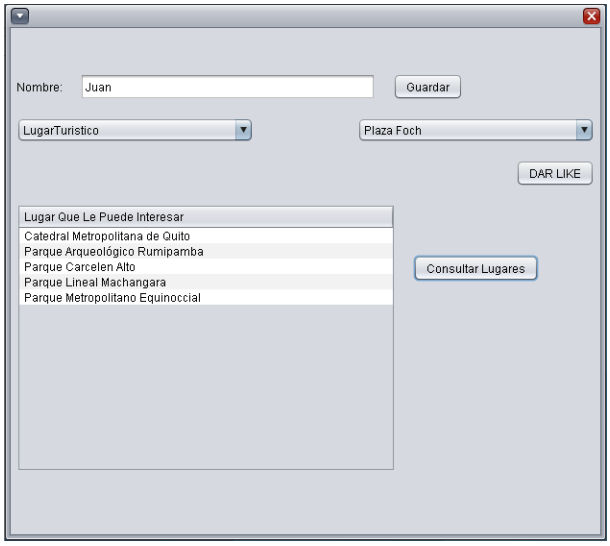

Fig. 16. Ventana Recomendación Lugares

III. Conclusiones y Recomendaciones

Luego de desarrollar el sistema se puede resolver ciertos problemas puntuales que se puede encontrar una persona o

turista al llegar a la ciudad de quito. El uso de la base de datos neo4j orientada a grafos nos permite realizar diferentes algoritmos para la resolución de problemas, es una base que ayuda a almacenar datos para poder trabajar de una manera mas rápida y fácil a la hora de realizar consultas, al igual que la implementación con el lenguaje de Java, su uso puede tener varias aplicaciones dentro de la resolución de diferentes problemas que se puedan dar. Para el uso y acceso de los diferentes algoritmos, los métodos aplicados y sus resultados desde la base de datos neo4j, se recomienda utilizar algoritmos que se encuentran en la pagina oficial de neo4j sección 5.0, allí se puede encontrar los algoritmos orientados a diferentes tipos de búsquedas, para la implementación junto con el lenguaje de programación Java, es necesario usar un proyecto de tipo Maven para poder realizar la integración de las librerías necesarias para establecer una comunicación con la base de datos.

REFERENCES

[1] H. MEZA, F. Ortega. *GRAFOS Y ALGORITMOS*, 2nd ed. Colombia, Caracas: EDITORIAL EQUINOCCIO, 2004.

[2] F. Moreno. *Clasificadores eficaces basados en algoritmos rapidos de busqueda del vecino mas cercano.* 24 de febrero de 2004. Disponible en: https://rua.ua.es/dspace/bitstream/10045/11790/1/Moreno-Seco-Francisco.pdf

[3] A. Guzman,E. Arango,D. Jimenez. *Búsqueda de la ruta óptima mediante los algoritmos: genético y dijkstra utilizando mapas de visibilidad* . Agosto de 2012. Disponible en: http://revistas.utp.edu.co/index.php/revistaciencia/article/view/1635/4713