Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ingeniería Eléctrica
Ingeniería en Computación

Título de la tesis

Solución al Problema del Agente Viajero Utilizando el API de Google Maps y Sistema Android

Para obtener el Título de:
Ingeniero En Computación

Presenta:
Irvin Samuel Bedolla Mota
irvinsbm@gmail.com

Asesor:
Dr. Juan José Flores Romero
juanf@umich.com

09/Julio/2019

Dedicatoria

Dedicado mi Tesis a mis padres,
por su apoyo incondicional a lo largo de mi carrera.

A mi familia,
por creer en mí y apoyarme en todo momento.

A mi hija Paola,
por entender y tener paciencia al compartir su tiempo

Agradecimientos

¡Agradezco a mis padres y a mi familia por confiar en mí, y en especial a mi maestro y asesor el Dr. Juan José Flores Romero por su enseñanza, apoyo y consejos a lo largo de mi Carrera!

Resumen

El algoritmo es la secuencia de instrucciones mediante la cual podemos resolver un problema o cuestión. Los algoritmos cuasi-óptimos son una familia de algoritmos cuya meta es dar soluciones aproximadas a problemas generales de tipo NP, sin necesidad de recorrer todo el espacio de búsqueda.

Las heurísticas clásicas realizan una exploración limitada sobre el espacio de búsqueda y normalmente las soluciones producidas son buenas en poco tiempo. Su implementación es sencilla y son fácilmente adaptables a problemas del mundo real. Existen dos tipos de heurísticas: constructivas y de mejoramiento o de búsqueda, el algoritmo de Kruskal implementado en el presente trabajo se ubica en las segundas, ya que pertenece a la familia de algoritmos cuya meta es precisamente dar soluciones aproximadas a problemas generales de tipo NP.

El problema del agente viajero consiste en: dado un conjunto finito de puntos y el costo de viaje entre todos los pares de puntos, encontrar la forma más barata de visitar todos los puntos exactamente una vez, y volver al punto de partida. Para resolver este problema se apoya en el algoritmo de Kruskal y en las librerías de Google Maps, para representar las rutas a seguir y el tiempo estimado de llegada.
Para solucionar este problema, lo principal es tener todos los puntos a visitar, a continuación, con el algoritmo de Kruskal resuelve y crea un árbol n-ario, el cual se va a recorrer.

El resultado de este recorrido traza una ruta que se representa en google Maps con el recorrido de todos los puntos visitando cada uno de ellos y regresando al origen en el menor tiempo posible con el apoyo de Google Maps.

Palabras Clave

Complejidad NP, Algoritmo aproximado, Kruskal, Heurística, Tiempo de ejecución, Ejecución en tiempo real, JSON

Abstract

The deterministic algorithms are a family of algorithms whose goal is precisely to give approximate solutions to general NP type problems, without the need to go through the entire search space.

The classic heuristics perform a limited exploration on the search space and normally the solutions produced are good in a short time. Its implementation is simple and easily adaptable to real world problems.

There are two types of heuristics: constructive and improvement or search, the algorithm implemented is located in the second type, because it belongs to the family of algorithms whose goal is precisely to give approximate solutions to general NP type problems.

The Traveler agent consists of: given a finite set of cities and travel costs between all the pairs of cities, find the cheapest way to visit all the cities exactly once and return to the starting point.

To solve this problem we rely on the Google Maps Api's, to represent the routes to follow and the estimated time of arrival.

Keywords

Complexity Np, Deterministic algorithm and no Deterministic, Kruskal, Heuristic, Execution time, Real-time execution, JSON.

Índice general

Índice de figuras

Capitulo 1

Introducción

Esta tesis tiene algunos algunas aplicaciones para resolver problemas reales por ejemplo para los repartidores médicos los cuales tiene que visitar varios lugares durante el día, así como también las personas que se encargan de elaborar pedidos y tienen que visitar varios clientes durante todo el día, también para las unidades de traslado como UPS, DHAL, etc. así como para apoyo en clase de algoritmos.

Esta tesis consiste en determinar la ruta semi-óptima de un conjunto de puntos o destinos. Al elegir cada uno de los destinos a visitar se forma un grafo el cual va a resolver por medio del Algoritmo de Kruskal.

El algoritmo de Kruskal da como resultado un árbol de expansión mínima. Los destinos a su vez los representamos en un mapa de Google Maps para dar como resultado la ruta semi-óptima.

Una vez representada la ruta, se cuenta con todas las opciones que ofrece la API de Google, como el seguimiento de ruta, el tiempo estimado de llegada, el tráfico, etc.

1.1. Conceptos principales

Los conceptos presentados en esta sección son la base para el desarrollo de la presente tesis y serán utilizados en el resto del documento. Entre los conceptos principales se tiene que saber diferenciar, qué es un algoritmo determinista de uno que no lo es, la principal diferencia es que el determinista siempre regresa el mismo resultado ante las mismas

entradas, mientras que el otro puede no dar el mismo resultado. La complejidad es el tiempo y recursos que se requieren para que termine la ejecución.

1.1.1 Algoritmo determinista y no determinista

En ciencias de la computación, un algoritmo determinista es un algoritmo que, en términos informales, es completamente predictivo si se conocen sus entradas.
Dicho de otra forma, si se conocen las entradas del algoritmo siempre producirá la misma salida, y la máquina interna pasará por la misma secuencia de estados.
Este tipo de algoritmos ha sido el más estudiado durante la historia y por lo tanto resulta ser el tipo más familiar de los algoritmos, así como el más práctico ya que puede ejecutarse en las máquinas eficientemente.

En ciencias de la computación, un algoritmo no determinista es un algoritmo que con la misma entrada ofrece muchos posibles resultados, y por tanto no ofrece una solución única. No se puede saber de antemano cuál será el resultado de la ejecución de un algoritmo no determinista. Como se observa en la Figura 1.1

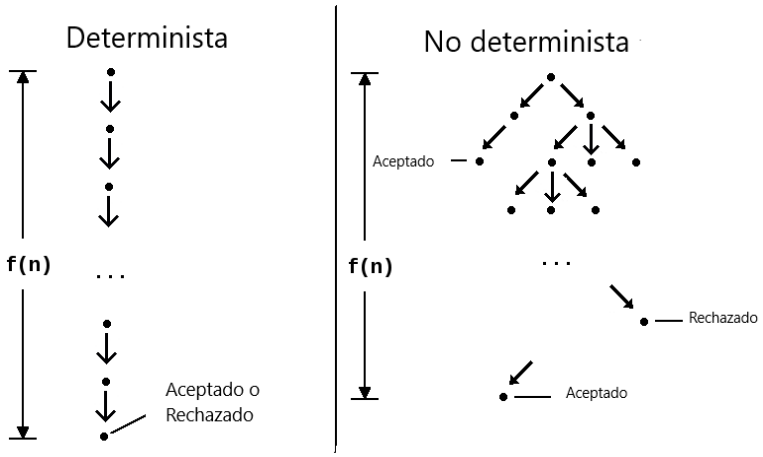


Figura 1. 1 Algoritmo determinista y no determinista

1.1.2 Complejidad NP

En teoría de la complejidad computacional, NP es el acrónimo en inglés de tiempo polinomial no determinista (Nondeterministic Polynomial time). Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista.
Los recursos comúnmente estudiados en complejidad computacional son:
– El tiempo: mediante una aproximación al número de pasos de ejecución que un algoritmo emplea para resolver un problema.

– El espacio: mediante una aproximación a la cantidad de memoria utilizada para resolver el problema.

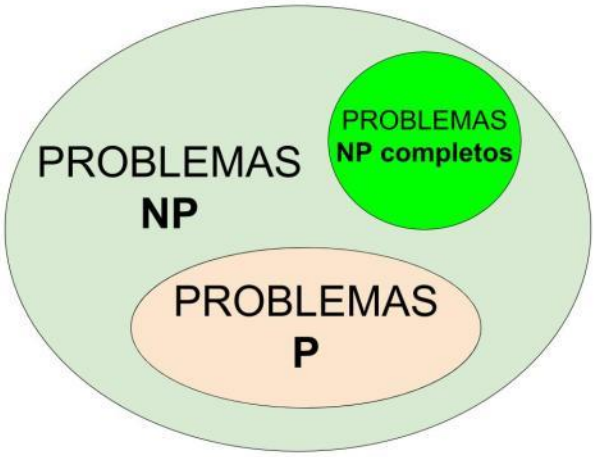


Figura 1. 2 Espacio de los problemas NP, P y NP completo

Clase Complejidad P, suele ser la clase de problemas computacionales que son "eficientemente resolubles" o "tratables", aunque haya clases potencialmente más grandes que también se consideran tratables. Aunque también existen problemas en P que no son tratables en términos prácticos; por ejemplo, unos requieren al menos $n^{1\,000\,000}$ operaciones. P es conocido por contener muchos problemas naturales, incluyendo las versiones de decisión de programa lineal, cálculo del máximo común divisor, y encontrar una correspondencia máxima.

En teoría de la complejidad computacional, la clase de complejidad NP-completo es el subconjunto de los problemas de decisión en NP tal que todo problema en NP se puede reducir en cualquier otro de los problemas en el conjunto NP-completo.

Como ejemplo de un problema NP-completo se tiene el problema de la suma de subconjuntos que se puede enunciar como sigue: dado un conjunto S de enteros, ¿existe un subconjunto no vacío de S cuyos elementos sumen cero? Es fácil verificar si una respuesta es correcta, pero no se conoce mejor solución que explorar todos los 2^{n-1} subconjuntos posibles hasta encontrar uno que cumpla con la condición. [1]. Por ejemplo, dado el conjunto { -7, -3, -2, 5, 8}, decimos que si tiene solución ya que al menos el subconjunto de {-3, -2, 5} da como resultado cero en la suma de todos sus números.

1.1.3 Árbol de Expansión Mínima

Un árbol de expansión es un árbol compuesto por todos los vértices y algunas (posiblemente todas) de las aristas de G. Al ser creado un árbol no existirán ciclos, además debe existir una ruta entre cada par de vértices.

Dado un grafo conexo, no dirigido y con pesos en las aristas, un árbol de expansión mínima es un árbol compuesto por todos los vértices y cuya suma de sus aristas es la de menor peso.
Grafo: Es una serie de puntos llamados nodos, unidos por arcos o aristas como se puede apreciar en la Figura 1.3.

Nodo: Elemento(s) que representan un punto u origen.

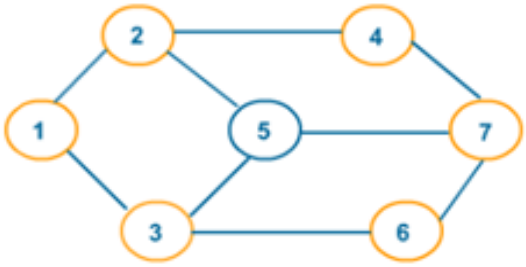


Figura 1. 3 Ejemplo Árbol de Expansión Mínimo

Red: Es un grafo con algún tipo de flujo en sus ramales, por ejemplo, la red Eléctrica o la red de transporte.

Cadena: Es una serie de elementos que van de un nodo a otro, por ejemplo, en el grafo de la Figura 1.3, una cadena es ir del nodo 1-2, luego del nodo 2 -5, y por ultimo del nodo 5 -7.

Ruta: Para la cadena anterior una serie es un conjunto de elementos que conforman una ruta, por ejemplo 1 -2 -5 -7, véase la figura 1.4.

Ciclo: Es la cadena que une un nodo consigo mismo, por ejemplo: 3-5, 5-2, 2-4, 4-7,7-6, 6-3. De esta manera el nodo de inicio también es el nodo final.

Grafo conectado: Es aquel en el cual todos los nodos están conectados, por ejemplo, el grafo anterior.

Arco orientado: Es aquel que tiene un sentido determinado, o sea, que tiene un nodo origen y un nodo destino, normalmente está representado por una flecha la cual indica el destino con la punta de la flecha.

Figura 1. 4 Ejemplo de Grafo

Gráfico orientado: Aquel en el cual todos sus arcos están orientados ver figura 1.5

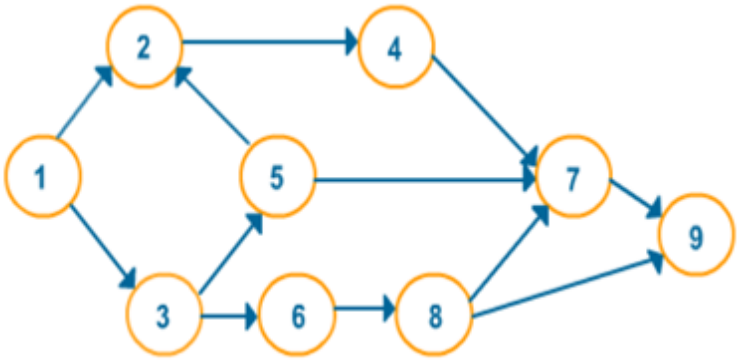


Figura 1. 5 Ejemplo Grafo Orientad

Nodo fuente: Es el nodo donde todos sus ramales están orientados hacia afuera, el nodo 1 es un nodo padre. Ya que va hacia el nodo 2 y 3 y no tiene ningún nodo que llegue a él.

Nodo receptor: Es en nodo donde todos sus arcos están orientados hacia él por ejemplo el nodo 9 es un nodo receptor ya que todos los nodos llegan a él y de este nodo no sale ningún arco.

1.1.4 Grafo

En matemáticas y ciencias de la computación, un grafo (del griego grafos: dibujo, imagen) es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre elementos de un conjunto.
Un grafo no dirigido o grafo propiamente dicho es un grafo G = (V, E).

Esto quiere decir que el Grafo no debe estar vacío, y además es un conjunto de pares no ordenados de elementos de V. Un par no ordenado es un conjunto de la forma { a, b}, de manera que { a, b} = {b, a}.
Para los grafos, estos conjuntos pertenecen al conjunto potencia de V, denotado P(V), y son de cardinalidad 2. La cardinalidad se refiere al número de elementos que contiene el grafo.

1.1.5 Algoritmo de Kruskal

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol de expansión mínimo en un grafo conexo y ponderado. Este algoritmo busca un subconjunto de aristas para formar un árbol que incluyen todos los vértices y donde la suma de pesos del total de todas las aristas del árbol es mínimo. Es decir, formar un sub-grafo que contenga sólo las aristas necesarias para visitar cada uno de los nodos.

Si el grafo no es conexo, entonces busca un árbol de expansión mínima. El algoritmo de Kruskal es un ejemplo de algoritmo voraz y es aquel que, para resolver un determinado problema, sigue una heurística consistente en elegir la mejor opción en cada paso local con la esperanza de llegar a una solución general óptima.

Esto quiere decir que el algoritmo elegido, resolviendo paso a paso y guardando las aristas para crear el árbol. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento. Normalmente se aplica a los problemas de optimización.

Dado un grafo conexo, no dirigido G, Un árbol de expansión es un árbol compuesto por todos los vértices y algunas (posiblemente de todas) las aristas de G.

Al ser creado un árbol no existirán ciclos, además debe existir una ruta entre cada par de vértices, es decir, genera el árbol de expansión mínimo el cual será utilizado para resolver el PAV (Problema del Agente Viajero).

Un grafo puede tener muchos árboles de expansión, veamos un ejemplo con grafo de la Figura 1.6

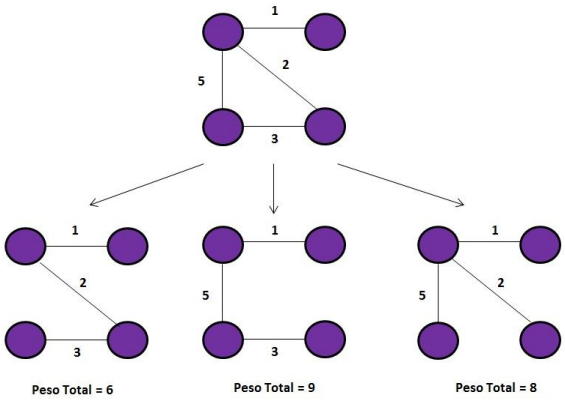


Figura 1. 6 Ejemplo recorrido de un grafo

En la figura anterior se puede observar que el grafo dado posee 3 árboles de expansión mínima, dichos arboles cumplen con las propiedades antes mencionadas como son unir todos los vértices usando algunas aristas.

1.1.6 Tiempo de ejecución

Se denomina tiempo de ejecución (runtime en inglés) al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.
Este tiempo se inicia con la puesta en memoria principal del programa, por lo que el sistema operativo comienza a ejecutar sus instrucciones.
El intervalo finaliza en el momento cuando éste programa envía al sistema operativo la señal de terminación, sea ésta una terminación normal,
en que el programa tuvo la posibilidad de concluir sus instrucciones satisfactoriamente, o una terminación anormal, en el que el programa produjo algún error y el sistema debió forzar su finalización.

1.1.7 Ejecución en tiempo real

Ejecución en tiempo real de un sistema informático que interacciona con su entorno físico y responde a los estímulos del entorno dentro de un plazo de tiempo determinado.
No basta con que las acciones del sistema sean correctas, sino que, además, tienen que ejecutarse dentro de un intervalo de tiempo determinado.

1.1.8 JSON

JSON es acrónimo de "JavaScript Object Notation", es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

1.2 Planteamiento del problema

El problema del agente viajero consiste en visitar todos los destinos y regresar al origen, visitando únicamente una vez cada uno de ellos y hacerlo con el menor costo, es decir; dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen? Este es un problema NP-completo.

1.3 Antecedentes

El problema fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiados. Es usado como prueba para muchos métodos de optimización.

Aunque el problema es computacionalmente complejo, una gran cantidad de heurísticas y métodos exactos son conocidos, y sirve para resolver desde unas cuantas ciudades hasta unos cientos.

El Problema del Agente Viajero PAV puede ser representado como un grafo no dirigido, de manera que las ciudades sean los vértices del grafo, los caminos son las aristas y las distancias de los caminos son los pesos de las aristas [2].

1.4 Objetivo de la tesis

Resolver de manera aproximada el problema del agente viajero el algoritmo de kruskal. Para resolver el PAV se requiere de varios nodos (destinos), una vez definidos se crea una matriz de adyacencia. Con dicha matriz se obtiene el origen, destino y costo de cada arista (todos contra todos) y obtenido la trayectoria o ruta se representa en Google Maps.

1.5 Descripción de capítulos

Capítulo 1.- Presentación del tema, Introducción y conceptos básicos.
Capítulo 2.- Desarrollo del tema, estrategias y métodos para resolver el PAV.
Capítulo 3.- Implementación del programa, ideas y resolución de problemas.
Capítulo 4.- Conclusiones y posibles modificaciones.
Capítulo 5.- Referencias.

Capítulo 2

2.1 Herramientas de apoyo

Este capítulo presenta un conjunto de herramientas que han sido usadas en la solución de problema de agente viajero, presentado en la tesis.
Lo primero a tomar en cuenta es que el problema tiene una complejidad exponencial, lo cual no es eficiente ni es aceptable.

2.2 Planteamiento del problema

Lo principal es hacer que le dé una solución, es decir asegurarse que termine y que lo haga dentro de un tiempo aceptable. El algoritmo de Kruskal consiste en obtener un árbol de expansión minino, el cual será recorrido para regresar la ruta a seguir.

Una vez resuelto, se decidió programar sobre el lenguaje de programación Java por varias razones. La primera y más importante fue que se requería de una aplicación Android y Android está basado en gran parte en Java. Por lo tanto, hacer una aplicación en Java no tiene tantos problemas de compatibilidad. Otra razón son las librerías que existen en Java, y por último el soporte que este lenguaje tiene dentro de Android Studio y sabiendo que Android tiene soporte directo con Google Maps, se tomó esa decisión para explotar al máximo el soporte que Google proporciona.

Por lo cual, se implementó un algoritmo aproximado para obtener la solución semi-óptima, la cual es aceptable ya que en la mayoría de sus resultados la solución es óptima.

2.2 Lenguajes de programación

2.2.1 Java

Es un lenguaje de programación de propósito general concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

El hecho de que sea orientado a objetos ayuda en la implementación de la aplicación, ya que Android Studio está basando en gran parte con java, por lo tanto, el código en gran parte es compatible.

Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados [3]

2.2.2 Lenguaje orientado a objetos

La primera característica de un lenguaje orientado a objetos se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. De esta manera los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "programa" (el código) y el "estado" (datos). Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema de software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos.

2.2.3 Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Android Studio está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Apps para Android.

Estuvo en etapa de vista previa de acceso temprano a partir de la versión 0.1, en mayo de 2013, y luego entró en etapa beta a partir de la versión 0.8, lanzada en junio de 2014. La primera compilación estable, la versión 1.0, fue lanzada en diciembre de 2014
Algunas características de Android Studio

- Integración de ProGuard y funciones de firma de aplicaciones.
- Renderizado en tiempo real
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en Gradle (es una herramienta para automatizar la construcción de proyectos android).
- Refactorización específica de Android y arreglos rápidos.
- Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programar aplicaciones para Android Wear.
- Soporte integrado para Google Cloud Platform, que permite la integración con Google Cloud Messaging y App Engine.
- Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones. [4]

2.2.4 Google Maps

Es un servidor de aplicaciones de mapas en la web que pertenece a Alphabet Inc. Ofrece imágenes de mapas desplazables, así como fotografías por satélite del mundo e incluso la ruta entre diferentes ubicaciones o imágenes a pie de calle con Google Street View. Existe una variante a nivel entorno de escritorio llamada Google Earth que ofrece Alphabet Inc. también de forma gratuita.

En 2014, los documentos filtrados por Edward Snowden revelaron que Google Maps es parte y víctima del entramado de vigilancia mundial operado por varias agencias de inteligencia occidentales y empresas tecnológicas.

Google Maps fue anunciado por primera vez en Google Blog el 8 de febrero de 2005. Originalmente soportaría solo a los usuarios de Internet Explorer y Mozilla Firefox, pero el soporte para Opera y Safari fue agregado el 25 de febrero de 2005.

El software estuvo en su fase beta durante seis meses, antes de convertirse en parte de Google Local, el 6 de octubre de 2005. Como en las aplicaciones web de Google, se usan un gran número de archivos Javascript para crear Google Maps.

Como el usuario puede mover el mapa, la visualización del mismo se baja desde el servidor. Cuando un usuario busca un negocio, la ubicación es marcada por un indicador en forma de pin, el cual es una imagen PNG transparente sobre el mapa.

Para lograr la conectividad sin sincronía con el servidor, Google aplicó el uso de AJAX dentro de esta aplicación. Es una aplicación para el desarrollo de mapas.

Google Maps se utiliza para representar en tiempo real la ubicación. También nos brinda todas las herramientas necesarias para representar la ruta, así como la manera de llegar a los distintos destinos. También nos brinda todas sus herramientas como el tiempo aproximando del viaje, el nivel de tráfico, la dirección a seguir, etc.

2.2.5 Obtención de una Clave para usar el API Google Maps

Para obtener los servicios de la Aplicación de Google Maps se tiene que obtener una clave desde la página https://console.developers.google.com/apis/credentials ver figura 2.1

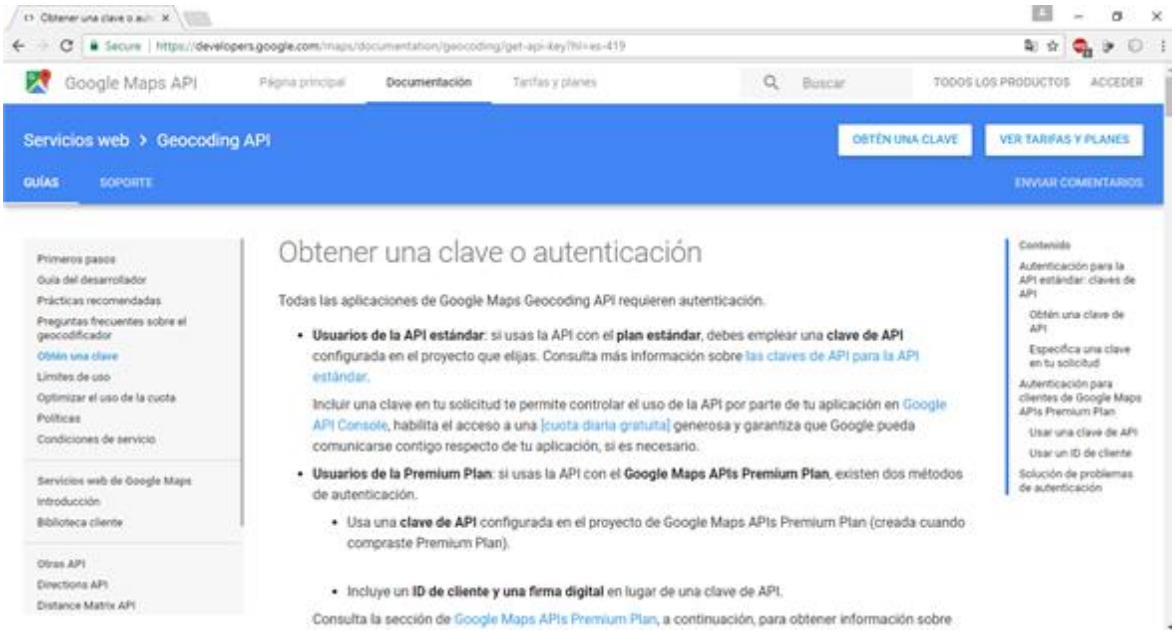


Figura 2. 1 Obtención de la clave para el API

La clave es gratuita para realizar prueba de la aplicación, se tiene que enlazar la aplicación realizada con el servicio de google. Para ello se tiene que seleccionar el proyecto que se va enlazar con el API de google ver figura 2.2.

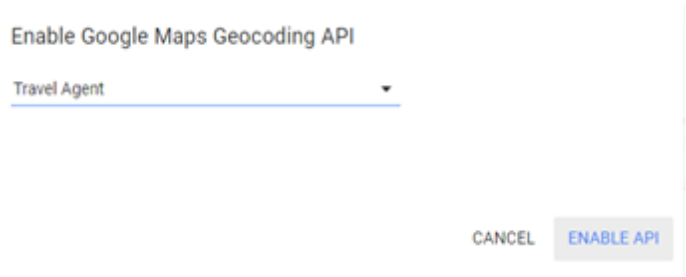


Figura 2. 2 Selección del Proyecto

Una vez habilitado, google te proporciona una clave única enlazada a tu proyecto ver figura 2.3.

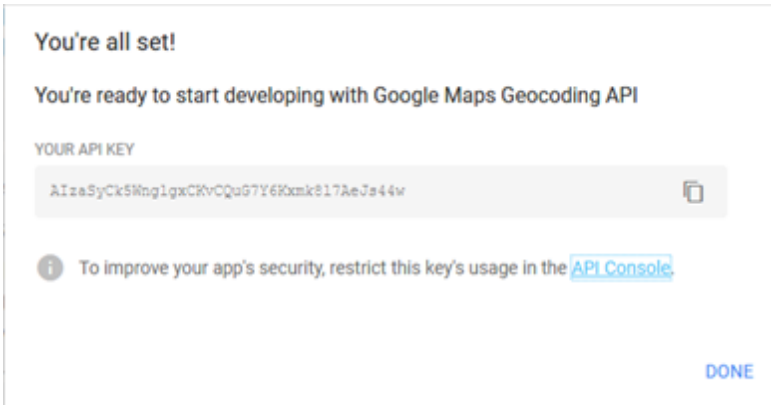


Figura 2. 3 Llave Google Maps.

Ya obtenida la clave se agrega la Huella Digital del certificado SHA-1 que genera nuestro proyecto a la clave que Genera Google Maps. Esto es necesario para que nuestro proyecto tenga los permisos de usar los servicios de Google ver figura 2.4.

Figura 2. 4 Enlazar proyecto a Google Maps

A continuación, se tienen que habilitar los servicios que se requieran del API de google; para nuestro caso es necesario habilitar las opciones de Google Maps ver figura 2.4.

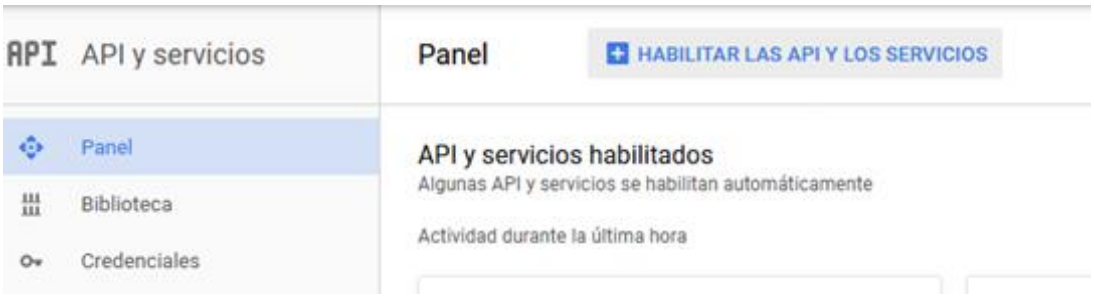


Figura 2. 5 Habilitar API

De las siguientes bibliotecas usaremos Google Maps Android API y Google Places API for Android que están dentro de Google Maps ver figura 2.6.

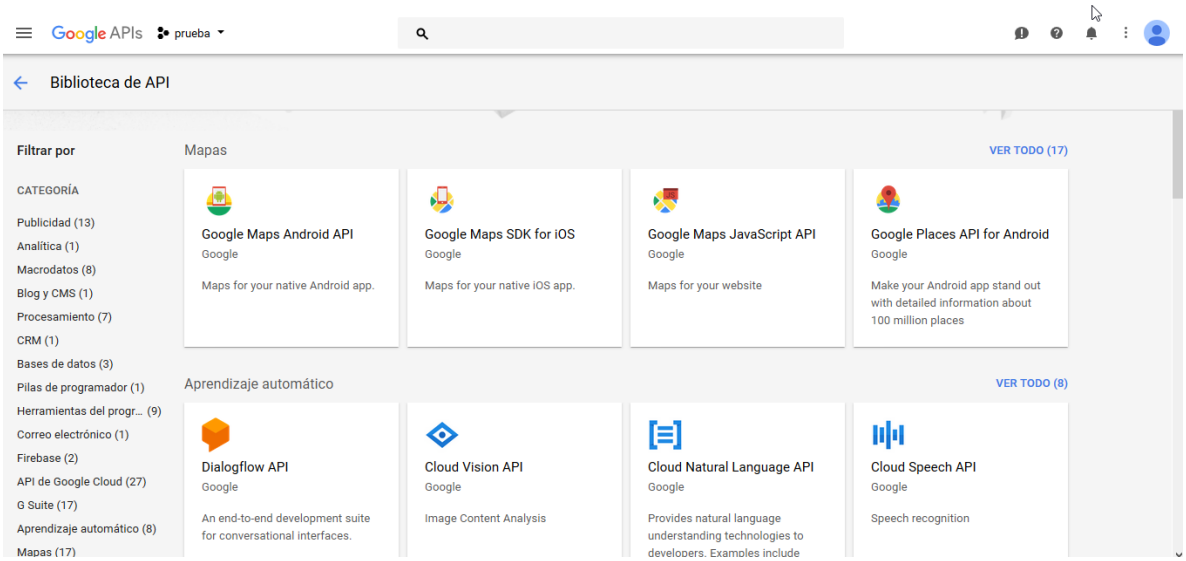


Figura 2. 6 Bibliotecas Google Maps

De la biblioteca de Google Maps se van habilitar las características necesarias, para el proyecto son las siguientes ver figura 2.7

Figura 2. 7 Habilitar Servicios

Como último paso en necesario agregar la clave del API de Google al proyecto en Android Studio, como se muestra a continuación ver figura 2.8

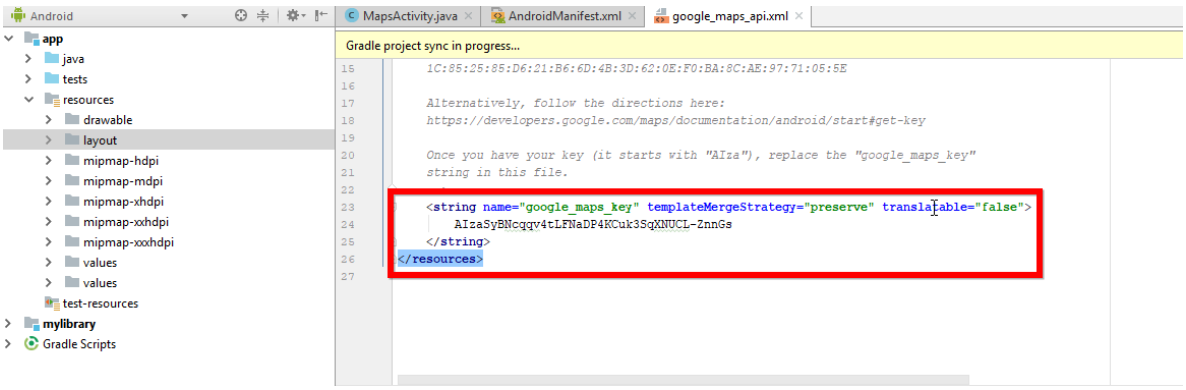


Figura 2. 8 Key de Google Maps en Proyecto Android

Con esto quedan habilitados los servicios de Google Maps para utilizar las librerías y los servicios de Google, pero con la licencia gratuita de google se tienen ciertas restricciones.

Usuarios de la API estándar:

1) 2500 solicitudes gratuitas por día, calculadas como la suma de las consultas del cliente y el servidor.
2) 50 solicitudes por segundo, calculadas como la suma de las consultas del cliente y el servidor.
3) Habilitar la facturación de pago según el uso para desbloquear cuotas mayores: USD 0.50/1,000 solicitudes adicionales, hasta 100,000 diarias.

Clientes de Google Maps APIs Premium Plan:

1) Cuota gratuita diaria compartida de 100,000 solicitudes cada 24 horas; las solicitudes adicionales se cargarán a la compra anual de créditos de Maps API.
2) 50 solicitudes por segundo del servidor.

3) Contratos anuales con términos corporativos
4) Soporte técnico las 24 horas
5) Acuerdo de nivel de servicio (SLA)
6) Licencias para casos de uso internos, de OEM y de seguimiento de recursos.

Para concluir este capítulo, se explica de qué manera será planteada la solución, así como presentamos los conceptos para el desarrollo y la estructura de una aplicación Android. También se planteó de qué manera se puede conseguir una clave para utilizar todos los servicios de Google Maps.

Capítulo 3

Implementación de AVA

En este capítulo explicaremos de los conceptos a tener en cuenta para resolver el problema planteado, un pseudocódigo y la implementación en Java, Android y Google Maps, en que consiste la aplicación y de qué manera funciona.

3.1 Métodos para resolver el problema

3.1.1 Tiempo de Ejecución

Al tratarse de un problema NP, el principal problema a resolver para esta aplicación es el tiempo, el cual crece de manera exponencial de la forma $(2^n - 1)$,con n con respecto al número de nodos(Destinos).

3.1.2 Espacio de almacenamiento

Mientras mayor sea el número de destinos mayor será el tiempo y consumo de memoria RAM del dispositivo. Ya que los celulares más comunes están muy limitados en estos recursos, es muy importante cuidar tanto el uso de recursos como el almacenamiento interno.

3.1.3 Conectividad a Google Maps

AVA utiliza Google Maps para determinar las distancias entre cada pareja de puntos, sin embargo, el API de Google Maps la conectividad limita la URL al número de caracteres por cada consulta. Esto incluye la encriptación agregada por URL, las coordenadas y las API KEY. Por lo tanto, no se pueden calcular todas las distancias en un solo query a Google Maps, es decir no se pueden realizar las consultas de todos los nodos contra todos los nodos en una sola consulta

3.1.4 Conectividad a Internet

Cuando exista el "típico" problema de conexión lenta o limitada éste hará que el mapa tarde más en mostrarse, pero ello es ajeno a nuestra aplicación. Es recomendable tener una conexión de datos mínimo de 3G.

3.1.5 Solución al tiempo de ejecución exponencial

Para la solución se implementó el algoritmo de Kruskal el cual fue adaptado para que se dé una solución semi-optima (idéntica a la óptima para la mayoría de casos, para algunos casos un poco diferente alguna variable) pero de esta manera se garantiza un término de ejecución en un tiempo aceptable.
20 nodos fue el tamaño de problema más grande que se usó en las pruebas realizadas con AVA. Este puede ser un numero aceptable de clientes a visitar en varias de las aplicaciones posibles del sistema. Para casos donde el número de nodos sea mayor no se garantiza un tiempo razonable de cómputo de la solución.

3.1.6 Realizar las consultas a Google Maps

Para dar una solución al problema de conectividad limitada por el número de caracteres, se decidió dividir la consulta haciendo de un nodo a otro y de regreso, con ello se divide el número de caracteres, pero con la consecuencia de incrementar el número de consultas realizadas.
Ya que las consultas se limitan a 2 500 consultas por día, se terminó realizando la consulta de un nodo a todos los demás nodos (destinos), con esto se encontró un equilibro entre el número de consultas diarias y el número de caracteres por consultas

3.1.7 Implementación en Java

En java se resuelve el problema del agente viajero. Para ello se realizan varias listas para guardar las aristas, y los nodos en los cuales guardamos el id de cada nodo para hacer el recorrido de la lista de destinos ver figura 3.1.

```
public static class Nodo{
    int nodo;

    public int getNodo() { return nodo; }
    public void setNodo(int nodo) {
        this.nodo = nodo;
    }
}
```

Figura 3 1 Clase donde queda registrando cada nodo

La clase arista es usada para guardar cada uno de los nodos, así como el inicio, destino y costo. Es decir, del punto inicial al punto que desea llegar y el costo del recorrido, esto para cada arista y formar la matriz de adyacencia ver figura 3.2

```
public static class Arista{
    int id;
    int inicio;
    int destino;
    double costo;

    public double getId() { return id; }
    public void setId(int id) { this.id = id; }
    public double getinicio() { return inicio; }
    public void setinicio(int inicio) { this.inicio = inicio; }
    public double getdestino() { return destino; }
    public void setdestino(int destino) { this.destino = destino; }
    public double getcosto() { return costo; }
    public void setcosto(double costo) {
        this.costo = costo;
    }
}
```

Figura 3 2 Clase Arista

Para resolver el problema el algoritmo de kruskal, se requiere una matriz de adyacencia; la cual se llena con los datos de costo del recorrido de un nodo a otro, luego se ordena por el costo de menor a mayor.

Se declara la lista de tipo nodo y se inicializa (como una lista vacía), al igual que la lista de tipo arista, así como se declara la matriz de adyacencia y se inicializa, por último, se crea el nodo padre y se inicializa en 0.

Una vez llenada la matriz de adyacencia que es la representación del Grafo, se manda a la función de Kruskal, Método de kruskal para calcular el PAV ver el siguiente algoritmo.

Kruskal (A)

A= Θ
for all vértice v ϵ G.V do
Hacer-Conjunto(v)
end for
ordena las aristas de G.E en un orden no decreciente por el peso w.
for all arista(u, v) ϵ G.E, tomado en orden no decreciente por peso do
if Encontrar-Conjunto(u) = Encontrar-Conjunto(v) then
A = A U (u, v)
Unión (u, v)
end if
end for
return A
Regresar arbol T

La matriz generada es de la forma:
Matriz [0][1][3]: Este ejemplo dice que en la arista de la matriz va del nodo 0 al nodo 1 y tiene un costo de viaje de 3

Matriz [1][0] [4.56]: Después que la distancia de regreso tiene un costo de 4.56 ya que la distancia de ida no necesariamente es igual a la distancia de regreso.

De esta manera se llena la matriz de acuerdo al número de nodos de la lista que se mencionó antes, la que contiene todos los destinos para analizarlos. Se tiene que tomar en cuenta la distancia de ida como la de regreso.

En el Algoritmo de krukal lo principal es obtener la primera arista: esta se guarda en la variable arista: int [][]arista = new int[1][3];

Una vez guardada se elimina de la matriz de adyacencia. A continuación, se guardan los valores de u para el valor inicial, y el valor de v para el valor del destino ver figura 3.3. Por último, se guarda el costo en su variable la cual es el tercer valor de la matriz.

Esos valores se guardan en un nodo de tipo arista, a continuación, se hace una validación para saber que el origen no es igual al destino y se agrega el nodo de tipo Arista a la lista de Aristas.

El algoritmo de Kruskal se puede resumir en nuestro caso como:

- Buscar el nodo Padre, es decir seleccionar el punto inicial o de origen.
- Agregar a la lista TSP, es la lista donde se van agregando los nodos que se van obteniendo con Kruskal.
- Borrar el Padre.
- Buscar en los hijos el siguiente Padre y se repite el proceso.

A continuación, se comprueba que le valor de destino sea menor, después se intercambian los valores del ordenamiento, es decir se pone por encima esa arista ya que su valor de recorrido es menor. Y por último me regresa la lista de nodos, pero ordenados con el Algoritmo de Kruskal.

```
public static List<Arista> Kruskal(double [][]matriz,int nodos,int aristas){
    Arista aux = new Arista();
    double costo = 0;
    int u,v,i = 0,p,aux1;
    int j,k;
    double [][]arista = new double[1][3];
    int [][]conexa = new int[nodos][2];
    List<Arista> lista = new ArrayList<Arista>();

    //Inicializa la matriz
    for(k=0;k<nodos;k++){
        for(j=0;j<2;j++){
            conexa[k][j] = k;
        }
    }
    while(i != aristas){
        arista = obtenerElemento(matriz);
        matriz = borrar(matriz);
        //obtienen los valores
        u = (int) arista[0][0];
        v = (int) arista[0][1];
        costo = arista[0][2];
        //Guardan los valores en el Arista
        aux.setId(i);
        aux.setinicio(u);
        aux.setdestino(v);
        aux.setcosto(costo);

        if(conexa[v][1] != conexa[u][1]){
            lista.add(aux);             //Agrega la Arista
            aux = new Arista();         //Crea nuevo Arista
            aux1 = conexa[v][1];        //Obtengo el valor del origen
            conexa[v][1] = conexa[u][1];//
            for(p=0;p<nodos;p++){
                if(conexa[p][1] == aux1){
                    conexa[p][1] = conexa[u][1];
                }
            }
        }
        i++;
    }
    return lista;
}
```

Figura 3 3 Implementación de Kruskal

Una vez obtenida la lista de Kruskal se manda llamar la función preorder ver figura 3.4, la cual crea una lista con el orden en que se van a recorrer los nodos, esta función inserta en la lista el nodo y este se considera el padre, luego se buscan sus hijos y se guardan en una lista. A continuación, se recorre la lista de todos los hijos y llama el siguiente hijo que a su vez será el padre en la segunda iteración.
A continuación, se buscan sus hijos, y así hasta recorrer y agregar todos los nodos.

```
public static ArrayList preoder(List<Arista> lk,double nodo, ArrayList tsp){
    ArrayList hijos = new ArrayList();
    Inserta(tsp,nodo);
    hijos = Childrens(lk,nodo,hijos);
    for(Object hijo : hijos){
        preoder(lk, (double) hijo,tsp);
    }
    return tsp;
}
```

Figura 3 4 Pre Order

3.1.8 Implementación en Android

La implementación en Android consiste en gran parte en el diseño de la aplicación, básicamente es la parte visual: los botones, los layouts y funcionamiento en general de la aplicación.

Por ejemplo, la creación de un hilo padre y sus hijos, el tiempo de vida de los hilos (hijos), así como su eliminación.

También se manejan los "intent" que son los cambios de un layout a otro. Es decir, al presionar el botón de inicio del layout principal se creará un intent (hilo) que dirija al layout que muestra el mapa. Al regresar al layout principal se termina ese hilo ver figura 3.5.

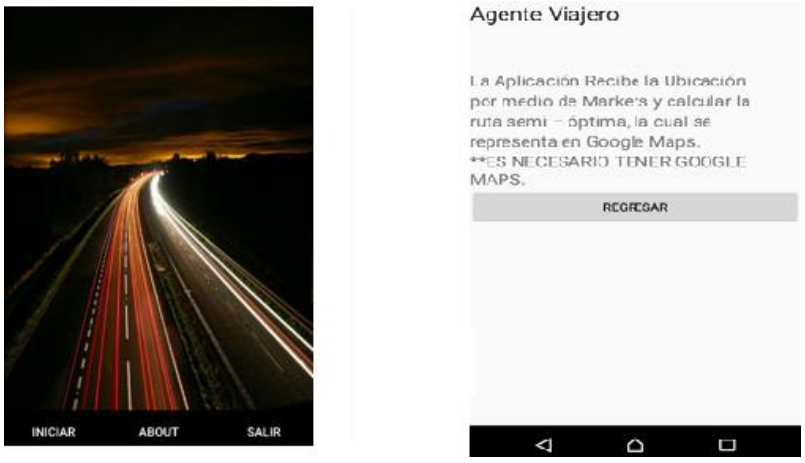


Figura 3 5 Ejemplo de vistas en los Layout

La creación de un botón por ejemplo el de inicio o about se lleva a cabo mediante las siguientes líneas de código en las cuales se define el id y los tamaños para los botones, así como el color y el nombre del botón ver la figura 3.6

```
<Button
    android:id="@+id/biniciar"
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@color/common_google_signin_btn_text_dark_focused"
    android:text="Iniciar" />

<Button
    android:id="@+id/bacercade"
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@color/common_google_signin_btn_text_dark_focused"
    android:text="About" />
```

Figura 3 6 Botones en Android

3.1.9 Uso del API de Google Maps

Google Maps utiliza un Fragment (representa una interfaz particular que se ejecuta dentro de la aplicación) el cual utilizamos para representar el mapa, Google deja utilizar todas sus bibliotecas para agregar Maker (Marcador de ubicación en el mapa). Un fragment es un componente que se usa en un layout para mostrar los mapas producidos por Google Maps.

Google Maps te permite también desplegar la ubicación en un plano cartesiano representado por las coordenadas de Latitud y Longitud; permite también trazar la ruta ver figura 3.7.

Figura 3 7 Mapa API Google Maps

3.2 Descripción de la aplicación

3.2.1 Vistas de aplicación

Las aplicaciones —también llamadas apps— están presentes en los teléfonos desde hace tiempo. De hecho, ya estaban incluidas en los sistemas operativos de Nokia o Blackberry años atrás.

Los móviles de esa época, contaban con pantallas reducidas y muchas veces no táctiles, y son los ahora llamados feature phones, en contraposición a los smartphones, más actuales.

Actualmente existen aplicaciones de todo tipo, forma y color, pero en los primeros teléfonos, estaban enfocadas en mejorar la productividad personal: se trataba de alarmas, calendarios, calculadoras y clientes de correo.

Hubo un cambio grande con el ingreso de iPhone al mercado, ya que con él se generaron nuevos modelos de negocio que hicieron de las aplicaciones algo rentable, tanto para desarrolladores como para los mercados de aplicaciones, como App Store, Google Play y Windows Phone Store.

Lo principal en la aplicación en cuanto a la implementación fue el diseño de las vistas, la vista principal es la siguiente.

Esta vista nos representa el índice de la aplicación, la cual cuenta con 3 botones los cuales son:

- Iniciar
- About
- Salir

Al mismo tiempo, también mejoraron las herramientas de las que disponían diseñadores y programadores para desarrollar apps, facilitando la tarea de producir una aplicación y lanzarla al mercado, incluso por cuenta propia.

En esencia, una aplicación no deja de ser una pieza de software. Para entender un poco mejor el concepto, diremos que las aplicaciones son para los móviles lo que los programas son para los ordenadores de escritorio.

Una parte esencial de toda aplicación es parte visual la cual, ya que por medio de ella se decide si la aplicación es buena o mala. A primera vista podría decirse que es la capa que hay entre el usuario y el corazón funcional de la app, el lugar donde se realizan las interacciones.

En mayor medida está compuesta por botones, gráficos, íconos y fondos, que tienen una apariencia visual diferente en cada uno de los sistemas operativos, porque Android, iOS y Windows Phone tienen su propia forma de diseño.
A continuación, la vista principal de la aplicación ver figura 3.8.

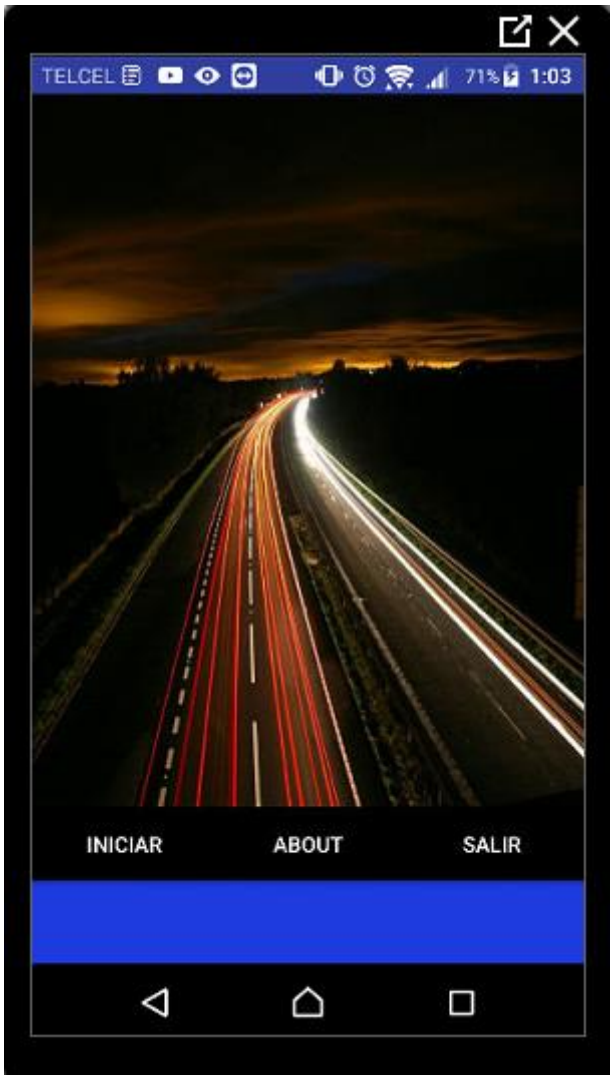


Figura 3 8 Vista Principal

3.2.2 Ejecución

La función del primer botón llamado Iniciar se encarga de mostrar el mapa que se va representar, así como las siguientes opciones:

- Mi ubicación.
- Agregar Destino.
- Quitar Destino.
- Calcular.
- Tipo de mapa: Satelital, Terreno e Híbrido.
- Regresar.

El Botón de mi ubicación: por defecto, al iniciar la aplicación manda a la ubicación actual, si se mueve el mapa para seleccionar los destinos; de ser necesario se puede seleccionar para acomodar el mapa de nuevo en la ubicación actual.

Agregar Destino: para agregar el destino se tiene que seleccionar esta opción y una vez hecho eso, dejar sostenido sobre el mapa 2 a 3 segundos para agregar el marcador.

Una vez agregado se desactiva la opción para agregar un nuevo marcador, por lo tanto; si se desea agregar un nuevo marcador se tiene que seleccionar nuevamente la opción de agregar un nuevo marcador.
Una vez agregado el marcador se puede mover de lugar si es necesario ver figura 3.9.

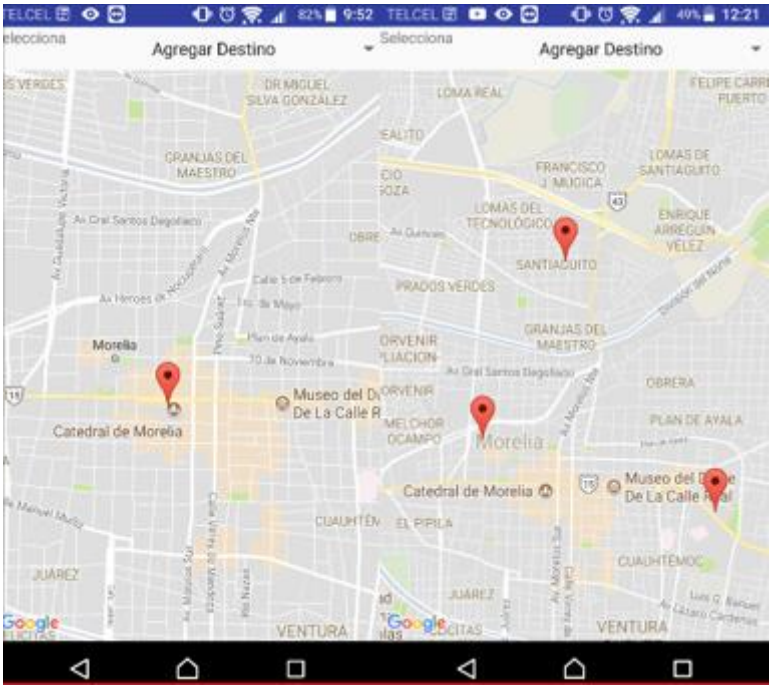


Figura 3 9 Agregar 1 o más Destinos

Quitar Destino: para quitar un marcador es necesario seleccionar la opción del menú desplegable, luego de eso seleccionar el marcador que requiere quitar.

Botón Calcular: Al momento de presionar el botón Calcular, se llama la función main, la cual se encarga de realizar los pasos siguientes ver figura 3.10.

```
public static void Inicializar(List<Destino> lista, double ma[][], int aristas ){
    Ordenar(ma[][] , aristas);
    preorder(lista);
}
```

Figura 3 10 Proceso para obtener el recorrido más corto

La función inicializar se encarga de recibir la lista de Destinos y la matriz de adyacencia. Después se encarga de hacer las consultas a Google Maps para obtener la distancia de un marcador a otro, con sus coordenadas y de esta manera se va llenado la matriz de adyacencia con la distancia de un destino a los demás.

En la figura 3.11 se representa el código como se mencionó anteriormente la diagonal de la matriz se considera infinito por lo que no se toma en cuenta, partiendo de esa premisa se tiene la condición "si inicio es diferente a destino" es decir si el nodo no va del punto 'A 'al punto 'A'. Una vez obtenida la consulta de Google Maps se verifica que la consulta sea válida, es decir que el lugar sea accesible, Después se obtiene los datos del JSON que regresa Google Maps y se agregan a la matriz hasta obtener la distancia de todos los nodos.

```
public double[][] Inicializar(List<Destino> nuevalista,double[][] matriz)
        throws FileNotFoundException, IOException, JSONException,
ExecutionException, InterruptedException {
    List<LatLng> Cordenadas = new ArrayList<LatLng>();
    int in,des;
    String status=null;
    ObtenerJSON consulta = new ObtenerJSON();
    float dis = 0;
    JSONObject str_result ;
    double[][] aux = new double[1][1];
    aux[0][0] = 9876.54321;
    for(Destino inicio: nuevalista){
            for(Destino fin: nuevalista){
            //le voy a mandar los dos nodos
                in = (int) inicio.getId();
                des = (int) fin.getId();

                if(in != des){
                    Cordenadas = ObtenCordenadas(in,des,nuevalista);
                    //Crear la cadena para agregar el valor
                    cadena = Crea_Cadena(Cordenadas,in);
                    //Mandar la cadena para calcular el valor de la distancia
                    str_result =  consulta.execute(" "," ").get();
                    //obtener los datos de JSON
                    status = validar(str_result);
                    if(status.equals("NOT_FOUND")) {
                        Toast.makeText(getApplicationContext(), "Alguna Ubicacion
no es Accesible ", Toast.LENGTH_LONG).show();
                        return aux;
                    }
                    else{
                        dis = Obtener_datos(str_result);
                        //mandar las cordenadas para realizar una consulta y
obtener la distancia
                        AgregarMatriz(matriz,Cordenadas,in,des,dis);
                        str_result = new JSONObject();
                        consulta = new ObtenerJSON();
                        //mandar la matriz y las cordenas para agregarlas
                        Cordenadas = new ArrayList<LatLng>();
                    }
                }
        }
    }
    return matriz;
}
```

Figura 3 11 Recorrido de los Destinos.

La función para obtener las coordenadas recibe el origen, el destino y la lista de nodos ver figura 3.12

```
public static List<LatLng> ObtenCordenadas(int inicio,int destino,List<Destino> lista){

    LatLng Cinicio,Cdestino;
    List<LatLng> cordenadas = new ArrayList<LatLng>() ;
    int idi,idd;

    for(Destino i: lista) {
       idi = (int) i.getId();
        if (idi == inicio) {
            Cinicio = i.getCordenada();
            cordenadas.add(Cinicio);
            for (Destino d : lista) {
                idd = (int) d.getId();
                if (idd == destino) {
                    Cdestino = d.getCordenada();
                    cordenadas.add(Cdestino);
                }
            }
        }
    }
    return cordenadas;
}
```

Figura 3 12 Función Para Comprobar las Coordenadas

La función consiste en recorrer la lista de nodos, al encontrar el origen y destino se obtienen las coordenadas y se regresan. Una vez obtenidas las coordenadas se tiene que generar la cadena para realizar la consulta a Google Maps.

Para ello se necesitan las coordenadas obtenidas anteriormente. Se debe ajustar al formato que utiliza Google Maps para las consultas de su servidor, y ajustar la cadena que vamos a mandar para la consulta ver figura 3.13.

```
public String Crea_Cadena (List<LatLng> lista, int ID){
    String cadena = "https://maps.googleapis.com/maps/api/distancematrix/json?origins=";
    String origen = null;
    String destino = null;
            origen = convierte(lista.get(0));
            cadena = cadena + origen;

    cadena = cadena+ "&destinations=";

            destino = convierte(lista.get(1));
            cadena = cadena + destino;

    cadena = cadena + "&key=AIzaSyAq0tTqKVDID6ck01m-Xez-YPCzipI2xXc";
    return cadena;
}
```

Figura 3 13 Función Para Crear la Cadena

Una vez realizada la consulta se analiza el JSON que nos regresa Google Maps el cual tiene el siguiente formato ver figura 3.14.


```
{
   "destination_addresses" : [ "New York, NY, USA" ],
   "origin_addresses" : [ "Washington, DC, USA" ],
   "rows" : [
      {
         "elements" : [
            {
               "distance" : {
                  "text" : "225 mi",
                  "value" : 361715
               },
               "duration" : {
                  "text" : "3 hours 49 mins",
                  "value" : 13725
               },
               "status" : "OK"
            }
         ]
      }
   ],
   "status" : "OK"
}
```


Figura 3 14 Resultado en formato JSON

Una vez recibidos los datos en formato JSON, lo primero que se requiere es validar la información con el "Status" si es OK se obtienen los datos. De lo contrario, si llega *"NOT_FOUND",* se aborta la señal, porque una de las coordenadas no es válida. Es decir, Google Maps
no tiene registrado ese punto, o no existe una ruta para llegar al mismo.

Si por el contrario todo está en orden se obtienen los datos que interesan, que son el tiempo o la distancia ver figura 3.15.

```
JSONArray rows = objeto.getJSONArray( name: "rows");
String ElementosJSON = "SIN DATOS PARA ESA ALTITUD Y LONGITUD";

String distancia = null;
String tiempo = null;

JSONArray jsonArray = null;

try {
    for (int i = 0; i < rows.length(); i++) {
        ElementosJSON = rows.getJSONObject(i).getString( name: "elements");
        jsonArray = new JSONArray(ElementosJSON);
        JSONObject objetoJson = jsonArray.getJSONObject(i);

        //Obtenemos los objetos.
        JSONObject jsonObjectDate = objetoJson.getJSONObject("distance");
        JSONObject jsonObjectTime = objetoJson.getJSONObject("duration");

        //A partir del objeto obtenemos los valores de date y time.
        distancia = jsonObjectDate.getString( name: "value");
        tiempo = jsonObjectTime.getString( name: "value");

    }
} catch (JSONException e) {
    e.printStackTrace();
}
```

Figura 3 15 Obtención de la respuesta del JSON

Los datos del resultado se guardan en la matriz de adyacencia como la distancia de un nodo a otro. Esto se repite hasta llenar la matriz con la distancia de todos los nodos hacia todos los nodos.
Una vez completada la matriz de adyacencia se debe ordenar por su valor de distancia de menor a mayor. Para ello basta con hacer una adaptación del método de la burbuja.

```
    for(k=0;k < aristas;k++){
        for(i=0 ; i<aristas-1 ; i++){
            if(matriz[i][2] > matriz[j][2]){
                x = matriz[i][0];
                y = matriz[i][1];
                aux = matriz[i][2];
                mat[0][0] = matriz[j][0];
                mat[0][1] = matriz[j][1];
                mat[0][2] = matriz[j][2];
                matriz[i][0] = mat[0][0];
                matriz[i][1] = mat[0][1];
                matriz[i][2] = mat[0][2];
                matriz[j][0] = x;
                matriz[j][1] = y;
                matriz[j][2] = aux;

            }
            if(j < aristas-1)
                j++;
        }
        j=1;i=0;
    }
    return matriz;
}
```

Figura 3 16 Adaptación del método burbuja

El siguiente fragmento de código se utiliza para ordenar la matriz de menor a mayor distancia de recorrido ver figura 3.16.

Una vez ordenadas las distancias se manda llamar la función que resuelve el algoritmo de Kruskal como se explicó anteriormente.

Este algoritmo regresa el árbol de expansión mínimo, el cual se recorre en preorden para obtener el orden en que se serán visitados los nodos ver figura 3.18.

```
public static List<Arista> Preorder(List<Arista> lista){
    double del = 0;

    ArrayList tsp = new ArrayList();
    List<Arista> TSP = new ArrayList<~>();
    double father = 0;
    Nodo nodo = new Nodo();

    father = BuscaRaiz(lista);
    nodo.nodo = father;
    TSP =  preoder(lista,nodo.nodo,tsp);
    return TSP;
}
```

Figura 3 17 Recorrido de la lista de coordenada

Una vez obtenida la lista se genera una cadena con todas las coordenadas que se van a representar en el mapa de Google. Donde L es la liga con los campos que requiere la consulta de google y aristas son las coordenadas que se calcularon anteriormente. Es decir, el orden del recorrido.

link = CreaLink(L,arista);

La función anterior se encarga de generar la liga en el orden en que se acomodó con la función pre-order y la agregar de una por una las coordenadas para que se representen de esa manera en Google Maps.

String cordenada = Convierte(Cordenada);
liga = liga+cordenada+"/";

Por último, se manda esa liga a Google Maps para que represente la ruta de la siguiente manera (por defecto el medio de transporte es automóvil) ver figura 3.19.

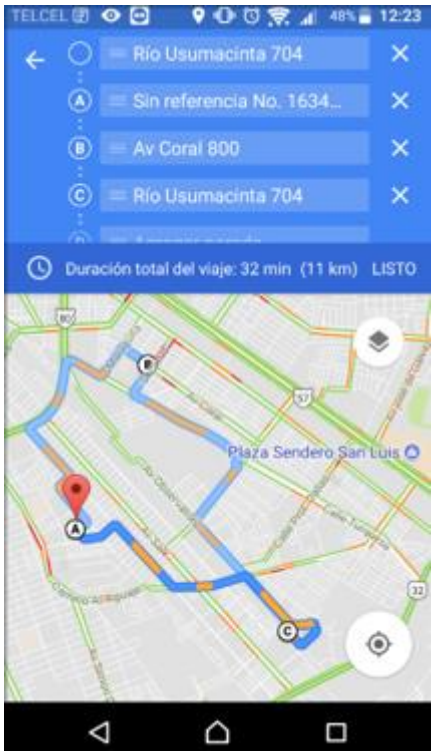


Figura 3 18 Resultado de la Aplicación

Google Maps permite usar 3 tipos de mapas para el gusto del usuario, los cuales son mapa satelital, mapa terreno y mapa híbrido, las Figuras 3.20, 3.21 y 3.22 muestran ejemplos de cada tipo de mapa.

Figura 3 17 Mapa Terreno

Figura 3 18 Mapa Híbrido

Figura 3 19 Mapa Satelital

Botón de Regresar, Consiste en cerrar el hilo del layout y abrir nuevamente el layout de index.

Botón About, crea un layout que explica en que consiste la aplicación y da la opción de regresar ver figura 3.23.

Figura 3 20 Vista información

3.3 Resultado

En esta sección se incluye un ejemplo del funcionamiento del sistema AVA, así como los resultados que produce. El resultado se ilustrará con el siguiente conjunto de puntos a utilizar.

- Catedral Morelia (@19.702423,-101.1945072)
- Mercado de Dulces y Artesanías, Centro Histórico, Morelia, Michoacán (19.7037391,-101.1979896)
- Hospital General "Dr. Miguel Silva" (@19.6996107,-101.1857702)
- Fuente Las Tarascas (@19.7028642,-101.1845269)
- Hospital Ángeles Morelia (@19.6675153,-101.1679358)
- Hospital Star Médica Morelia (@19.6826667, -101.1939679)
- Parque Zoológico Benito Juárez (@19.6827518,-101.193904)
- Plaza de Toros Monumental de Morelia (@19.7004001,-101.2132134)
- Estadio Morelos (@19.7187738,-101.2355831)
- Teatro José María Morelos (@19.6824276,-101.1852499)

Tras obtener todas las consultas de google maps, realizamos la matriz de adyacencia la cual nos arrojó los siguientes resultados.

	Catedral Morelia	M. dulces	General "Dr. Miguel Silva"	Fuente Las Tarascas	Hospital Ángeles Morelia	Star Médica Morelia	Zoológico Benito Juárez	Toros Monumental de	Estadio Morelos	Teatro José María Morelos
Catedral Morelia	0	898	1312	1280	6444	2810	2802	2489	7466	3283
M. dulces	701	0	1797	1760	6804	3139	3162	2317	7294	3642
Hospital General "Dr. Miguel Silva"	1496	2126	0	632	6335	2700	2692	3634	107727	3096
Fuente Las Tarascas	1426	1798	587	0	6885	4186	4179	3578	10278	3359
Hospital Ángeles Morelia	8004	8435	7689	11204	0	5306	5378	9705	20302	6090
Hospital Star Médica Morelia	2546	2977	3158	3761	4558	0	1008	4247	9030	1880
Parque Zoológico Benito	2553	2984	3168	3768	4566	8	0	4254	9037	1887
Plaza de Toros Monumental	2371	2003	5821	5649	8412	4202	4194	0	5628	4674
Estadio Morelos	7578	7209	12590	12978	16354	9409	9401	5980	0	9881
Teatro José María	3664	4095	2700	2805	4130	1119	1111	5365	10148	0

Figura 3 21 Matriz de Adyacencia

Origen	Destino	Costo (Metros)
0	1	898
0	2	1,312
0	3	1,280
0	4	6,444
0	5	2,810
0	6	2,802
0	7	2,489
0	8	7,466
0	9	3,283
1	0	701
1	2	1,797
1	3	1,760
1	4	6,804
1	5	3,169
1	6	3,162
1	7	2,317
1	8	7,294

1	9	3,642
2	0	1,496
2	1	2,126
2	3	632
2	4	6,334
2	5	2,700
2	6	2,692
2	7	3,624
2	8	10,727
2	9	3,096
3	0	5,123
3	1	1,426
3	2	1,798
3	4	587
3	5	6,885
3	6	4,186
3	7	4,179
3	8	3,578
3	9	10,278
4	0	8,004
4	1	8,435
4	2	7,689
4	3	11,204
4	5	5,386
4	6	5,378
4	7	9,705
4	8	20,302
4	9	6,090
5	0	2,546
5	1	2,977
5	2	3,158
5	3	3,761
5	4	4,558
5	6	1,008
5	7	4,247
5	8	9,030
5	9	1,880
5	0	2,553
6	1	2,984
6	2	3,165
6	3	3,768
6	4	4,556
6	5	8
6	7	42
6	8	90

6	9	1,887
7	0	2,371
7	1	2,003
7	2	5,821
7	3	5,649
7	4	8,412
7	5	4,202
7	6	4,194
7	8	5,628
7	9	4,674
8	0	7,578
8	1	7,278
8	2	12,590
8	3	12,978
8	4	15,354
8	5	9,409
8	6	9,401
8	7	5,989
8	9	9,881
9	0	3,664
9	1	4,095
9	2	2,700
9	3	2,805
9	4	4,130
9	5	1,119
9	6	1,111
9	7	5,365
9	8	10,148

Figura 3 22 Matriz de Distancias

Después se obtiene la siguiente figura, la cual representa el resultado obtenido tras usar el algoritmo de Kruskal.

Origen	Destino	Costo(metros)
6	5	8
3	2	587
0	8	898
0	1	701
9	6	1,111
0	3	1,280
5	0	2,546
1	7	2,317
9	4	4,130

Figura 3 23 Resultado del Algoritmo de Kruskal

A partir de

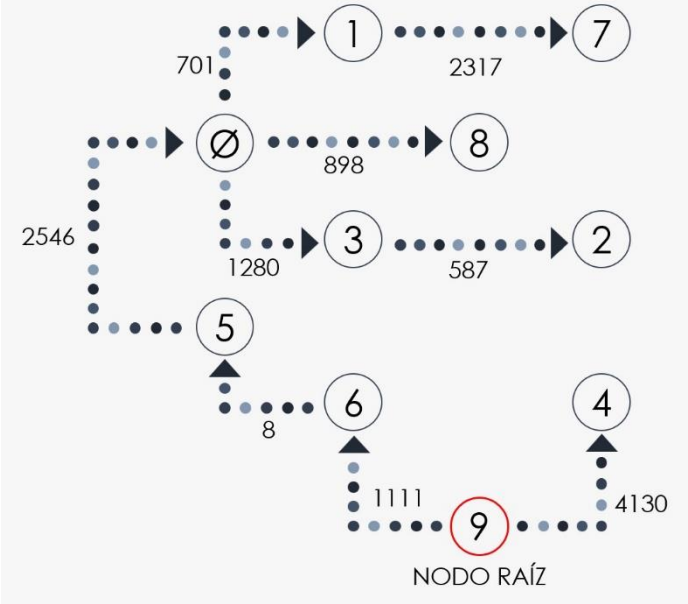
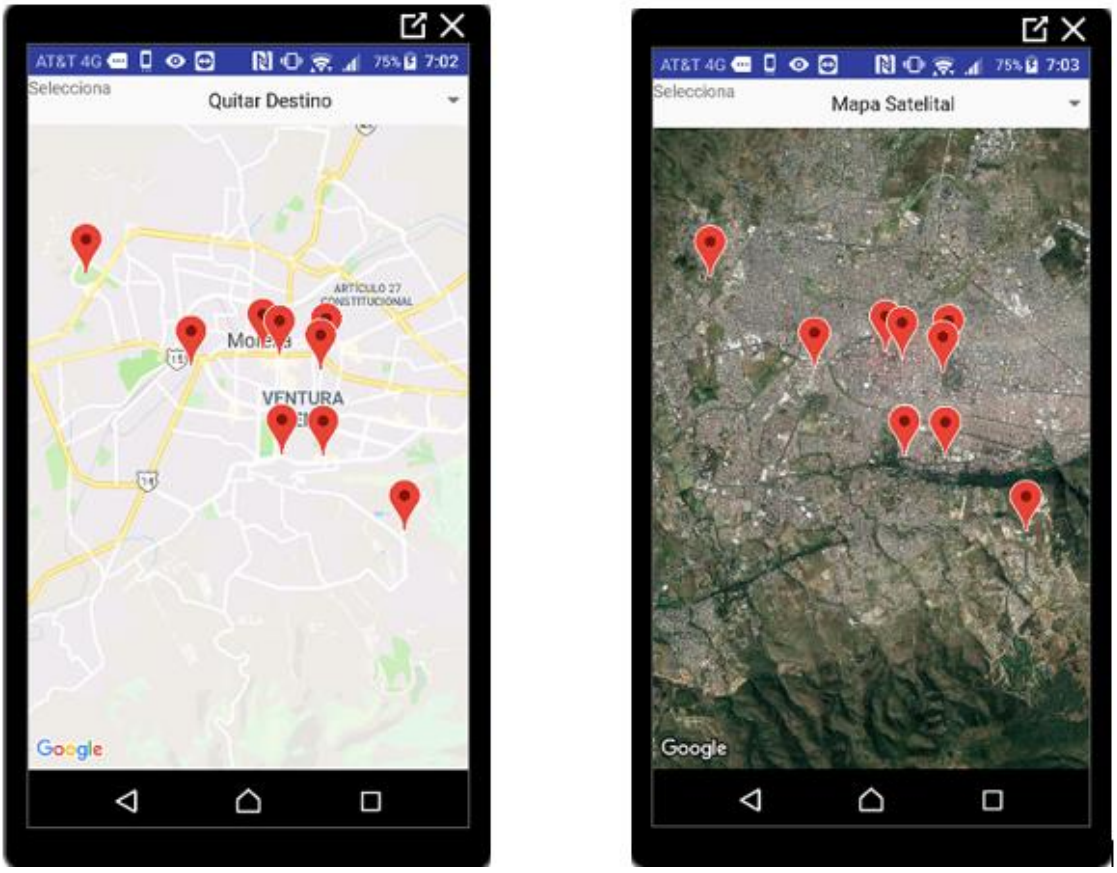


Figura 3 24 Árbol Expiación Mínima

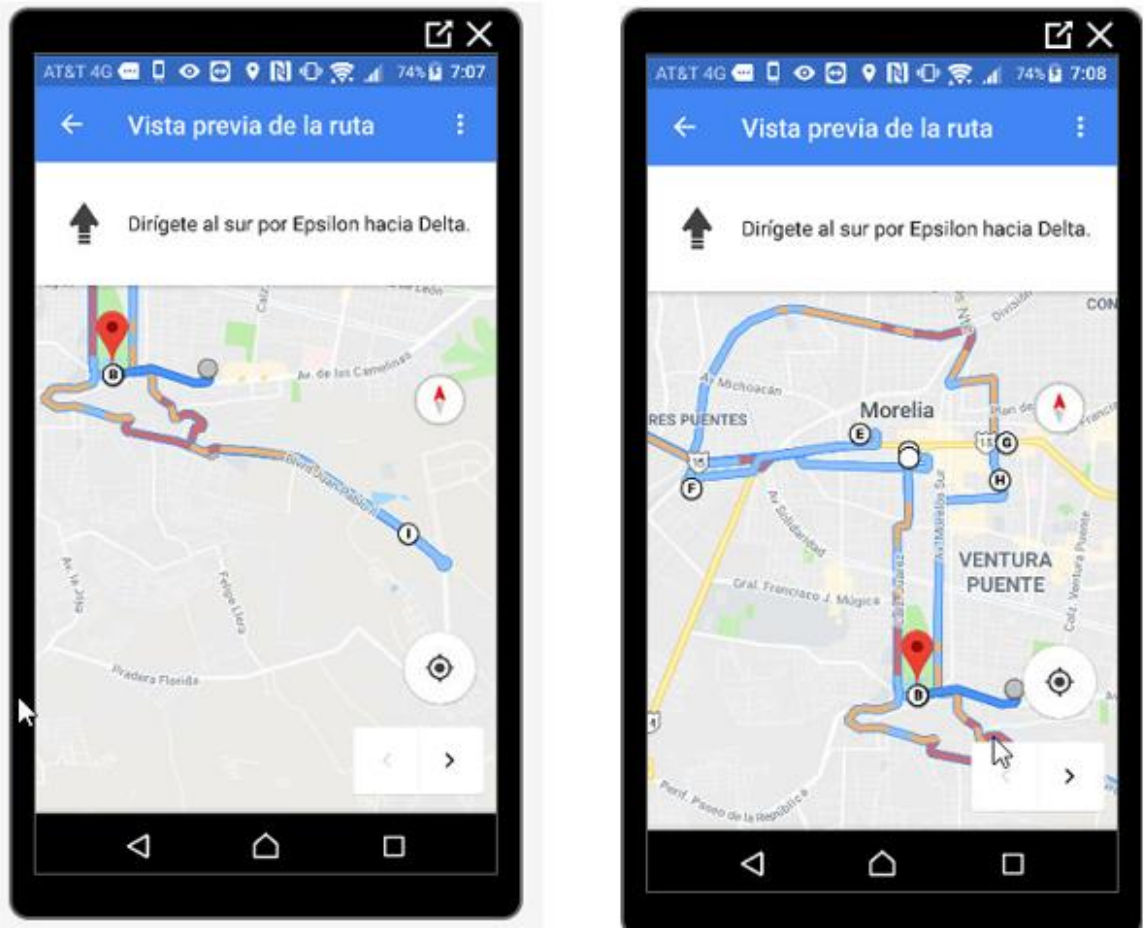
A continuación, se presenta el resultado del recorrido en pre-orden se obtiene la siguiente lista donde el origen o nodo padre es el nodo: 9

- PADRE (raíz) = 9
- Lista: [9, 6, 5, 0, 8]
- HIJOS: []
- Lista: [9, 6, 5, 0, 8, 1, 7]
- HIJOS: []
- Lista; [9, 6, 5, 0, 8, 1, 7]

- HIJOS: [7]
- Lista; [9, 6, 5, 0, 8, 1, 7, 3, 2]
- HIJOS: []
- Lista: [9, 6, 5, 0, 8, 1, 7, 3, 2]
- HIJOS: [2]
- Lista: [9, 6, 5, 0, 8, 1, 7, 3, 2]
- HIJOS: [8, 1, 3]
- Lista: [9, 6, 5, 0, 8, 1, 7, 3, 2]
- HIJOS: [0]
- Lista: [9, 6, 5, 0, 8, 1, 7, 3, 0]
- HIJOS: [5]
- Lista: [9, 6, 5, 0, 8, 1, 7, 3, 2, 4]
- HIJOS: []
- Lista: [9, 6, 5, 0, 8, 1, 7, 3, 2, 4]
- HIJOS: [6, 4]
- Resultado: [9, 6, 5, 0, 8, 1, 7, 3, 2, 4] Tiempo ejecución: 0.017s

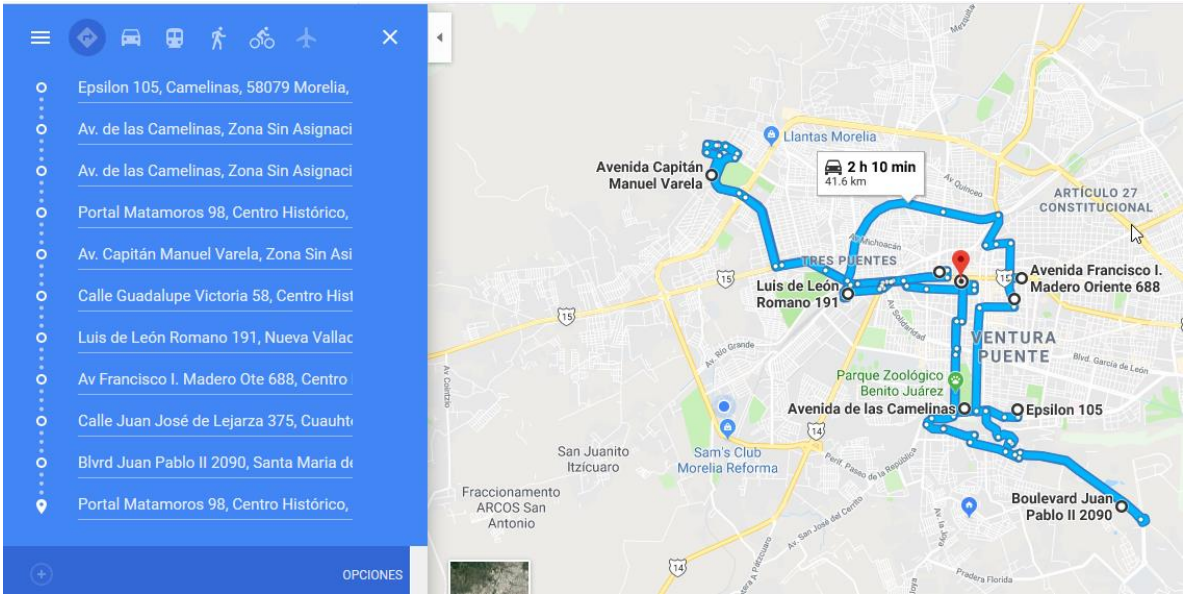


Resultado de AVA 1. 1

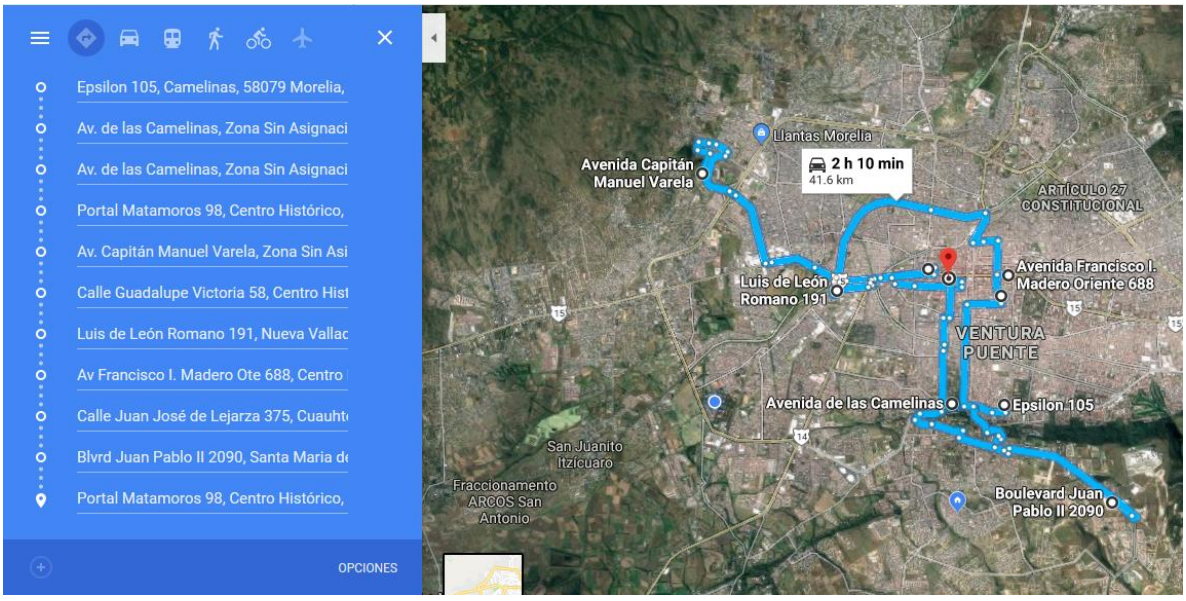


Resultado de AVA 1. 2

Resultado de AVA 1. 3



Resultado en Google Maps 1. 1



Resultado en Google Maps 1. 2

Capítulo 4

Conclusiones

Esta tesis presenta la solución del PAV. La implementación de la solución trabaja con el algoritmo de Kruskal el cual nos dio mejor resultado que el algoritmo de Prim y Dijkstra, ya que al comienzo del desarrollo se implementaron y el algoritmo de kruskal dio los mejores resultados en cuanto a tiempo y computó. El tiempo de cálculo y de uso de recursos es bastante aceptable ya que los Smart Phone de gama media y baja no cuentan con bastantes recursos.

Una ventaja es que la ruta es trazada en Google Maps, por lo tanto, es trazada y actualizada constantemente. Las principales ventajas son que los mapas respetan los sentidos, horarios y niveles de tráfico. Otra ventaja es que se cuenta con todas las opciones de navegación y asistencia. Así como la comodidad sobre todas las opciones que recorren varios lugares y regresar al origen.

El recorrido en pre-orden en resumen es:

- Busca el primer nodo y se usa como raíz.
- Se agrega el hijo izquierdo debajo el padre.
- A continuación, sus demás hijos se ponen como hermanos.
- De esta manera se construye el árbol N-ario.

A continuación, breve resumen del funcionamiento de la App.

Primero es inicializar la matriz en orden de menor a mayor costo.
Manda llamar el algoritmo de Kruskal.

1)Buscar el nodo padre.
2)Agregar a la lista TSP.
3)Borrar el padre agregado.
4)Buscar el siguiente padre.

Hacer el recorrido en pre-orden.

1)Busca el primer nodo de la lista de TSP y se usa como raíz.
2)Se Agrega la raíz y se busca el hijo de la izquierda el cual se usa como raíz.
3)De esta manera se construye el recorrido que aparecerá en Google Maps.

Para concluir, el problema del agente viajero en se requieren varios nodos (destinos) los cuales serán determinados por los usuarios desde la aplicación.

Una vez obtenidos los nodos se creará un grafo que posteriormente se analizará con el algoritmo de Kuskal; una vez resuelto se obtendrá como resultado un árbol de expansión mínima.

La trayectoria final será representada en Google Maps con la ruta a seguir para visitar todos los destinos con la menor distancia y regresar al origen.

Para lograr la trayectoria se recorre el árbol n-ario recorriendo primero su hijo izquierdo y luego su hermano a la derecha.

El resultado de la aplicación es bastante bueno ya que termina en un tiempo aceptable y con un muy buen resultado. Ya que la ruta que genera es la mejor opción para visitar todos los lugares en el menor tiempo.

Bibliografía

[1] ***Johnson, Garey.*** *Computers and Intractability: A Guide to the Theory of NP-Completeness.*
[2] **home, Wikibooks. 2018.** ingenieriaindustrialonline.com/problema-del-agente-viajero-tss. *SCIELO.a08v21n1.pdf.* [En línea] 2018.
[3] https://www.java.com/es/download/faq/whatis_java.xm
[4] https://developer.android.com/about/
[5] EL PROBLEMA DEL AGENTE VIAJERO UN ALGORITMO DETERMINÍSTICO Revista Matemática: Teoría Y Aplicaciones 2014 Autores: Erasmo López, Oscar Salas y Alex Murillo
[6] EL PROBLEMA DEL AGENTE VIAJERO UN ALGORITMO DETERMINÍSTICO Revista Matemática: Teoría Y Aplicaciones 2014 Autores: Erasmo López, Oscar Salas y Alex Murillo
[7] EL Problema del Vendedor Viajero (TSP) y Programación Entera (IP), Daniel Espinoza, Universidad de Chile Facultad de Ciencias Físicas y Matemáticas Departamento de Ingeniería Industrial
[8] http://appdesignbook.com/es/contenidos/las-aplicaciones/
[9] https://console.developers.google.com/apis/credentials/key
[10] https://developers.google.com/maps/?hl=es-419
[11] https://developers.google.com/maps/documentation/javascript/examples/?hl=es-419
[12] https://developers.google.com/maps/documentation/javascript/support?hl=es-419
[13] https://developers.google.com/maps/documentation/javascript/adding-a-google-map?hl=es-419\\