UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Análisis e implementación de un algoritmo, incrustado en un administrador de base de datos relacional, para encontrar la ruta más corta entre dos puntos geográficos

Norman Leonel Penagos Estrada

Guatemala

2010

Análisis e implementación de un algoritmo, incrustado en un administrador de base de datos relacional, para encontrar la ruta más corta entre dos puntos geográficos

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Análisis e implementación de un algoritmo, incrustado en un administrador de base de datos relacional, para encontrar la ruta más corta entre dos puntos geográficos

Trabajo de investigación presentado por Norman Leonel Penagos Estrada para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación

Guatemala

2010

Vo.Bo.:

(f)_____

M.Sc. Alejandra Reynoso

Tribunal:

(f)_____

M.Sc. Alejandra Reynoso

(f)_____

Ing. Irene Prieto

(f)_____

Ing. Carlos Marroquín

Fecha de aprobación: 10 de Septiembre de 2010

PREFACIO

A menudo se tiene la necesidad de llegar a un lugar de forma fácil y rápida, lo cual se puede ver afectado por no conocer la mejor ruta a seguir. La falta de herramientas de uso público en Guatemala que permitan generar estas rutas impulsó la realización de este trabajo.

La motivación principal fue programar un algoritmo eficiente, para encontrar el camino más corto entre dos direcciones de Guatemala, con el fin de que las personas lo utilicen de forma gratuita y vía Internet.

Para la selección del algoritmo a implementarse, se evaluaron tres algoritmos especializados en generar las rutas más cortas: el algoritmo de Dijkstra, el algoritmo de Bellman-Ford y el algoritmo de Floyd-Warshall.

Siendo el objetivo principal la minimización del tiempo de respuesta del algoritmo seleccionado, la mejora que se pretende dar a dicho algoritmo es su implementación en la base de datos relacional que contenga la información de las direcciones sobre las cuales se realizarán las búsquedas. De esta manera el algoritmo obtiene directamente la información, evitando consultas externas a la base de datos, presentando la ruta en el menor tiempo.

Quiero agradecer a Dios por permitirme llegar hasta este momento tan importante en mi vida y poner en mi camino a las personas que han sido mi soporte y compañía durante toda la carrera.

Gracias a mis padres Leticia Estrada y Norman Penagos Sandoval, y a mi hermana Jeissel Penagos por su cariño, comprensión y apoyo en todo momento.

Gracias a mi asesora M.Sc. Alejandra Reynoso por toda la ayuda que me brindó, por los consejos y comentarios que me ayudaron a realizar este trabajo.

Gracias a todos mis compañeros y amigos por todos los momentos que vivimos en la universidad y fuera de ella, ya que su apoyo y amistad siempre fue incondicional.

CONTENIDO

Página

LISTA DE ILUSTRACIONES

RESUMEN

El trabajo consiste en el estudio comparativo de tres algoritmos que permiten obtener la ruta más corta (Dijkstra, Bellman-Ford y Floyd-Warshall) contra el algoritmo implementado en el megaproyecto de Averiguate. Estos algoritmos fueron traducidos a lenguaje SQL para poder implementarlos en una base de datos relacional, por medio de procedimientos almacenados.

Para el experimento se creó una base de datos que permitiera almacenar la información de las direcciones de la zona 10 y una aplicación web para poder realizar las pruebas con los algoritmos a comparar. Se realizó una encuesta a miembros de la Universidad del Valle de Guatemala y el algoritmo favorecido fue el de Dijkstra.

Adicionalmente se llevaron a cabo pruebas técnicas que evaluaron el rendimiento de los cuatro algoritmos involucrados. Estas pruebas consisten en generar rutas entre todas las direcciones que se guardaron en la base de datos para determinar el promedio que cada algoritmo se tarda en generar cada ruta y el total de tiempo que se tarda en generar todas las rutas. Se determinó que el algoritmo de Dijkstra permite a los usuarios encontrar el camino más corto entre dos direcciones en un menor tiempo.

I. INTRODUCCIÓN

Este documento tiene por finalidad adaptar un algoritmo que permita encontrar la ruta más corta entre dos direcciones en la ciudad de Guatemala, de una forma rápida y gratuita, que pueda ser utilizado por cualquier persona que cuente con una computadora e Internet.

La primera parte del trabajo consiste en una investigación sobre los algoritmos especializados en encontrar las rutas más cortas. Entre los cuales están el algoritmo de Dijkstra, el algoritmo de Bellman-Ford y el algoritmo de Floyd-Warshall. Se investigó la forma en cómo cada algoritmo encuentra la ruta más corta y se revisó el pseudocódigo para realizar la traducción a procedimientos almacenados de SQL Server. Se diseñó la estructura de las tablas para almacenar la información de las direcciones y se ingresaron las direcciones de la zona 10 de la capital de Guatemala.

La segunda parte del trabajo consiste en las pruebas de los algoritmos, para esto se realizó una encuesta a los miembros de la Universidad del Valle de Guatemala en la cual se le pedía al usuario que generara dos rutas predefinidas y una ruta personal con los tres algoritmos implementados y el algoritmo de Averíguate. Con esto se pudo obtener información sobre los navegadores que los usuarios utilizaron, el ancho de banda con que contaban y la apreciación subjetiva sobre el algoritmo que encontraba la ruta más corta en el menor tiempo. La prueba técnica se llevó a cabo con una aplicación que genera las rutas, con los cuatro algoritmos, entre todas las direcciones que se encontraban en la base de datos, almacenando el tiempo que cada algoritmo se tarda en encontrar cada ruta. Se obtuvo el tiempo total que cada algoritmo demora en generar todas las rutas y el tiempo promedio por ruta. Luego se realizó una gráfica para poder comparar los resultados obtenidos.

Se pudo determinar que el algoritmo de Dijkstra implementado en la base de datos es el más adecuado porque redujo el tiempo para generar la ruta más corta entre dos puntos y demostró consistencia al ser probado con distintas direcciones.

Se recomienda crear un servicio web que brinde la funcionalidad del algoritmo sin utilizar directamente la interfaz implementada en este trabajo, lo cual permite que este algoritmo se pueda utilizar en diversas aplicaciones que necesiten generar rutas. También se puede mejorar el trabajo mostrando resultados de publicidad georeferenciada lo cual puede mejorar la experiencia del usuario al mostrar posibles sitios de interés que se encuentran en la ruta que se está buscando.

II. OBJETIVOS

A. Objetivo general

Implementar un algoritmo incrustado en una base de datos relacional que permita determinar la ruta más óptima entre dos direcciones, en el menor tiempo de respuesta mediante una interfaz web que a su vez muestre la secuencia de pasos que recorren la ruta entre los dos puntos.

B. Objetivos específicos

- Evaluación de tres algoritmos (Algoritmo de Warshall, Algoritmo de Bellman-Ford y Algoritmo de Dijkstra) para obtener rutas más cortas versus el algoritmo implementado en el Megaproyecto de Averiguate.
- Implementar el algoritmo, que permita encontrar la ruta en el menor tiempo, en una base de datos relacional
- Crear una aplicación web que permita la consulta de rutas entre direcciones de la Ciudad de Guatemala
- Aportar un recurso de uso público y gratuito que permita satisfacer la necesidad de rutas óptimas entre direcciones de la ciudad de Guatemala

III. MARCO TEÓRICO

A. Algoritmo de la ruta más corta

El problema de la ruta más corta surge cuando en algún proceso es necesario recorrer dos puntos dentro de una red lo más rápido, económico o rentable posible. En el problema de la ruta más corta se cuenta con un conjunto de calles que tienen una vía (que no tienen ciclos), cada una de las calles puede tener definido un costo establecido el cual puede representar tiempo o distancia. Si tomamos cualquier ruta para llegar de un nodo a otro consideraremos el costo de la ruta como la suma de los costos de cada uno de los arcos que se utilizan para completar la ruta. Este problema consiste en determinar la ruta con el menor costo posible para llegar de un nodo a otro. [C]

B. Algoritmo de Dijkstra

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo dirigido y con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959. [G]

La idea subyacente en este algoritmo consiste en ir explorando los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo). [G]

C. Algoritmo Bellman-Ford

El algoritmo Bellman-Ford permite calcular la ruta más corta en grafos que cuentan con vértices con pesos, a diferencia del algoritmo de Dijkstra los pesos de estos vértices pueden ser negativos. Este algoritmo fue creado por Richard Bellman y Lester Ford, Jr. Este algoritmo recorre los nodos pasando a lo sumo una vez por cada nodo y lo recorre V-1 veces, donde V es el número de vértices del grafo. [H]

D. Algoritmo Floyd-Warshall

Este algoritmo detecta el camino más corto en un grafo dirigido, a diferencia de los algoritmos anteriores, al ejecutar este algoritmo una vez se pueden encontrar los caminos más cortos entre todos los pares de vértices del grafo. Este algoritmo es un ejemplo de programación dinámica. Este algoritmo compara todos los caminos más cortos entre cada par de vértices, hasta llegar a obtener un camino óptimo. Este algoritmo usa recursión para poder realizar las evaluaciones de las rutas del grafo. [R]

E. Bases de datos

Conjunto de datos almacenados en un soporte informático no volátil. Los datos están interrelacionados y estructurados de acuerdo con un modelo de datos. [I]

1. **Modelo de datos.** Un modelo de datos es un conjunto de conceptos que permiten describir los datos, escondiendo los detalles del almacenamiento físico. Un esquema es la descripción de una colección de datos en particular, usando un modelo de datos. [I]

2. **Modelo Relacional.** Los datos son representados por medio de filas y columnas de una tabla. Cada relación tiene un esquema, que describe las columnas o campos. Con los conceptos antes mencionados se puede ahora definir a las Bases de datos relacionales como una colección de datos relacionados entre sí, que pueden ser accedidos en forma simultánea por distintos usuarios y/o aplicaciones en forma integrada para poder tomar decisiones, garantizando seguridad, consistencia, integridad y redundancia mínima. [N]

IV. ANTECEDENTES

A. Sistema base

Este trabajo surgió de la idea de mejorar el algoritmo de búsqueda del megaproyecto *Sistema de posicionamiento y trazado de rutas en Guatemala,* pero es independiente de este ya que los algoritmos de este trabajo pueden ser utilizados en diversas aplicaciones de búsqueda de rutas.

El megaproyecto *Sistema de posicionamiento y trazado de rutas en Guatemala* brinda un sistema de consulta de datos geográficamente referenciados. Las direcciones y sus puntos geográficos se ingresan a través del sitio web www.averiguate.info para luego ser consultados permitiendo obtener rutas entre dos direcciones. [U]

1. **Ingreso de datos en la página Web.** En la página web se encuentra la opción "Dibuja tu mapa", al entrar a esta página aparece un mapa de Guatemala con botones que permiten agregar puntos y líneas, mover el mapa, modificar, borrar y guardar los cambios. El ingreso de datos se hace a través de controles para agregar atributos y valores que representan una dirección o nombre de un lugar. Para ingresar una dirección se debe de ubicar el lugar en el mapa, se selecciona la opción de crear punto, se coloca sobre el lugar deseado y se agrega la dirección como un atributo. [U]

2. **Almacenamiento de datos en la BD.** Los datos son almacenados en tablas que representan arcos, nodos, vías y atributos. Un arco contiene la información de la calle que conecta dos intersecciones por lo que puede almacenar el número inicial, número final, zona, calle, lote, sección y manzana dependiendo de la dirección ingresada. Cada arco está relacionado con uno o más nodos, que representan las casas que se encuentran en esa calle. Cada arco también cuenta con información de la vía, esta información es utilizada para el algoritmo que traza las rutas. Los atributos se asignan a los arcos para poder agregar información relevante que sirva para identificar un lugar más fácilmente. El algoritmo para encontrar rutas utiliza el largo de cada arco para definirle un peso y evaluar qué camino es más corto. [U]

3. Consulta de datos en la página. Web La página para consultar datos permite buscar un lugar en base a categorías predefinidas por los usuarios. Primero se ingresa el nombre de una categoría o se selecciona la categoría de la lista y luego se escoge el lugar deseado. El resultado es una página en donde se muestra el lugar ubicado en el mapa. Adicionalmente se pueden obtener la descripción y una imagen del lugar, si se han ingresado. [U]

También se pueden consultar rutas en la opción "Busca tu ruta", en la cual se muestran dos cuadros de texto para ingresar la dirección origen y destino de la ruta. Al presionar el botón de Ruta, se muestra en pantalla el recorrido para llegar al destino. [U]

4. Consulta a la base de datos. Para realizar consultas a la base de datos, la dirección ingresada por el usuario se divide en partes para determinar la calle o avenida, el número de casa y la zona. Con base a esto se busca el vértice que representa a la calle o avenida con su zona correspondiente y sobre este vértice se busca el número de casa que se ingresó. [U]

a. Algoritmo utilizado

El algoritmo que se utiliza es una combinación entre el algoritmo de Dijkstra y algoritmos genéticos, siendo implementados en el código con pilas y colas. Este algoritmo funciona con cuatro entidades que manejan la información y dos entidades que permiten el acceso a tal información. A continuación se detallan las entidades de manejo de información:

- Arco: Representa un segmento de calle y, junto con los nodos, forma un grafo que permite la búsqueda de rutas entre dos direcciones. Se define un arco como la distancia que existe entre dos nodos. [U]
- Nodos: Elementos geográficos que permiten definir y delimitar los arcos. Además, indican la adyacencia de diferentes arcos. [U]
- Ruta: Representa una lista de arcos los cuales definen un camino ó ruta para irse de un arco inicial a un arco final. Tanto el arco inicial como el final debe estar asociado a direcciones. [U]

- Dirección: Representa una dirección válida en el mapa. Debe existir un mapeo entre dirección y arco, de forma que a partir de una dirección se pueda obtener el arco correspondiente. [U]

El pseudocódigo de este algoritmo se muestra a continuación:

1. *Se tienen como parámetros: dirección origen, dirección destino y costo*
2. *Encontrar los arcos de las direcciones*
    - *Arc origen = GetArc (dirección origen)*
    - *Arc destino = GetArc (dirección destino)*
3. *Crear una pila rt que guarda las listas de arcos (rutas)*
4. *Crear la ruta que será retornada*
5. *Crear dos listas de arcos, una temporal y otra que guarda la óptima*
6. *Crear una variable que almacene el menor costo e inicializar a infinito*
7. *Crear una pila de costos y un costo actual inicializado a cero*
8. *Crear una lista de arcos inicial*
    - *Añadir el arco origen a la lista*
    - *Meter la lista en la pila rt*
    - *Meter el costo del arco origen en la pila de costos*
9. *Mientras la pila todavía tenga elementos:*
    - *Sacar un elemento de la pila rt y asignar a lista actual*
    - *Sacar un elemento de la pila de costos y asignar a costo actual*
    - *Poner el origen como el último arco de la lista actual*
    - *Buscar los arcos adyacentes al origen*
    - *Para cada arco adyacente al origen:*
        - *Para cada arco en la lista actual:*
            - *Añadir a una lista temporal*
        - *Si el arco actual es diferente al anterior y minima es mayor costo actual:*
            - *Agregar el arco adyacente a la lista temporal*
            - *Agregar la lista temporal a la pila rt*
            - *Agregar el costo correspondiente a la pila de costo*
    - *Si el arco origen es el mismo que el arco destino:*
        - *Si costo actual es menor a minima:*
            - *Asignar lista resultado a lista actual*
            - *Asignar minima a costo acutal*
10. *Crear el objeto Route*
11. *Para cada elemento de la lista resultado*

- *Agregar arco a ruta*

12. Ejecutar la ruta para cargar sus propiedades

13. *Retornar la ruta óptima*

5. Forma de mostrar resultados en la página Web. Los resultados son desplegados en el mapa que aparece en pantalla. Cada resultado se representa con líneas si son calles y puntos para direcciones o intersecciones específicas. En la pantalla se puede cambiar la vista a imagen de mapa, imagen satelital o imagen de terreno. La imagen de mapa muestra el dibujo de las calles y su respectiva señalización. La imagen satelital muestra una foto de toda el área y los nombres de las calles. La imagen de terreno muestra una superposición de las imágenes de mapa y satelital. Estas imágenes provienen de Google Maps, Yahoo Maps o Live Search Maps. El usuario puede seleccionar que proveedor prefiere para las imágenes y el tipo de imagen que desea ver. Los resultados se dibujan sobre las imágenes. [U]

B. Algoritmos y aplicaciones actuales.

Existen varios algoritmos de búsqueda del camino más corto que han sido implementados de diferentes maneras y lenguajes. Estas implementaciones son utilizadas en sistemas como Google Maps, Yahoo Maps, Live Maps y ArcGis. Cada algoritmo toma como base la información de una base de datos y busca las rutas entre nodos que mejor cumplan con ciertas condiciones. Estas condiciones se llaman pesos y son características que definen que ruta es más óptima que otra. Por ejemplo, un peso asignado a una calle es la distancia, una calle con una distancia menor que otra calle, nos va a generar una ruta más corta. Entre los pesos se puede encontrar distancias, tráfico y reductores de velocidad. [D]

Arcgis cuenta con la capacidad de ser ejecutado sobre una máquina cliente o un servidor mediante una página web y cuenta con características más avanzadas para edición de rutas. Este programa permite trabajar con mapas propios, lo cual facilita enfocarse en áreas específicas de un lugar. Por ejemplo, si se tiene un mapa de una colonia con una alta resolución, se pueden trazar con mayor detalle las calles y separar las casas en el mapa para ingresar información relevante del mapa. Toda la información que

se ingresa puede ser utilizada para generar rutas dentro del programa. Debido a que este programa utiliza código propietario y no se cuenta con formas exactas de evaluar cada ruta, el tiempo que se tarda este programa en generar las rutas no puede ser medido exactamente ni se conoce la complejidad de los algoritmos utilizados. Arcgis actualmente es utilizado en el departamento de GIS de la Universidad del Valle de Guatemala y el departamento de GIS de la Universidad Rafael Landivar. [D]

C. Importancia de los sistemas basados en posiciones geográficas.

Los proyectos que se han desarrollado para trazar rutas impactan la forma en cómo las personas llegan de un lugar a otro. Actualmente aplicaciones como Google maps o ArcGis son utilizadas por empresas que deben generar rutas óptimas para distribución de productos. El beneficio de generar una ruta óptima es disminuir los recursos necesarios para poder llegar al destino. Entre los costos tenemos, combustible, tiempo y cantidad de vehículos. Otro uso que se le da a Google maps actualmente es en la industria del turismo, donde una persona que no conoce un país puede revisar el recorrido que necesita para llegar a su destino y hacerlo de una forma más fácil. En la actualidad la facilidad de contar con dispositivos móviles hace que consultar en tiempo real las rutas sea rápido y exacto. [O]

Los sistemas basados en posiciones geográficas han logrado revolucionar la forma en cómo personas y empresas se conectan. Actualmente Google maps permite enviar publicidad dirigida a lugares geográficos en donde se encuentran las personas en ese momento. [O]

V. DELIMITACIÓN E IMPACTO DEL TEMA

A. Delimitación

- Ingreso de los datos de la zona 10 de la capital para generar rutas entre distintas ubicaciones.
- Las rutas tomarán en cuenta las vías de las calles y la distancia del recorrido. Para cada par de nodos y arcos paralelos se tomará como una calle de doble vía. De lo contrario la vía va en la dirección que tiene el arco.
- Si la estructura de la base de datos del megaproyecto es compatible con el algoritmo a implementar, los datos se almacenarán en esas tablas. De lo contrario se crearán tablas definidas para los datos y serán validados en el ingreso a la base de datos.
- El algoritmo de ruteo implementado podrá crear rutas en base a la información que esté ingresada en el sistema.
- Si la estructura de la base de datos del megaproyecto es compatible con la base de datos del algoritmo a implementar se utilizará la aplicación de www.averiguate.info para desplegar los resultados, de lo contrario se creará una aplicación web a través de la cual se podrán ingresar dos direcciones y se listará el recorrido que se debe seguir para llegar al destino.
- La aplicación web se encontrará en línea (www.GuateRutas.com) y podrá ser accedida a través de Internet desde cualquier computadora que cuente con acceso a Internet.

B. Impacto

Este trabajo impactará principalmente a personas que necesitan obtener rutas entre dos direcciones, de la zona 10 de Guatemala. Al implementar el algoritmo sobre una base de datos, ésta puede unirse a cualquier plataforma o lenguaje de programación ampliando la cantidad de usuarios que pueden beneficiarse y disminuyendo el tiempo de desarrollo de las aplicaciones.

La población, en general puede consultar las direcciones almacenadas en la base de datos a través de la aplicación Web y obtener las indicaciones para llegar de una dirección a otra. La implementación del algoritmo para generar rutas también facilitará la creación de sistemas que permitan trabajar con datos geográficamente referenciados y necesiten la generación de rutas óptimas. El principal beneficio para estas aplicaciones es la reutilización de los algoritmos para encontrar las rutas y la información disponible en la base de datos. Esta información se encuentra disponible en línea, por lo que estos sistemas realizarán las consultas donde se encuentre implementado el algoritmo y podrán manipular la información obtenida en base a los datos requeridos por cada aplicación.

Se implementarán tres algoritmos de los cuales se seleccionará el que menos tiempo utilice para encontrar la ruta más corta. Además, al implementar el algoritmo en la base de datos, se reduce el número de llamadas al servidor y el ancho de banda utilizado entre la generación de la ruta y la aplicación del cliente, enviando únicamente el resultado al usuario final. Todas las tablas que están en el documento contienen los tiempos que se obtienen en la base de datos al iniciar y finalizar cada ruta, no toman en cuenta el ancho de banda. Este ancho de banda que se me menciona es para ver que únicamente se hace una llamada al servidor de base de datos, la base de datos genera la ruta y la devuelve, en lugar de una llamada por cada dirección que se necesita desde la aplicación de los usuarios, ya que la ruta se genera en la aplicación.

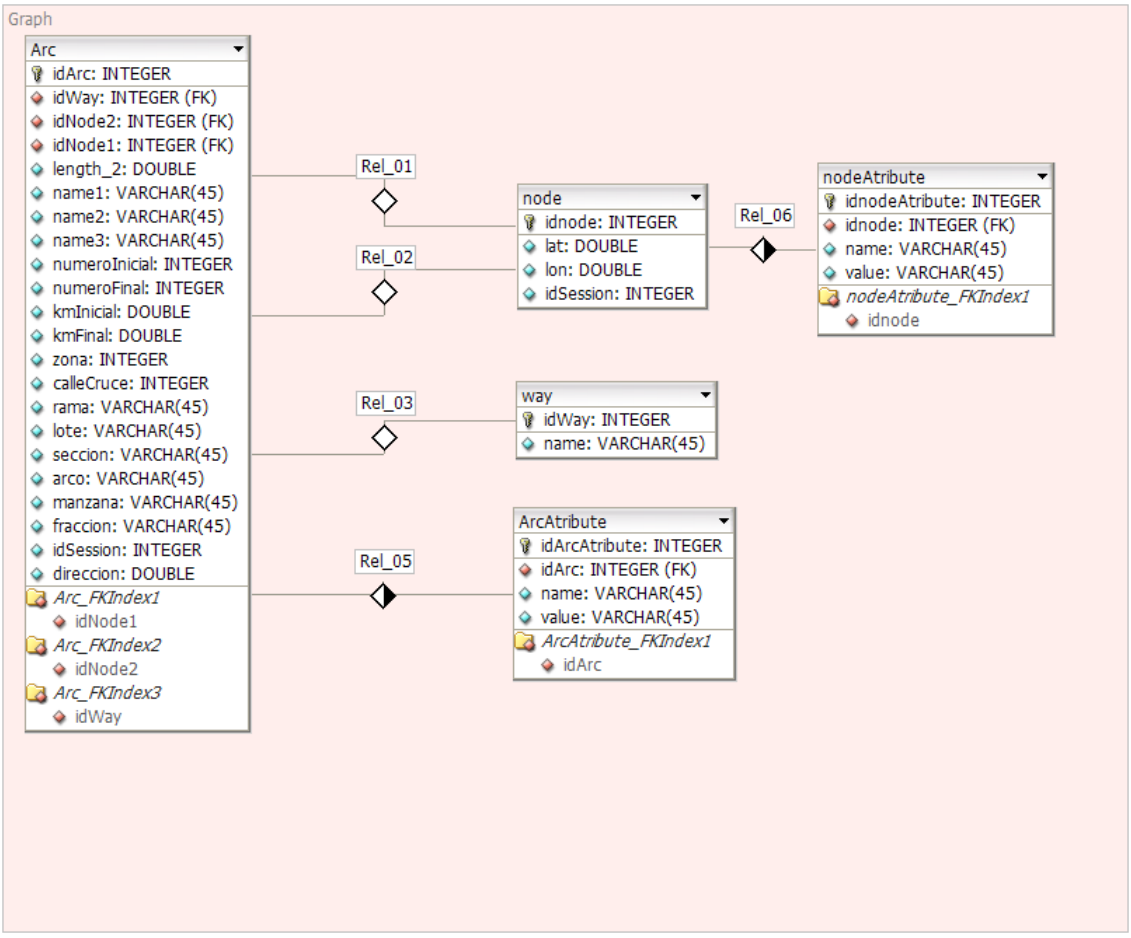
Los turistas pueden utilizar la aplicación para buscar lugares de interés y obtener indicaciones para llegar a sitios turísticos que deseen visitar. Al momento de mostrar el resultado se puede desplegar publicidad de lugares que se encuentren en el transcurso de la ruta, que tengan cierta relación a la dirección destino.

VI. ANÁLISIS Y DISEÑO DE LA EVALUACIÓN DE ALGORITMOS

A. Análisis de la base de datos de Averiguate

El primer paso es revisar la estructura de la base de datos del megaproyecto *Sistema de posicionamiento y trazado de rutas en Guatemala*. Se realizará una lista de las tablas, campos, procedimientos almacenados y funciones. Se compararán las tablas con las requeridas por los algoritmos a probar (**Nodes** y **Paths).** Si cumplen con los mismos campos y tipos de datos se utilizará la base de datos actual para implementar los algoritmos. [U]

Ilustración 1 Diseño de la base de datos de averiguate.info



Las tablas de la aplicación de averiguate.info son:

- Arc: Representa la entidad arco del diagrama entidad relación. Un arco está compuesto por dos nodos, llaves foráneas idNode1 y idNode2, los cuales lo delimitan y restringen su valor a nodos válidos previamente almacenados en la tabla Node. El campo idSession se utiliza para mantener bloqueos en los arcos, de forma que si la sesión X está utilizando cierto conjunto de arcos, estos deben tener el campo idSession con su valor X de la sesión. Este bloqueo impide que cualquier otro usuario pueda modificar dichos arcos. [U]
- ArcAtribute: Permite definir atributos adicionales a los ya definidos en la tabla arco, y permite asociar dichos atributos con su arco correspondiente. [U]
- Node: Representa la entidad nodo del diagrama entidad relación. Esta tabla contiene las coordenadas latitud y longitud del mapa. Al igual que la tabla Arc, la tabla Node contiene el campo idSession que indica que sesión está utilizando el nodo actualmente. [U]
- NodeAtribute: Permite definir atributos a los nodos, y permite asociar dichos atributos con su nodo correspondiente. [U]
- Way: Catálogo que almacena los tipos de vía que tiene cada arco. La vía puede tener tres valores: [U]
    - Una vía: Permite moverse del nodo origen al nodo destino.
    - Vía contraria: Permite moverse del nodo destino al nodo origen.
    - Doble vía: Permite moverse en ambas direcciones del arco.

B. Análisis de la base de datos a implementarse

La base de datos que será implementada almacenará la información geográfica de tal forma que pueda ser utilizada con los algoritmos que se evaluarán. La información necesaria para encontrar una ruta son las direcciones o nodos y los vértices que unen a dos direcciones.

C. Diseño de la base de datos a implementarse

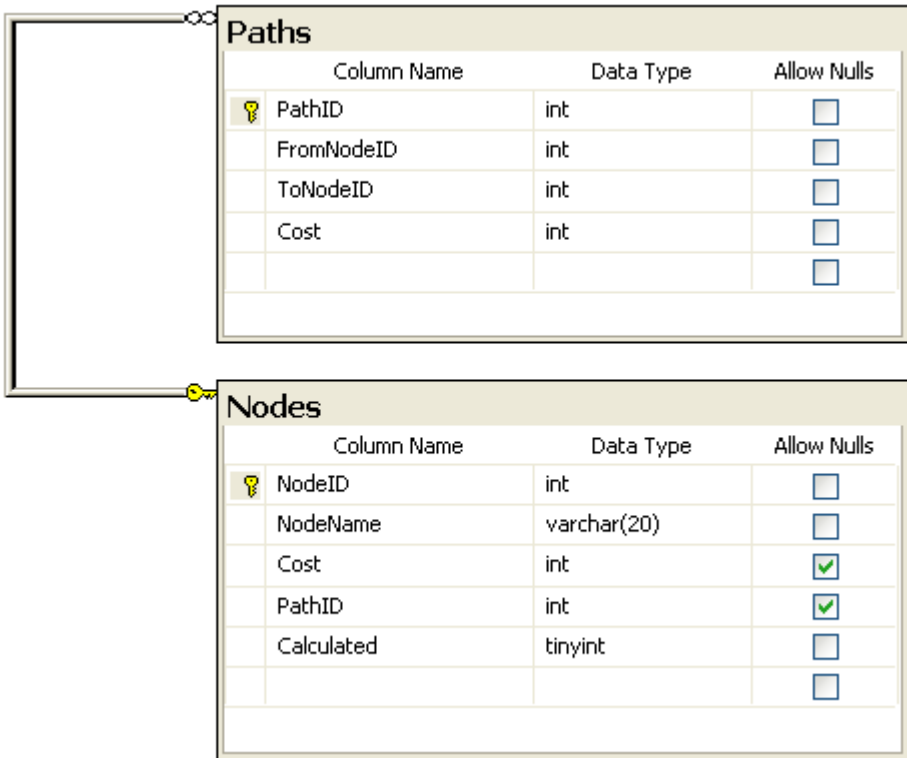
Las direcciones en la base de datos de Averiguate se encuentran divididas en varios campos y estas se almacenan como arcos, para los algoritmos que serán implementados se necesita que el nodo sea una dirección especifica y los datos de atributos se almacenen en una tabla aparte. La tabla de vías no es necesaria ya que la conexión entre dos nodos (vértices) representa la dirección de la vía. Por lo tanto el actual diseño de la base de datos de Averiguate no es compatible con la estructura de la base de datos que utilizaran los algoritmos a implementarse. Por lo que se crearán nuevas tablas para almacenar la información de las direcciones para los algoritmos.

Las tablas necesarias para almacenar la información serán:

- Nodes: esta tabla almacena la información de cada nodo. Entre los datos se encuentran: el identificador único del nodo, nombre del nodo (dirección), costo que representa pasar por ese nodo, identificador de ruta y un indicador para saber si ese nodo ya fue utilizado en la ruta.
- Paths: contiene la información de los nodos que están conectados entre sí. Entre los datos se encuentran: el identificador único de la ruta, el nodo de partida, el nodo destino y el costo.

El servidor Web y el servidor de Base de Datos a utilizarse son de la compañía Drundo, cuentan con el sistema operativo Windows Server 2003 y la base de datos MS SQL Server 2005. El procesador es Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core y 4GB de memoria RAM. La aplicación para consultar las rutas se realizará en ASP.NET, por la interoperabilidad que tiene con MS SQL Server 2005.

Ilustración 2 Tablas de información de las rutas y nodos



Para la búsqueda de las rutas se utilizarán procedimientos almacenados. Cada algoritmo contará con un conjunto de procedimientos que le permitan obtener la ruta entre dos direcciones.

Se utilizarán tres procedimientos que serán compartidos por los algoritmos para poder realizar operaciones básicas sobre las tablas. Las operaciones son:

- **AgregarRuta**: recibe como parámetros el nodo origen, nodo destino y el costo para almacenar una nueva ruta o actualizar una ruta previamente ingresada.
- **LimpiarMapa**: coloca el costo de todos los nodos en 0 para poder calcular una nueva ruta.
- **InicializarMapa**: elimina todos los datos de la tabla Nodes y la tabla Paths.

Para generar las rutas se utilizará un procedimiento almacenado por cada algoritmo:

1. **Algoritmo de Dijkstra: DijkstraResolver.** Genera la ruta entre los nodos origen y destino utilizando el algoritmo de Dijkstra, toma como parámetros la dirección inicial y final del recorrido. Evalúa los pesos de cada camino desde la dirección inicial hasta encontrar la dirección final devolviendo el recorrido de menor peso.

2. **Algoritmo Bellman-Ford: BellmanFordResolver.** Genera la ruta entre los nodos origen y destino utilizando el algoritmo de Bellman-Ford, este algoritmo cuenta con la habilidad de encontrar el recorrido aún cuando se encuentre pesos negativos en los caminos.

3. **Algoritmo Floyd-Warshall: FloydWarshallResolver.** Genera la ruta entre los nodos origen y destino utilizando el algoritmo de Floyd-Warshall. Al calcular la ruta entre dos direcciones este algoritmo calcula todas las rutas óptimas para llegar desde el origen a cualquier destino.

D. Ingreso de datos de prueba

Se recolectará la información de diez direcciones por cada cuadra de la zona 10 de la capital. Esta información será ingresada en la base de datos, en la tabla de nodos y rutas. Las pruebas se realizaran con 104 direcciones, 147 direcciones y 208 direcciones.

E. Evaluación de tiempos entre algoritmos

Se realizará una lista con direcciones origen y destino, en base a la cual se generarán rutas, con cada uno de los tres algoritmos, para determinar el algoritmo que devuelva un resultado más rápidamente.

Para medir la velocidad se creará una tabla de tiempos en la base de datos, en donde se ingresará el tiempo que se tarda cada algoritmo en calcular cada ruta. Se hará una comparación entre el tiempo por ruta entre los tres algoritmos y el tiempo total de todas las rutas generadas. Cada vez que se mande a llamar un procedimiento para generar una ruta con cualquiera de los algoritmos, se ingresa la hora en el tiempo inicial y antes de devolver los resultados se ingresa la hora en el campo tiempo final. Las horas se obtienen de la base de datos con la función GetDate() de SQL Server. Luego se tomará el

tiempo empleado en cada ruta y se comparará entre los algoritmos. Para obtener el tiempo en la aplicación Averiguate se usará la función DateTime.now de C#.

F. Análisis de pseudocódigo

Para determinar la complejidad de los algoritmos se verificarán los siguientes aspectos: [A]

- sentencias sencillas (asignación de valores)
- secuencia (;)
- decisión (*if*)
- iteraciones(*while, for*)
- llamadas a procedimientos

El pseudocódigo será traducido a instrucciones de SQL para poder crear los procedimientos almacenados y evaluar el rendimiento de los algoritmos. El pseudocódigo de cada algoritmo se presenta a continuación:

1. **Pseudocódigo algoritmo de Dijkstra.** Este algoritmo recorre los vértices del grafo por medio de iteraciones *for* y *while*. El algoritmo realiza a lo más n-1 iteraciones, ya que en cada iteración se añade un vértice a la lista de resultado (Q). Para estimar el número total de operaciones basta estimar el número de operaciones que se llevan a cabo en cada iteración. Se puede identificar el vértice origen entre los que no están en Q realizando n-1 comparaciones o menos.

Después se hace una suma y una comparación para actualizar el valor de cada uno de los vértices que no están en Q. Por tanto, en cada iteración se realizan a lo sumo 2(n-1) operaciones, ya que no puede haber más de n-1 valores por actualizar en cada iteración. Como no se realizan más de n-1 iteraciones, cada una de las cuales supone a lo más 2(n-1) operaciones, por lo que el algoritmo de Dijkstra realiza $O(n^2)$ operaciones. [V]

```
function Dijkstra(Graph, source):
    for each vertex v in Graph: // Inicialización
       // Función de distancia desconocida del nodo de  partida a destino
       dist[v] := infinity
       previous[v] := undefined // Nodo previo en ruta del inicio
    dist[source] := 0                  // Distancia del destino
```

```
Q := the set of all nodes in Graph
// Todos los nodos en el grafo no están optimizados por lo que están en Q
while Q is not empty:          // Ciclo principal
   u := vertex in Q with smallest dist[]
   if dist[u] = infinity:
      break                    // vértices restantes no son deseables
   remove u from Q
   for each neighbor v of u: // donde v no ha sido quitado de Q.
      alt := dist[u] + dist_between(u, v)
      if alt < dist[v]:        // Relax (u,v,a)
         dist[v] := alt
         previous[v] := u
return previous[]
```

2. **Pseudocódigo algoritmo Bellman-Ford.** La primera iteración recorre el grafo asignando las distancias a infinito y eliminado los predecesores. Como esta iteración está compuesta sólo por sentencias se dice que tiene una complejidad N * O (1) = O(n). La siguiente iteración está compuesta por otra iteración de tipo *for* por lo que su complejidad está determinada por N * N * O (1) = $O(n^2)$. Siendo esta la complejidad mayor del algoritmo decimos que la complejidad del algoritmo Bellman-Ford es O (n^2). [T]

```
BellmanFord(Grafo G, nodo_fuente s)
    // inicialización del grafo. Se ponen las distancias a INFINITO menos el
    // nodo fuente que tiene distancia 0
    for v ∈ V[G] do
        distancia[v]=INFINITO
        predecesor[v]=NIL
    distancia[s]=0
    // Se relaja cada arista del grafo tantas veces como número de
    // nodos -1 haya en el grafo
    for i=1 to |V[G]-1| do
        for (u,v) ∈ E[G] do
            if distancia[v]>distancia[u] + peso(u,v) then
                distancia[v] = distancia[u] + peso (u,v)
                predecesor[v] = u
    // Se comprueba si hay ciclos negativos
    for (u,v) ∈ E[G] do
        if distancia[v] > distancia[u] + peso(u,v) then
            print ("Hay ciclo negativo")
            return FALSE
    return TRUE
```

3. **Pseudocódigo algoritmo Floyd-Warshall.** Este algoritmo utiliza una matriz bidimensional con la cual sólo se necesita recorrer cada una de sus posiciones evaluando el camino mínimo entre el origen y destino. Esta comparación se realiza con una iteración *for* desde 0 hasta el número de nodos menos 1 (n-1) y recorre con una iteración anidada las posiciones de la matriz. La complejidad de esta operación se obtiene por N * N * O (1) = O (n^2) y al ser la operación con mayor complejidad del algoritmo, la complejidad del algoritmo Floyd-Warshall es O (n^2). [T]

```
/* Se supone que la función pesoArista devuelve el coste del camino que va
   de i a j (infinito si no existe).
   También se asume que n es el número de vértices y pesoArista(i,i) = 0
*/

int camino[][];
/* Una matriz bidimensional. En cada paso del algoritmo, camino[i][j] es el
   camino mínimo de i hasta j usando valores intermedios de (1..k-1). Cada
   camino[i][j] es inicializado a pesoArista(i,j)
*/

procedimiento FloydWarshall ()
  para k: = 0 hasta n − 1
    para todo (i,j) en (0..n − 1)
      camino[i][j] = mín ( camino[i][j], camino[i][k]+camino[k][j]);
```

4. **Pseudocódigo algoritmo genético Averiguate.** Individuo: cada individuo representa la solución de la ruta y no debe contener puntos repetidos. El tamaño del individuo es el total de puntos que se encuentran en el mapa ya que puede haber casos en los que el camino más corto incluya el total de los puntos. El valor de ajuste (Fitness) es calculado para cada individuo dependiendo de la función de ajuste así como el rango que se le da a cada individuo en la población. [U]

El primer ciclo valida que la solución se encuentre en el valor de ajuste definido. Se inicializa el individuo y se itera hasta encontrar una solución, para buscar los puntos se hace un ciclo que obtiene el conjunto de los puntos para llegar a la dirección destino. Al tener 3 iteraciones anidadas se establece que la complejidad del algoritmo Genetico-Averiguate es O (n^3). [U]

adir: dirección del origen al destino
dir: dirección entre dos puntos adyacentes
twn: siguiente dirección a visitarse
While valor de ajuste del individuo sea mayor que la función de rendimiento
    dir = adir
    Inicializar un nuevo individuo
    Colocar el índice de 0 al punto origen
    Do
        S = el conjunto de puntos para llegar a la dirección
        While S esté vacío y existen más direcciones para buscar
            S = obtener los puntos en las otras direcciones
        End while
        If S está vacío
            twn = Destino
        Else
            twn = seleccionar un punto de S
        End if
        If twn no existe en el individuo
            Agregar twn al individuo
            dir = obtener dirección twn al destino
        Else
            twn = last town in the individual
        End if
    Repeat until (twn = DESTINATION)
End

G. Diseño de pseudocódigo

- **Traducción a SQL Dijkstra**

```
DECLARE
    @FromNodeID INT,
    @ToNodeID INT,
    @NodeID INT,
    @Cost INT,
    @PathID INT

SELECT  @FromNodeID = NodeID, @NodeID = NodeID
FROM    Nodes
WHERE   NodeName = @FromNodeName

IF @FromNodeID IS NULL
    BEGIN
        SELECT @FromNodeName = ISNULL(@FromNodeName, '')
        RAISERROR ('From node name "%s" can not be found.', 16, 1,
@FromNodeName)
```

```
            RETURN
    END

SELECT  @ToNodeID = NodeID
FROM    Nodes
WHERE   NodeName = @ToNodeName

IF @ToNodeID IS NULL
    BEGIN
            SELECT @ToNodeName = ISNULL(@ToNodeName, '')
            RAISERROR ('To node name "%s" can not be found.', 16, 1,
@ToNodeName)
            RETURN
    END

UPDATE  Nodes
SET     Cost = 0
WHERE   NodeID = @FromNodeID

WHILE @NodeID IS NOT NULL
    BEGIN
            UPDATE ToNodes
            SET ToNodes.Cost =  CASE
            WHEN ToNodes.Cost IS NULL THEN FromNodes.Cost +
Paths.Cost
            WHEN FromNodes.Cost + Paths.Cost < ToNodes.Cost THEN
FromNodes.Cost + Paths.Cost
            ELSE ToNodes.Cost
    END,

    ToNodes.PathID = Paths.PathID

    FROM Nodes AS FromNodes
    INNER JOIN Paths ON Paths.FromNodeID = FromNodes.NodeID
    INNER JOIN Nodes AS ToNodes ON ToNodes.NodeID =
Paths.ToNodeID
    WHERE       FromNodes.NodeID = @NodeID
    AND (ToNodes.Cost IS NULL OR FromNodes.Cost + Paths.Cost <
ToNodes.Cost)
    AND ToNodes.Calculated = 0

    UPDATE FromNodes
    SET FromNodes.Calculated = 1
    FROM Nodes AS FromNodes
    WHERE FromNodes.NodeID = @NodeID
```

```
    SELECT      @NodeID = NULL

    SELECT TOP 1        @NodeID = Nodes.NodeID
    FROM Nodes
    WHERE       Nodes.Calculated = 0  AND Nodes.Cost IS NOT NULL
    ORDER BY    Nodes.Cost
END
```

-- Tabla temporal para almacenar los nodos origen y destino. Evita utilizar recursión para recorrer las rutas.

```
CREATE TABLE #Map
    (
        RowID INT IDENTITY(-1, -1),
        FromNodeName VARCHAR(20),
        ToNodeName VARCHAR(20),
        Cost INT
    )

IF EXISTS (SELECT NULL FROM Nodes WHERE NodeID = @ToNodeID
AND Cost IS NULL)
    BEGIN
        SELECT FromNodeName, ToNodeName, Cost
        FROM #Map

        DROP TABLE #Map
        RETURN
    END

WHILE @FromNodeID <> @ToNodeID
    BEGIN
        SELECT      @FromNodeName = FromNodes.NodeName,
                    @ToNodeName = ToNodes.NodeName,
                    @Cost = ToNodes.Cost,
                    @PathID = ToNodes.PathID
        FROM        Nodes AS ToNodes
        INNER JOIN  Paths ON Paths.PathID = ToNodes.PathID
        INNER JOIN  Nodes AS FromNodes ON FromNodes.NodeID =
Paths.FromNodeID
        WHERE       ToNodes.NodeID = @ToNodeID

        INSERT      #Map
            (
                    FromNodeName,
                    ToNodeName,
                    Cost
            )
```

```
            VALUES
                (
                    @FromNodeName,
                    @ToNodeName,
                    @Cost
                )

            SELECT  @ToNodeID = Paths.FromNodeID
            FROM    Paths
            WHERE   Paths.PathID = @PathID
        END

    SELECT  FromNodeName, ToNodeName, Cost
    FROM    #Map
    ORDER BY    RowID

    DROP TABLE #Map
```

- **Traducción a SQL Bellman-Ford**

```
    PROCEDURE Path (
            @Start AS INTEGER ,
            @End   AS INTEGER ) AS
        SET NoCount ON
        DECLARE
            @Parent AS INTEGER
            SELECT @Parent = @End
-- Tabla temporal para almacenar los nodos origen y destino. Evita utilizar recursión para recorrer las rutas.
            CREATE TABLE #temp
                (
                    [SortId] INTEGER Identity ,
                    [JoinId] INTEGER
                )
                IF @@Error > 0 GOTO Cleanup
            INSERT
            INTO   #temp
                (
                    [JoinId]
                )
                VALUES
                (
                    @End
                )
                WHILE @Parent <> @Start
              AND NOT @Parent IS NULL BEGIN
            SELECT @Parent = ParentId
```

```
        FROM   Data
        WHERE  [Id] = @Parent
        INSERT
        INTO   #temp
            (
                [JoinId]
            )
            VALUES
            (
                @Parent
            )
    END
SELECT   Field1
FROM     #temp
      INNER JOIN Data
      ON       #temp.JoinId = Data.[Id]
ORDER BY #temp.SortId DESC
DROP TABLE #temp RETURN Cleanup:
DROP TABLE #temp RETURN SET NoCount OFF GO
```

- **Traducción a SQL Floyd-Warshall**

```
UPDATE tree
SET    nodenum = 1
WHERE  node_id = root_id;

    LOOP
    nxt := NULL;
        BEGIN
        SELECT MIN(node_id)
        INTO   nxt
        FROM   tree c
          WHERE  c.parent_id = last_visited
    AND c.nodenum IS NULL ;

        EXCEPTION
    WHEN no_data_found THEN
    nxt := NULL;
        END;
            IF nxt IS NULL THEN
            BEGIN
            SELECT MIN(c2.node_id)
                INTO   nxt
            FROM   tree c,
              tree c2
              WHERE  c.parent_id = c2.parent_id
        AND c.node_id   = last_visited
```

```
        AND c.node_id   < c2.node_id ;

        EXCEPTION
    WHEN no_data_found THEN
    nxt := NULL;
        END;
            END IF;
            IF nxt IS NULL THEN
            BEGIN
            SELECT parent_id
              INTO   nxt
        FROM   tree c
        WHERE  node_id = last_visited
            AND c.parent_id IS NOT NULL;

    EXCEPTION
    WHEN no_data_found THEN
    nxt := NULL;
    END;
      END IF;
    UPDATE tree
        SET    nodenum = last_nodenum + 1
    WHERE  node_id = nxt
    AND nodenum IS NULL;

    IF ( sql%rowcount     > 0 ) THEN
        last_nodenum := last_nodenum + 1;
        END IF;
    IF nxt IS NOT NULL THEN
last_visited := nxt;
                ELSE
                EXIT;
       END IF;
```

H. Aplicación para evaluación técnica de los algoritmos

Para evaluar los algoritmos se generará la ruta entre todos los puntos de la base de datos hacia todos los puntos de la base de datos. Para esto se creará una aplicación de escritorio que obtendrá todos los nodos que se encuentran en la base de datos del servidor remoto. Luego se ejecutará la función de la base de datos para encontrar la ruta de cada nodo con todos los nodos restantes. Se guardará el tiempo que se tarda en encontrar cada ruta. Esto se realizará con los tres algoritmos. Se compararán los tiempos independientes por rutas para saber la cantidad de veces que un algoritmo es más rápido que los otros.

También se obtendrá el tiempo total de encontrar todas las rutas por algoritmo para saber el que tiene el menor tiempo.

La prueba se realizará con 10816 rutas (n = 104), luego con 21632 rutas (n = 147) y finalmente con 43264 direcciones (n = 208); con el objetivo de evaluar el comportamiento de los algoritmos al ir incrementando la cantidad de datos con la cual trabajar. Otro de los objetivos de la prueba es evaluar la diferencia entre la teoría y la práctica de la complejidad de los algoritmos y la influencia que tiene el hardware para obtener las rutas rápidamente.

I. Encuestas para evaluar rendimiento

Se realizarán encuestas a miembros de la comunidad de la Universidad del Valle de Guatemala, para determinar cuál de los algoritmos les parece más eficiente, estas encuestas son únicamente para evaluar la percepción de los usuarios pero no impactan en los resultados prácticos obtenidos de las pruebas llevadas a cabo. Las encuestas sirvieron únicamente para ver la aceptación de la aplicación web de los usuarios ya que los datos objetivos de las pruebas sirvieron para ver cuál de los algoritmos era más rápido. El número de miembros a entrevistar está dado por muestreo probabilístico.

Para una población N = 1000 de estudiantes se necesita obtener información adecuada con error estándar menor de 0.1 al 90% de confiabilidad. La fórmula utilizada es:

$$n = \frac{n'}{1 + {}^{n'}/_{N}}$$

Siendo n’

$$n' = \frac{s^2}{\sigma^2}$$

σ^2 Es la varianza de la población respecto a determinadas variables. [F]

s^2 Es la varianza de la muestra, la cual podrá determinarse en términos de probabilidad como $s^2 = p(1-p)$ [F]

se Es error estándar que está dado por la diferencia entre ($\mu - \bar{x}$) la media poblacional y la media muestral. [F]

$(se)^2$ Es el error estándar al cuadrado, que nos servirá para determinar σ^2, por lo que $\sigma^2 = (se)^2$ es la varianza poblacional. [F]

Despejando tenemos,

N=1000

se = 0.05

$\sigma^2 = (se)^2 = (0.05)^2 = 0.0025$

$s^2 = p\ (1-p) = 0.9\ (1-0.9) = 0.09$

Por lo que

$n' = s^2/\sigma^2 = 0.09/0.0025 = 36$

$n = n' / (1 + n'/N) = 36/(1+36/1000) =$ **34.74**

Por lo que el número mínimo de miembros a ser encuestados es de 35.

Se pasará la encuesta a los miembros de la Universidad del Valle de Guatemala para que prueben los cuatro algoritmos. La encuesta se realizará por medio de correo electrónico.

La prueba consistirá en:

- Mostrar la información ingresada para que sepan qué rutas pueden generar
- Generar dos rutas con cada algoritmo
- Documentar observaciones y comentarios

La encuesta se encuentra en el anexo.

VII. DISEÑO DE LA IMPLEMENTACIÓN DE LA APLICACIÓN WEB

El objetivo de la aplicación es permitir a los usuarios obtener las calles y avenidas que deben tomarse para llegar de un punto A a un punto B, haciendo uso de uno de los algoritmos de la ruta más corta, implementado en la base de datos.

La aplicación tiene una interfaz Web ya que es necesario que los usuarios puedan conectarse remotamente a través de Internet y debe poderse utilizar por varios usuarios simultáneamente. [P]

- Entradas: la aplicación recibe la dirección origen y la dirección a la cual se desea llegar. Con un combo de opciones mutuamente excluyentes se selecciona el algoritmo con el cual se desea generar la ruta.
- Salidas: se mostrará una lista de las direcciones que deben seguirse para llegar al destino.
- Restricciones: Únicamente se pueden ingresar direcciones de la zona 10 de Guatemala. Las direcciones deben ser ingresadas de la siguiente manera:

Número de calle/avenida. Calle/Avenida. Número de casa. Zona 10

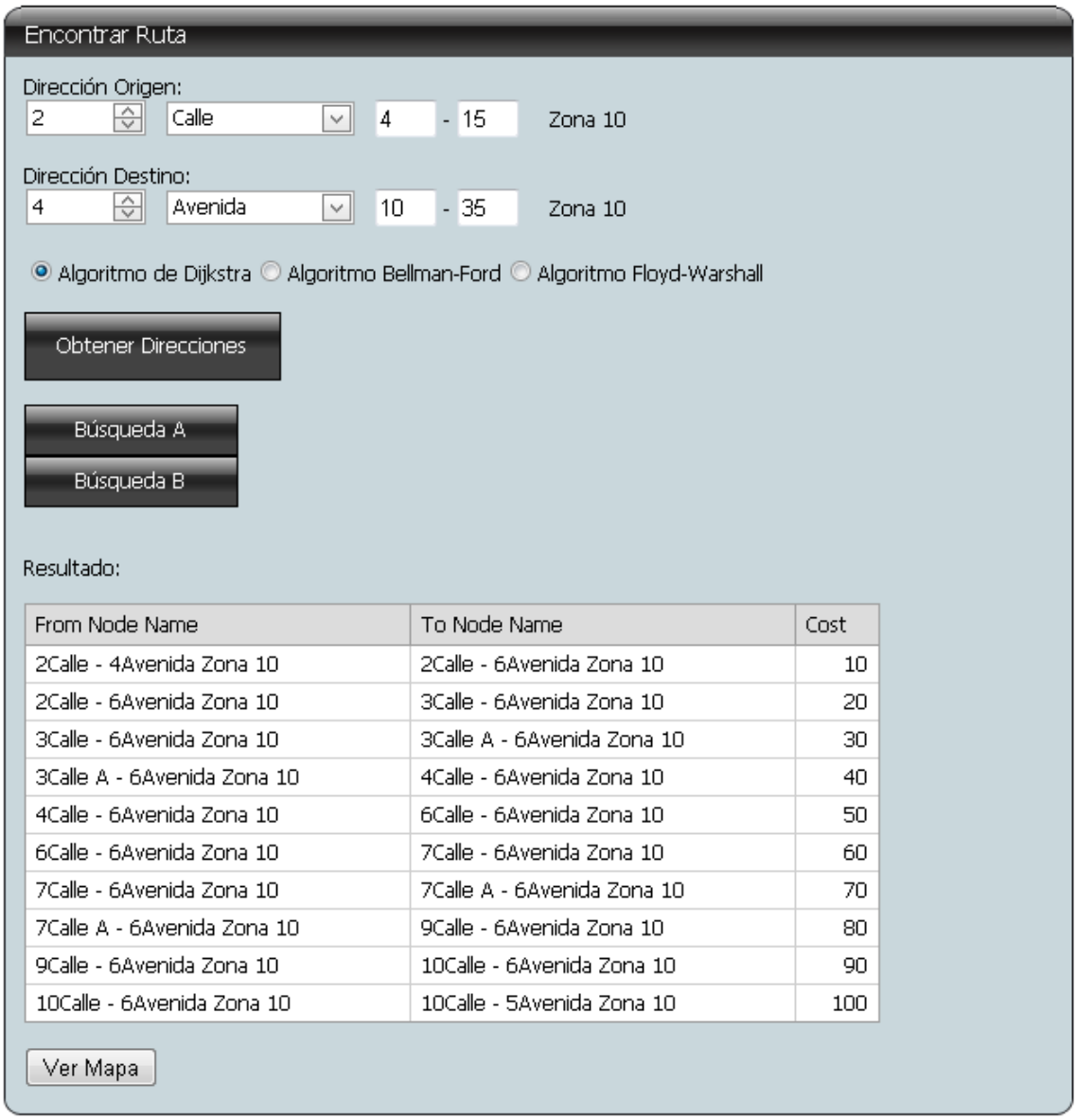
Ejemplo: *15 calle 16-23 zona 10*

En base a esto se determina que el nodo es el que corresponde a la intersección de la 15 calle con la 16 avenida.

- Resultados: se despliega la lista de intersecciones que deben seguirse para llegar al destino.

Ejemplo: Se necesita llegar de la 2 calle 2-12 zona 10 a la 7 calle 4-35 zona 10. El algoritmo genera la ruta y nos devuelve las direcciones por las que se debe pasar para llegar al destino. En la columna de la derecha se representa el costo para llegar de una dirección a otra en minutos.

Ilustración 3 Aplicación web

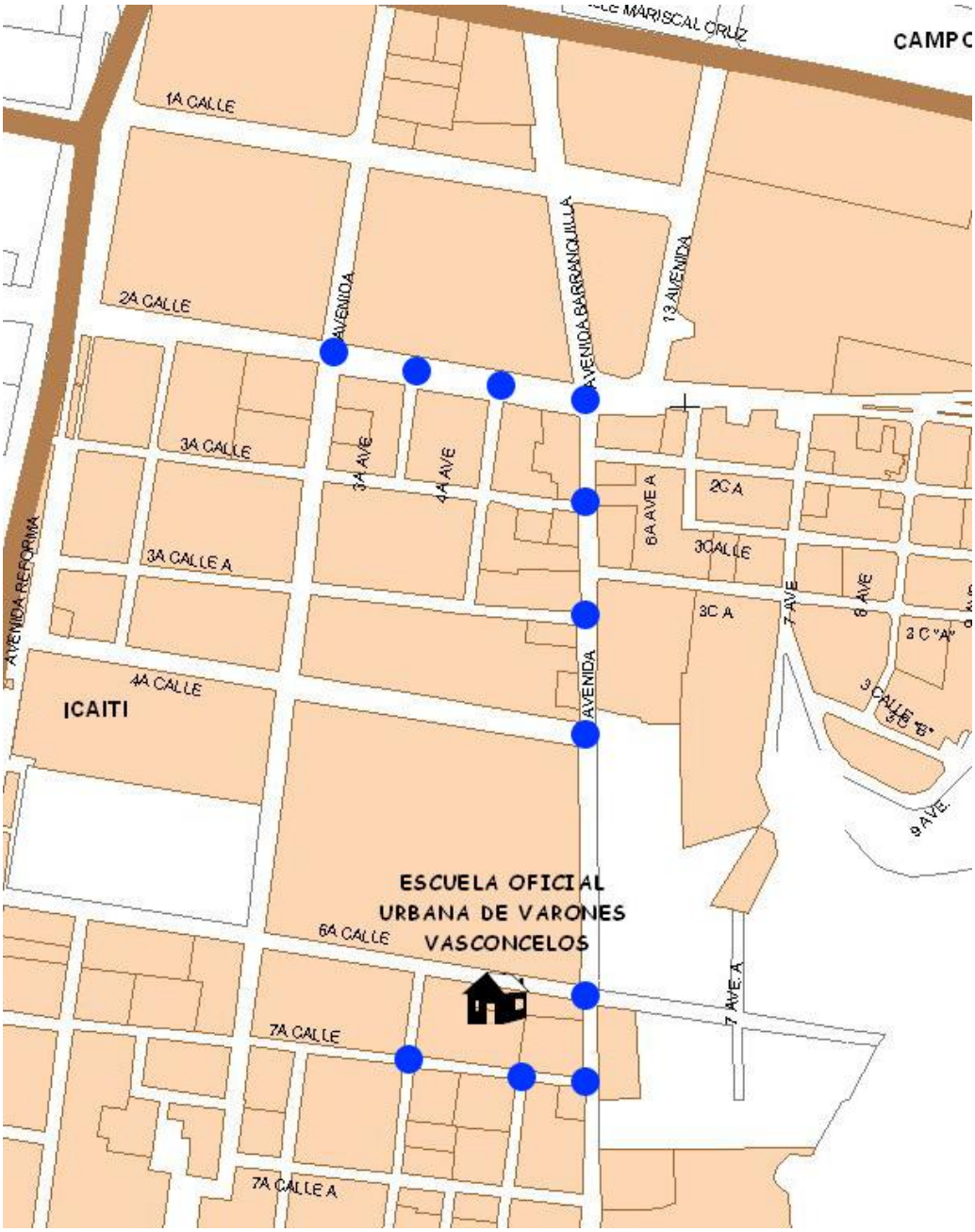


From Node Name	To Node Name	Cost
2Calle - 4Avenida Zona 10	2Calle - 6Avenida Zona 10	10
2Calle - 6Avenida Zona 10	3Calle - 6Avenida Zona 10	20
3Calle - 6Avenida Zona 10	3Calle A - 6Avenida Zona 10	30
3Calle A - 6Avenida Zona 10	4Calle - 6Avenida Zona 10	40
4Calle - 6Avenida Zona 10	6Calle - 6Avenida Zona 10	50
6Calle - 6Avenida Zona 10	7Calle - 6Avenida Zona 10	60
7Calle - 6Avenida Zona 10	7Calle A - 6Avenida Zona 10	70
7Calle A - 6Avenida Zona 10	9Calle - 6Avenida Zona 10	80
9Calle - 6Avenida Zona 10	10Calle - 6Avenida Zona 10	90
10Calle - 6Avenida Zona 10	10Calle - 5Avenida Zona 10	100

La prueba 1 consiste en buscar la ruta entre la 2 calle 4-15 hacia la 4 avenida 10-35. La prueba 2 es desde la 6 calle 2-21 hacia la 14 calle 6-18. Prueba 1, 2 colocan automáticamente los valores para las direcciones de la prueba 1 y 2 respectivamente. Estas direcciones se escogieron para que los usuarios pudieran hacer una búsqueda cercana (Búsqueda A) y una lejana (Búsqueda B), y evaluar el tiempo para generar la ruta dependiendo la distancia.

La siguiente imagen sirve únicamente como ilustración del ejemplo anterior, para ver de manera gráfica las direcciones que conforman la ruta, al representarlas con puntos. Si se siguen los puntos se llega del origen al destino.

Ilustración 4 Mapa zona 10. Ejemplo cálculo de ruta.



VIII. RESULTADOS DE LA EVALUACIÓN

A. Evaluación técnica

Se calculó el tiempo que se tarda cada algoritmo en encontrar la ruta óptima entre todos los puntos de la base de datos.

Primera prueba: 104 direcciones de la zona 10, por lo que al calcular la dirección de un punto hacia todos los demás se obtuvieron 10816 resultados.

Segunda prueba: 147 direcciones de la zona 10, por lo que al calcular la dirección de un punto hacia todos los demás se obtuvieron 21632 resultados.

Tercer prueba: 208 direcciones de la zona 10, por lo que al calcular la dirección de un punto hacia todos los demás se obtuvieron 43264 resultados.

Para realizar esto se tomaron todas las direcciones guardadas como direcciones origen y a su vez como direcciones destino. El tiempo se almacenó por cada par de direcciones buscadas y se comparó entre los 4 algoritmos.

Especificación del sistema para las pruebas:

- **Dell Precision M6400**
- Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core
- 2.0GB, DDR3-1066MHz SDRAM, 2 DIMMS
- 250GB Hard Drive, 7200RPM

Los datos teóricos se obtuvieron al multiplicar el tiempo que le toma a la base de datos encontrar una ruta por el total de rutas de cada prueba.

Tabla 1 Datos Teóricos

Datos teóricos	10816 rutas	21632 rutas	43264 rutas
Algoritmo de Dijkstra	35.59083min	71.18122 min	142.36332 min
Algoritmo de Bellman-Ford	35.86293 min	71.72586 min	143.45172 min
Algoritmo de Floyd-Warshall	35.81988 min	71.63976 min	143.27952 min
Averiguate (Dijkstra genético)	37.00533 min	111.0159 min	296.04264 min

Tabla 2 Datos obtenidos en la prueba

Datos obtenidos en la prueba	10816 rutas	21632 rutas	43264 rutas
Algoritmo de Dijkstra	35.59083 min	72.18152 min	165.76327 min
Algoritmo de Bellman-Ford	35.86293 min	73.71586 min	168.36351 min
Algoritmo de Floyd-Warshall	35.81988 min	73.83983 min	166.23381 min
Averiguate (Dijkstra genético)	37.00533 min	118.36871 min	323.44272 min

Tabla 3 Porcentaje de mejora sobre Averiguate

Porcentaje de mejora sobre Averiguate	10816 rutas	21632 rutas	43264 rutas
Algoritmo de Dijkstra	3.82242%	39.01976%	48.75034%
Algoritmo de Bellman-Ford	3.08712%	37.72352%	47.94641%
Algoritmo de Floyd-Warshall	3.20345%	37.61879%	48.60486%
Averiguate (Dijkstra genético)			

Después de analizar los resultados se pudo determinar que el algoritmo de Dijkstra implementado en la base de datos, generó las rutas más rápido que los otros algoritmos. La mejora obtenida con base al algoritmo de Averiguate es de:

1.414 minutos 3.8224219156% más rápido con 10816 rutas.

77.68 minutos 46.86167810% más rápido con 43264 rutas.

Se puede ver que al incrementar la cantidad de los datos el tiempo para encontrar todas las rutas aumenta con los cuatro algoritmos. Sin embargo, el incremento del tiempo es menor para los algoritmos implementados en la base de datos que para el algoritmo de Averiguate. Debido a la complejidad de los algoritmos, funciones de O() de cada algoritmo, se pudo observar que la variación del tiempo al incrementar el número de rutas (n) se mantiene entre los algoritmos que tienen complejidad cuadrática.

Para la prueba de 43264 rutas se obtuvo una diferencia de 22.64 minutos con el tiempo teórico esperado (142.3633 minutos) para calcular las rutas con el algoritmo de Dijkstra. Esta diferencia de la teoría con la práctica puede explicarse analizando el hardware en el que se están realizando las pruebas de la siguiente manera: Se realizaron pruebas cambiando la memoria RAM de la máquina, la velocidad del procesador y utilizando una máquina con un disco duro más rápido, con el fin de ver cuál de estos factores es el que más altera los resultados.

Cantidad de datos: 10816 rutas

Ilustración 5 Mejoras en hardware para 10816 rutas

Especificación del sistema	Algoritmo de Dijkstra	Algoritmo Bellman-Ford	Algoritmo Floyd-Warshall	Averiguate
Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	35.59083 minutos	35.86293 minutos	35.81988 minutos	37.00533 minutos
MÁS PROCESADOR **Intel® Core™ 2 Duo P8700 (3.06 GHz 3M L2 Cache, 1066MHz) Dual Core** **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	35.48365 minutos	35.36524 minutos	35.63254 minutos	36.9036 minutos
MÁS MEMORIA RAM **Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core**	35.23636 minutos	35.47089 minutos	35.36542 minutos	36.86332 minutos

Continuación Ilustración 6

3.0GB, DDR3-1066MHz SDRAM **250GB Hard Drive, 7200RPM**				
MÁS REVOLUCIONES EN DISCO DURO **Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core** **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 10000RPM**	35.03566 minutos	35.47311 minutos	35.20263 minutos	36.82502 minutos

Cantidad de datos: 21632 rutas

Ilustración 7 Mejoras en hardware para 21632 rutas

Especificación del sistema	Algoritmo de Dijkstra	Algoritmo Bellman-Ford	Algoritmo Floyd-Warshall	Averiguate
Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	72.18152 minutos	73.71586 minutos	73.83983 minutos	118.36871 minutos
MÁS PROCESADOR **Intel® Core™ 2 Duo P8700 (3.06 GHz 3M L2 Cache, 1066MHz) Dual Core** **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	70.29631 minutos	71.7511 minutos	72.3083 minutos	117.0355 minutos
MÁS MEMORIA RAM **Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core** **3.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	70.3349 minutos	72.4789 minutos	70.5356 minutos	116.3679 minutos
MÁS REVOLUCIONES EN DISCO DURO **Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core** **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 10000RPM**	69.1311 minutos	70.7891 minutos	70.0213 minutos	116.0026 minutos

Cantidad de datos: 43264 rutas

Ilustración 8 Mejoras en hardware para 43264 rutas

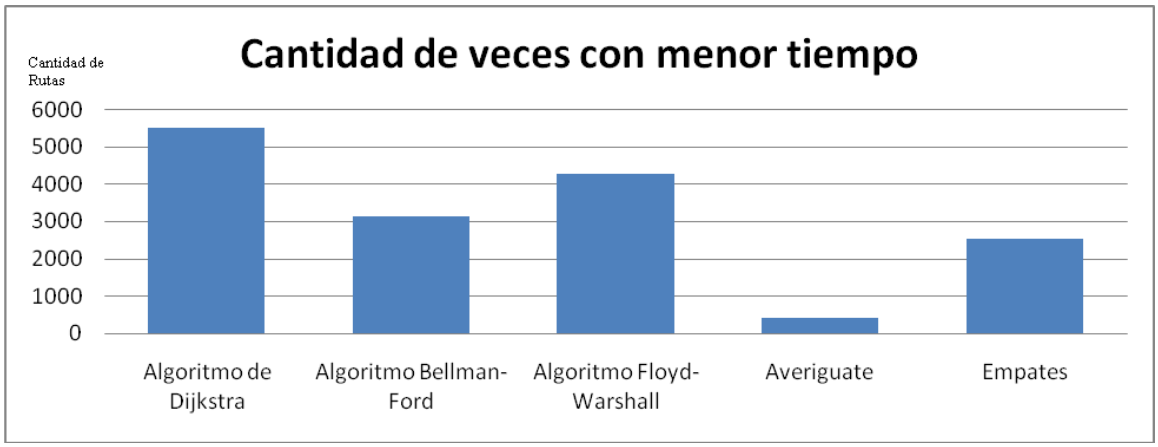
Especificación del sistema	Algoritmo de Dijkstra	Algoritmo Bellman-Ford	Algoritmo Floyd-Warshall	Averiguate
Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	165.76327 minutos	168.36351 minutos	166.2338 minutos	323.44272 minutos
MÁS PROCESADOR **Intel® Core™ 2 Duo P8700 (3.06 GHz 3M L2 Cache, 1066MHz) Dual Core** **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	163.29631 minutos	166.7511 minutos	164.3083 minutos	320.0355 minutos
MÁS MEMORIA RAM **Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core** **3.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 7200RPM**	161.3349 minutos	164.4789 minutos	162.5356 minutos	320.3679 minutos
MÁS REVOLUCIONES EN DISCO DURO **Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core** **2.0GB, DDR3-1066MHz SDRAM** **250GB Hard Drive, 10000RPM**	156.1311 minutos	157.7891 minutos	159.0213 minutos	319.0338 minutos

Después de realizar las pruebas se pudo determinar que entre más datos se tienen en la base de datos, se necesitan más lecturas al disco y más memoria RAM para almacenar las rutas mientras se calculan. Revisando el algoritmo de Dijkstra (Ilustración 6) al trabajar con 10816 rutas, aumentar la velocidad del disco duro implica una mejora de 6.1692%, al aumentar la memoria RAM se obtuvo una mejora de 2.7448% y al aumentar la velocidad del procesador se obtuvo una mejora de 1.5107%. Mientras que al tener 43264 rutas (Ilustración 8) aumentar la velocidad del disco duro implica una mejora

de 5.81%, al aumentar la memoria RAM se obtuvo una mejora de 2.6742% y al aumentar la velocidad del procesador se obtuvo una mejora de 1.4882%.

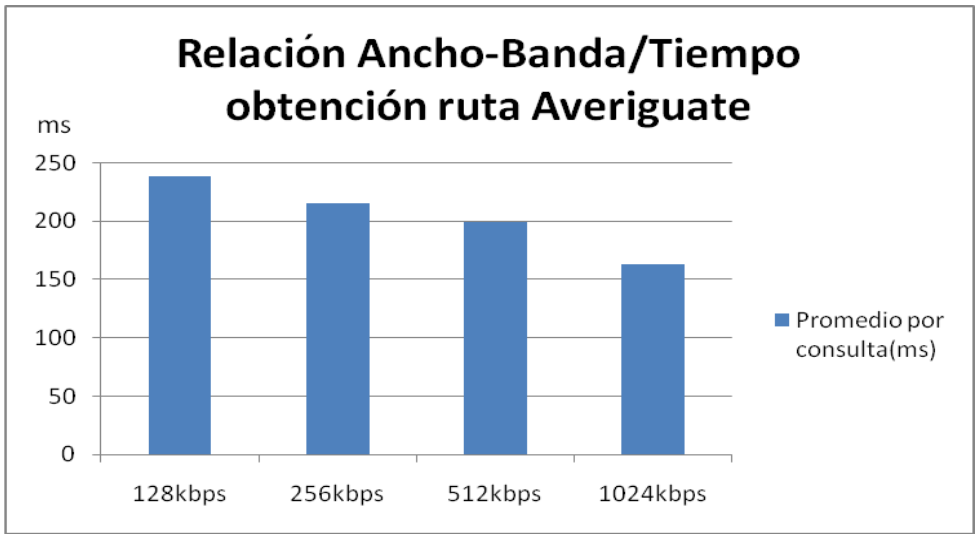
En la siguiente gráfica se puede ver la cantidad de veces que cada algoritmo fue más rápido para encontrar cierta ruta. Se puede observar que el algoritmo de Dijkstra fue el que más veces obtuvo el resultado más rápidamente seguido por el algoritmo Floyd-Warshall.

Ilustración 9 Tiempo menor para generar ruta



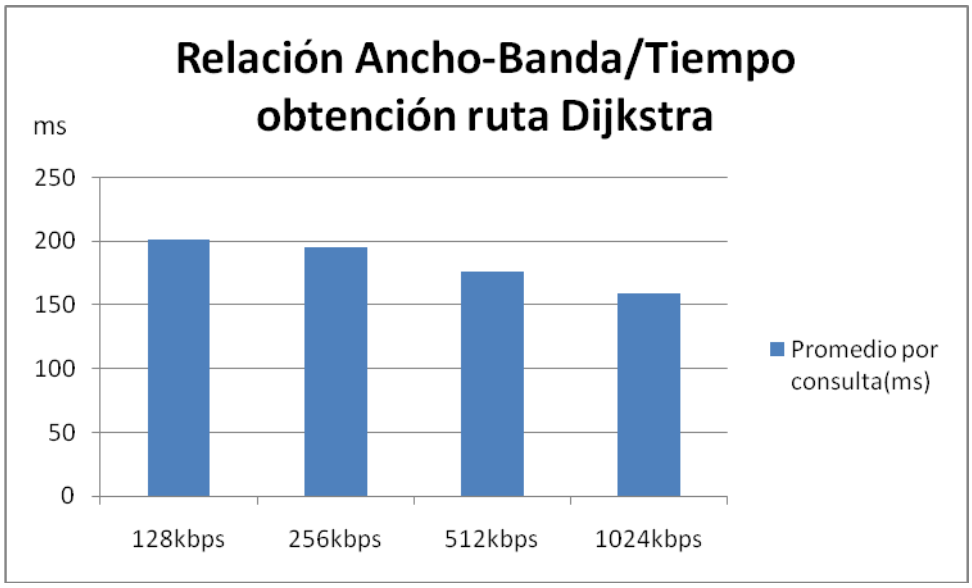
En la siguiente gráfica se puede ver el tiempo requerido para encontrar una ruta con el algoritmo de Averiguate. Esta imagen sirve para ver que Averiguate sí depende del ancho de banda, a menor ancho de banda, mayor tiempo de respuesta.

Ilustración 10 Relación Ancho-Banda/Tiempo obtención ruta Averiguate



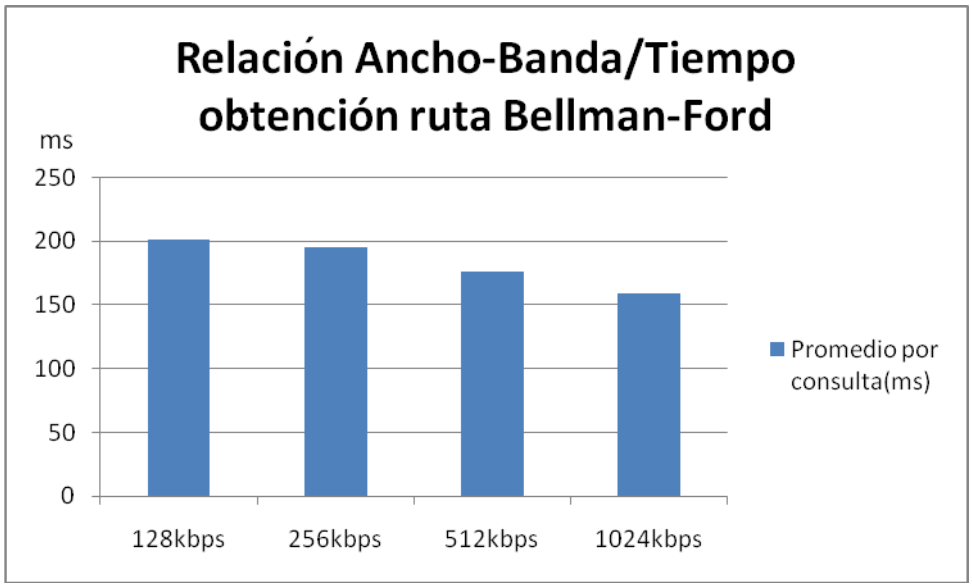
En la siguiente gráfica se puede ver el tiempo requerido para encontrar una ruta con el algoritmo de Dijkstra.

Ilustración 11 Relación Ancho-Banda/Tiempo obtención ruta Dijkstra



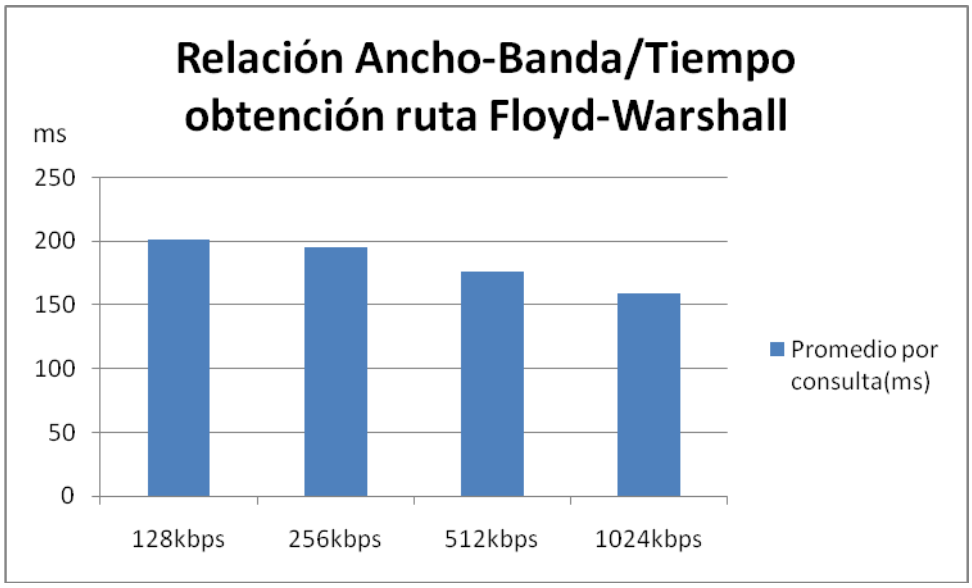
En la siguiente gráfica se puede ver el tiempo requerido para encontrar una ruta con el algoritmo de Bellman-Ford.

Ilustración 12 Relación Ancho-Banda/Tiempo obtención ruta Bellman-Ford



En la siguiente gráfica se puede ver el tiempo requerido para encontrar una ruta con el algoritmo de Floyd-Warshall.

Ilustración 13 Relación Ancho-Banda/Tiempo obtención ruta Floyd-Warshall



IX. DISCUSIÓN

La evaluación técnica para los algoritmos implementados se realizó a través de una aplicación de escritorio que se conecta, a través de internet, a la base de datos que se encuentra en el servidor web y consulta las rutas entre todos los puntos. Se realizó esta aplicación separada de la base de datos ya que el algoritmo de Averiguate se encuentra en su propia aplicación web. Ambas aplicaciones se instalaron en una computadora con conexión a Internet (512kbps) para realizar la generación de las rutas. Esto con el fin de obtener resultados más exactos ya que ambas aplicaciones toman en cuenta el tiempo de demora del Internet, al obtener los datos del servidor web.

Utilizando la aplicación de la evaluación técnica, para encontrar las rutas entre todas las direcciones ingresadas de la zona 10, se pudo determinar que la implementación del algoritmo de Dijkstra en la base de datos disminuyó el tiempo total de 37.0053333 minutos (Averiguate) a 35.59083333 minutos (algoritmo de Dijkstra), por lo que el algoritmo de Dijkstra es 3.8224219156% más rápido que el de Averiguate. Esto se debe a que la aplicación de la evaluación técnica sólo realiza una llamada a la base de datos para encontrar los resultados de la ruta y estos son devueltos a la aplicación. Mientras que la aplicación de Averiguate hace múltiples llamadas a la base de datos para poder generar cada ruta, ya que depende de la ejecución del código de la aplicación para decidir que nodo es el siguiente en la ruta. Principalmente, se economiza el ancho de banda que se utiliza por las consultas realizadas, ya que no es necesario que se hagan distintas llamadas al servidor. En general los tres algoritmos implementados en la base de datos (algoritmo de Dijkstra, algoritmo de Bellman-Ford y el algoritmo de Floyd-Warshall), mejoraron la eficiencia (Ilustración 5) para encontrar una ruta entre dos puntos, requiriendo un tiempo menor que el del algoritmo implementado en una aplicación Web.

Las estructuras de programación utilizadas en el algoritmo de Averiguate afectan el tiempo para encontrar una ruta, ya que para programar este algoritmo en una aplicación web, se utilizaron pilas y colas que contienen los objetos de nodos, caminos y rutas. Las pilas se utilizaron para agregar direcciones a la ruta y luego retroceder cuando se

determina que el camino que se está siguiendo es incorrecto. Las colas se utilizaron para almacenar el recorrido final de la ruta más corta. Cada una de estas estructuras y objetos ocupa espacio en la memoria del programa, por lo cual entre mayor sea la cantidad de direcciones entre el origen y el destino, más objetos deben instanciarse por lo que se ocupa más memoria RAM y más recursos del procesador, lo que vuelve más lenta a la aplicación. Esto puede observarse en la Ilustración 8 al obtener una mejora notable en el rendimiento al incrementar la memoria RAM o el procesador.

Al mantener la lógica de los algoritmos y la información de las direcciones en la base de datos, la utilización de estas estructuras de programación se evitó, ya que todas las direcciones se almacenan como registros en una tabla y pueden ser recorridas con el lenguaje SQL. Los algoritmos fueron implementados como procedimientos almacenados, con lo cual se evita que el motor de la base de datos valide si la sintaxis de la consulta de SQL es correcta cada vez que se ejecuta (el tiempo de validación varía dependiendo de la complejidad de la consulta). Adicionalmente los datos que se obtienen en cada consulta son únicamente los que se necesitan para continuar con la ruta que se está buscando, con lo que se evita sobrecargar la memoria con toda la información en la base de datos. Se utilizó la menor cantidad de ciclos para mejorar el rendimiento de los algoritmos. En lugar de recorrer recursivamente los resultados se optó por utilizar tablas temporales que se crean en tiempo de ejecución.

Para obtener las direcciones a las cuales se pueden llegar desde un punto sólo es necesario consultar si una relación existe en la tabla Path y determinar el camino más corto. La ventaja de utilizar el lenguaje SQL y la base de datos es el alto rendimiento que ofrecen para trabajar con conjuntos de datos.

Con 10816 rutas, el promedio para encontrar la ruta entre dos direcciones con el algoritmo de Dijkstra fue de 199 milisegundos (2,135,450 milisegundos/10816 rutas) mientras que con el algoritmo de Averiguate fue de 206 milisegundos(2,220,463 milisegundos/10816 rutas), 4% más que el de Dijkstra.

Con 21632 rutas, el promedio para encontrar la ruta entre dos direcciones con el algoritmo de Dijkstra fue de 201 milisegundos (4,330,891 milisegundos/21632 rutas)

mientras que con el algoritmo de Averiguate fue de 328 milisegundos(7,102,122 milisegundos/21632 rutas), 38% más que el de Dijkstra.

Con 43264 rutas, el promedio para encontrar la ruta entre dos direcciones con el algoritmo de Dijkstra fue de 229 milisegundos (9945796 milisegundos/43264 rutas) mientras que con el algoritmo de Averiguate fue de 449 milisegundos(19406563 milisegundos/43264 rutas), 49% más que el de Dijkstra.

En los resultados se puede observar que el algoritmo de Dijkstra no fue el más rápido el 100% de las veces, esto se debe a que los tres algoritmos tienen una complejidad polinomial. Esta complejidad indica el tiempo mediante una aproximación al número de pasos de ejecución que un algoritmo emplea para resolver un problema, y ya que la complejidad es polinomial nos indica que los algoritmos pueden ser resueltos. Al poseer los tres algoritmos una cantidad de pasos similares permite a cualquiera de los tres obtener el menor tiempo.

Observando los resultados de la evaluación se obtuvo una diferencia entre los tiempos teóricos y los obtenidos en las pruebas, esto es debido a que la teoría no toma en cuenta el hardware que se tiene para realizar las pruebas. Por lo que entre más direcciones se encuentren en la base de datos y más rutas se deseen generar, se necesita contar con mejores recursos, por ejemplo más velocidad de procesador y más memoria Ram.

La ventaja que tiene Dijkstra y Floyd-Warshall sobre el tercer algoritmo implementado es la información que se encuentra en la base de datos, ya que al calcular rutas no se tienen pesos negativos, que es un aspecto que el algoritmo de Bellman Ford toma en cuenta y conlleva a más validaciones. Al calcular la ruta más corta entre dos direcciones los pesos negativos se omiten, ya que en la práctica no se puede llegar de una dirección a otra con un tiempo negativo. Si se sale del punto A en tiempo T se debe llegar al destino en un tiempo T2 que sea mayor o igual a T. Al momento de ingresar pesos negativos con los algoritmos de Dijkstra y Floyd-Warshall no se va a encontrar nunca una ruta más corta si en el camino se encuentra un vértice con peso negativo, ya que estos algoritmos se quedarían en un ciclo infinito pasando sobre este vértice para seguir

bajando el costo total de la ruta, en cambio Bellman-Ford valida que el peso de la ruta no sea negativo y que solo se recorra una vez cada vértice, evitando los ciclos infinitos.

Al terminar la evaluación técnica con 10816 rutas en la aplicación de escritorio se observó que el algoritmo de Dijkstra fue el más rápido 5526(41.35%) veces, para generar las rutas. Luego el Algoritmo Floyd-Warshall con 4274(31.99%) veces, el Algoritmo Bellman-Ford con 3142(23.54%) veces y el algoritmo de Averiguate con 417(3.12%) veces (Ilustración 9). Con esto se verifica que, a pesar de los empates entre Dijkstra y los otros algoritmos, el algoritmo de Dijkstra obtuvo la mayor cantidad de rutas en el menor tiempo y la mayor cantidad de veces (5526).

Al momento de realizar la encuesta se encontraban 10816 rutas en la base de datos y se realizó con el servidor con el sistema operativo Windows Server 2003 y la base de datos MS SQL Server 2005. El procesador es Intel® Core™ 2 Duo P8700 (2.53GHz 3M L2 Cache, 1066MHz) Dual Core y 4GB de memoria RAM. La encuesta realizada es únicamente una apreciación subjetiva de los posibles usuarios y no influye directamente en el análisis de los algoritmos. Las preguntas que se hicieron en la encuesta ayudan a evaluar factores externos que pueden afectar el tiempo en generar las rutas. En este caso pudo observarse que a mayor ancho de banda, el tiempo en obtener la ruta no impacta tanto ya que el tiempo en el que se carga la página compensa el tiempo de la base de datos.

La ventaja que se puede obtener de los algoritmos implementados en la base de datos es que no dependen del ancho de banda de la conexión a Internet de los usuarios para mejorar su eficiencia al obtener los resultados. Esto es debido a que la base de datos únicamente recibe una solicitud para encontrar la ruta, la genera y devuelve el resultado, al contrario de Averiguate que realiza múltiples solicitudes para poder obtener los datos y encontrar la ruta.

X. CONCLUSIONES

El algoritmo de Dijkstra implementado en la base de datos utilizó un menor tiempo para generar las rutas, con una mejora de 3.8224219156% sobre el algoritmo de Averiguate con 10816 rutas y una mejora de 48.75034% con 43264 rutas, con lo que se logra que los usuarios esperen menos al realizar las búsquedas de su ruta y puedan generarlas en el momento que las necesitan.

Con 10816 rutas el algoritmo de Dijkstra implementado en la base de datos fue el más rápido 5526(41.35%) veces mientras que el algoritmo de Averiguate solo lo fue 417(3.12%) veces, lo que asegura la consistencia en la eficiencia para realizar búsquedas de rutas entre dos direcciones de la zona 10 de la Ciudad de Guatemala.

Con 21632 y 43264 rutas el algoritmo de Dijkstra implementado en la base de datos también fue más rápido para encontrar las rutas. Al incrementar los datos el algoritmo de Averiguate también incrementaba el tiempo de búsqueda, haciendo que las búsquedas fueran más lentas. Esto corresponde a la función de complejidad que ambos algoritmos presentaban en la cual el algoritmo de Averiguate tiene una complejidad mayor $O(n^3)$ que la del algoritmo de Dijkstra $O(n^2)$.

Para mejorar los tiempos obtenidos en la práctica se observó que aumentar la velocidad del disco duro implica una mejora de 5.81%, al aumentar la memoria RAM se obtuvo una mejora de 2.6742% y al aumentar la velocidad del procesador se obtuvo una mejora de 1.4882% (Ilustración 8).

XI. RECOMENDACIONES

- Crear un Web Service, que se encuentre en línea y que permita obtener las rutas en base a las direcciones origen y destino que cualquier aplicación envíe como parámetros.

- Realizar pruebas simultáneas entre varios usuarios que accedan a la aplicación y cuenten con recursos homogéneos (ancho de banda, procesador, memoria RAM).

- Realizar una interfaz gráfica para mostrar a los usuarios en un mapa la ruta más corta generada, para facilitar el seguimiento de la misma.

- Mostar publicidad georeferenciada en base a la cercanía de las búsquedas de los usuarios.

- Crear categorías de destinos para poder realizar búsquedas hacia restaurantes, cines, universidades y demás lugares de interés, y así evitar el uso de la dirección exacta.

- Implementar la base de datos y la aplicación en el mismo servidor web para evitar llamadas entre dos servidores diferentes, evitando el factor de ancho de banda en la aplicación de Averiguate.

XII. GLOSARIO

- Algoritmo: lista bien definida, ordenada y finita de operaciones que permite hallar la solución a un problema. Dado un estado inicial y una entrada, a través de pasos sucesivos y bien definidos se llega a un estado final, obteniendo una solución. [X]
- Ancho de Banda: cantidad de datos que se pueden transmitir en una unidad de tiempo. [Q]
- Aplicación Web: aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. [Q]
- Base de datos: Conjunto de datos almacenados en un soporte informático no volátil. Los datos están interrelacionados y estructurados de acuerdo con un modelo de datos. [I]
- Grafo: conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto. [S]
- Interfaz: medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo, normalmente suelen ser fáciles de entender y fáciles de accionar. [X]
- Internet: conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial. [Q]
- Kbps (Kilobits por segundo): unidad de medida que se usa en telecomunicaciones e informática para calcular la velocidad de transferencia de información a través de una red. [M]
- Memoria volátil: tipo de memoria que requiere energía constante para mantener la información almacenada. La memoria volátil se suele usar sólo en memorias primarias. La memoria RAM es una memoria volátil, ya que pierde información en la falta de energía eléctrica. [Q]

- Memoria no volátil: tipo de memoria que retendrá la información almacenada incluso si no recibe corriente eléctrica constantemente. Se usa para almacenamientos a largo plazo y, por tanto, se usa en memorias secundarias, terciarias y fuera de línea. [Q]
- Navegador de Internet: programa que permite visualizar la información que contiene una página web, interpreta el código, HTML generalmente, en el que está escrita la página web y lo presenta en pantalla permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces o hipervínculos. [W]
- Nodo: elementos de una lista enlazada, de un árbol o de un grafo. Cada nodo es una estructura o registro con varios campos a partir del cual debe poder accederse a otros nodos de la estructura. [L]
- Pseudocódigo: serie de palabras léxicas y gramaticales referidos a los lenguajes de programación, pero sin llegar a la rigidez de la sintaxis de estos ni a la fluidez del lenguaje coloquial. [Q]
- Ruta: sucesión de direcciones que definen un camino. [Q]
- Servidor Web: programa que se ejecuta continuamente en un ordenador (también se emplea el término para referirse al ordenador que lo ejecuta), manteniéndose a la espera de peticiones por parte de un cliente (un navegador web) y que responde a estas peticiones adecuadamente, mediante una página web que se exhibirá en el navegador o mostrando el respectivo mensaje si se detectó algún error. [W]
- Vértice: punto común que une dos nodos. [L]

XIII. BIBLIOGRAFÍA

A. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. Data Structures and Algorithms Addison-Wesley, Massachusetts, 1983. http://downloads2.esri.com/support/whitepapers/other/arcgis-engine_library _diagram 0507.pdf

B. A Relational Model of Data for Large Shared Data Banks Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387

C. Ahuja, R.K. Magnanti. TL. Orlin, J.B. 1993 Network Flows: theory, algorithms and applications. Prentice Hall. 864 págs.

D. Arc GIS Engine 9.2 . 2009. Library reference diagram.

E. Barabasi, Albert-Laszlo. Linked: The New Science of Networks. Cambridge, MA: Perseus Publishing, 2002.

F. Calero Vinelo, Arístides. Técnicas de Muestreo / Arístides Calero Vinelo.- La Habana: Editorial. Pueblo y Eduacación, 1978. 514p.

G. Codd, E.F. June 1970. A Relational Model of Data for Large Shared Databanks. Communications of the ACM. 377-387

H. Date C.J. 1994. An Introduction to Database Systems. Addison-Wesley. 839 págs.

I. Dimitri P. Bertsekas March 1992. A Simple and Fast Label Correcting Algorithm for Shortest Paths. Networks, Vol. 23, pp. 703-709

J. Encuesta en línea. http://www.surveymonkey.com/

K. Floyd, Robert W. June 1962. Algorithm 97: Shortest Path. Communications of the ACM 5 345

L. G. Chartrand, P. Zhang: "Introduction to Graph Theory". McGraw-Hill, 2005

M. G. Chartrand, O. R. Oellermann: Applied and Algorithmic Graph Theory. McGraw-Hill, 1993

N. Gómez. M. A. 2001. Genetic Algorithms and Simulation Applied to Optimization. The Stochastic Shortest Path model, Disertación. New Mexico State University. Las Cruces.

O. Google Maps API. 2009. http://code.google.com/apis/maps/

P. Improving ASP.NET Performance http://msdn.microsoft.com/en-us/library/ms998549.aspx

Q. J. Aldous, A. Dolan: Networks. Wiley, 1993.

R. Kenneth H. Rosen. 2003. Discrete Mathematics and Its Applications, 5ª Edición. Addison Wesley

S. Kocay, D. Kreher: "Graphs, Algorithms and Optimization". Chapman & Hall/CRC, 2005

T. L. Goldschlager and A. Lister. Computer Science, A Modern Introduction Series in Computer Science. Prentice-Hall Intl., London (UK), 1982.

U. Megaproyecto *Sistema de posicionamiento y trazado de rutas en Guatemala* Universidad del Valle de Guatemala. 2008

V. R. Skvarcius and W. B. Robinson. Discrete Mathematics with Computer Science Applications Benjamin/Cummings, Menlo Park, California, 1986

W.Ronald M. Baecker & William A. S. Buxton (Editors). Readings in Human-Computer Interaction: A Multidisciplinary Approach. Los Altos, CA: Morgan-Kaufmann Publishers, 1987

X. Thomas H. Cormen, Ronald L. Rivest. Introduction to Algorithms, The MIT Press, Segunda Edición, Septiembre 1, 2001

XIV. Anexos

Encuesta algoritmo para generar rutas:

El propósito de esta encuesta es evaluar tres algoritmos para generar rutas que se implementaron como trabajo de graduación. El objetivo es determinar el algoritmo que obtenga la ruta de una manera más rápida. Se cuenta con información de las calles y avenidas de la zona 10 de Guatemala por lo que con esta aplicación se pueden generar rutas de las direcciones de esta zona.

Las aplicaciones se encuentran en:

www.lastutorias.net/Averiguate
www.lastutorias.net/EncontrarRuta

Nota: Las preguntas marcadas con * son obligatorias.
1. ¿Qué proveedor de Internet está utilizando en este momento?
   - Turbonett de Telgua
   - Convergence
   - Telefónica
   - Tigo
   - Claro
   - Otro (especifique)

2. ¿De qué velocidad es su conexión a Internet?
   - 64 kbps
   - 128 kbps
   - 256 kbps
   - 512 kbps
   - Más de 512 kbps
   - No lo sé.

3. ¿Qué navegador de Internet está utilizando para probar las aplicaciones web?
   - Firefox
   - Internet Explorer
   - Safari
   - Opera
   - Chrome
   - Otro (especifique)

Búsquedas Sugeridas

Búsqueda A: Pasos para llegar de “2 Calle 2-28” a “3 Calle 4-16”
Búsqueda B: Pasos para llegar de “2 Calle 2-28” a “4 Calle 1-32”

4. ¿Qué aplicación fue más rápida para presentar resultados utilizando la búsqueda A?
   - averiguate.info
   - lastutorias.net/GenerarRuta - Algoritmo #1
   - lastutorias.net/GenerarRuta - Algoritmo #2
   - lastutorias.net/GenerarRuta - Algoritmo #3

5. ¿Qué aplicación fue más rápida para presentar resultados utilizando la búsqueda B?
   - averiguate.info
   - lastutorias.net/GenerarRuta - Algoritmo #1
   - lastutorias.net/GenerarRuta - Algoritmo #2
   - lastutorias.net/GenerarRuta - Algoritmo #3

6. ¿Qué aplicación fue más rápida para presentar resultados utilizando su búsqueda?
   - averiguate.info
   - lastutorias.net/GenerarRuta - Algoritmo #1
   - lastutorias.net/GenerarRuta - Algoritmo #2
   - lastutorias.net/GenerarRuta - Algoritmo #3

7. ¿Cuál fue su búsqueda?

Resultados de la encuesta

Al finalizar la encuesta se obtuvieron los siguientes resultados:

Ilustración 14 Encuesta: Proveedor de Internet

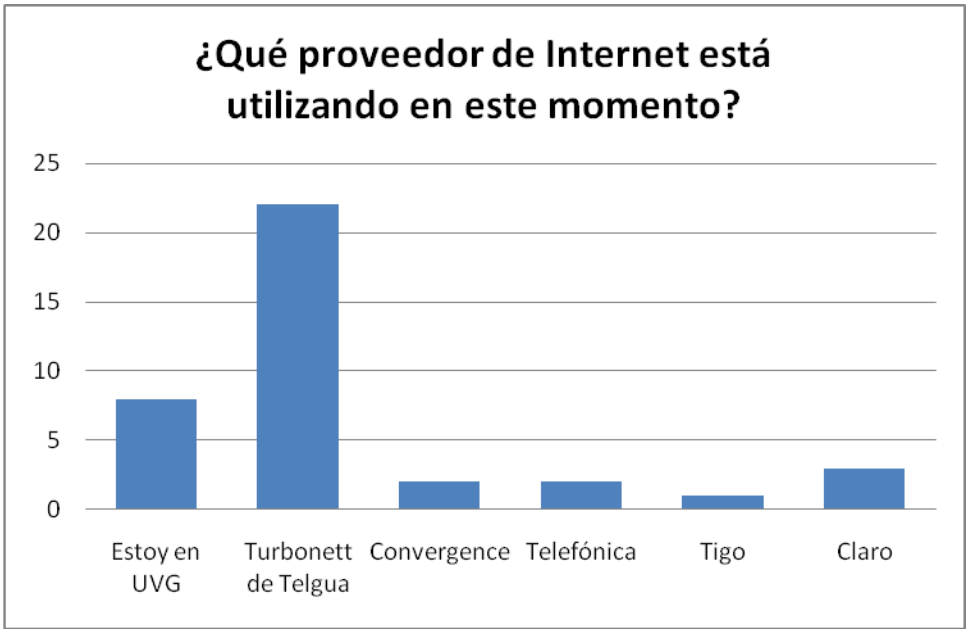


Ilustración 15 Encuesta: Velocidad de conexión

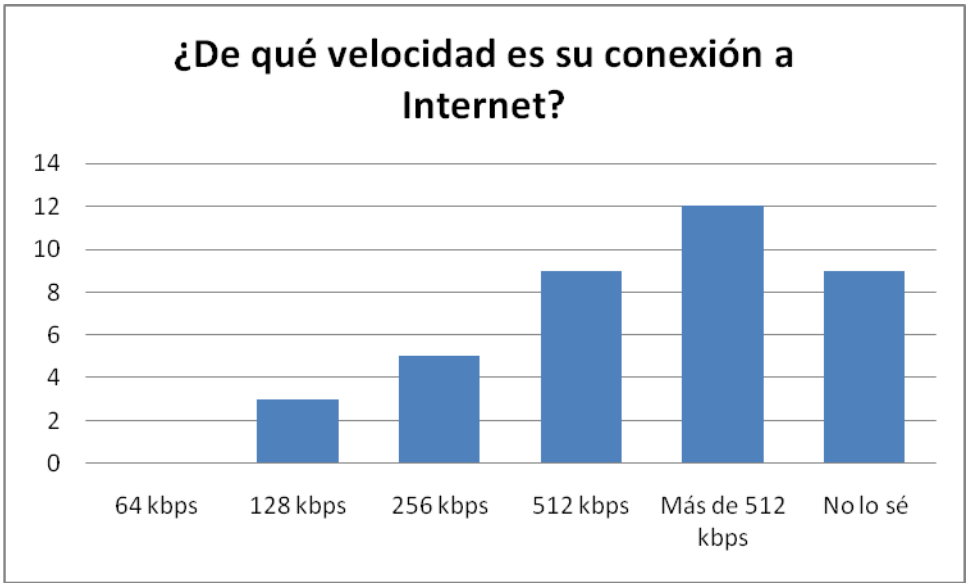


Ilustración 16 Encuesta: Navegador de Internet

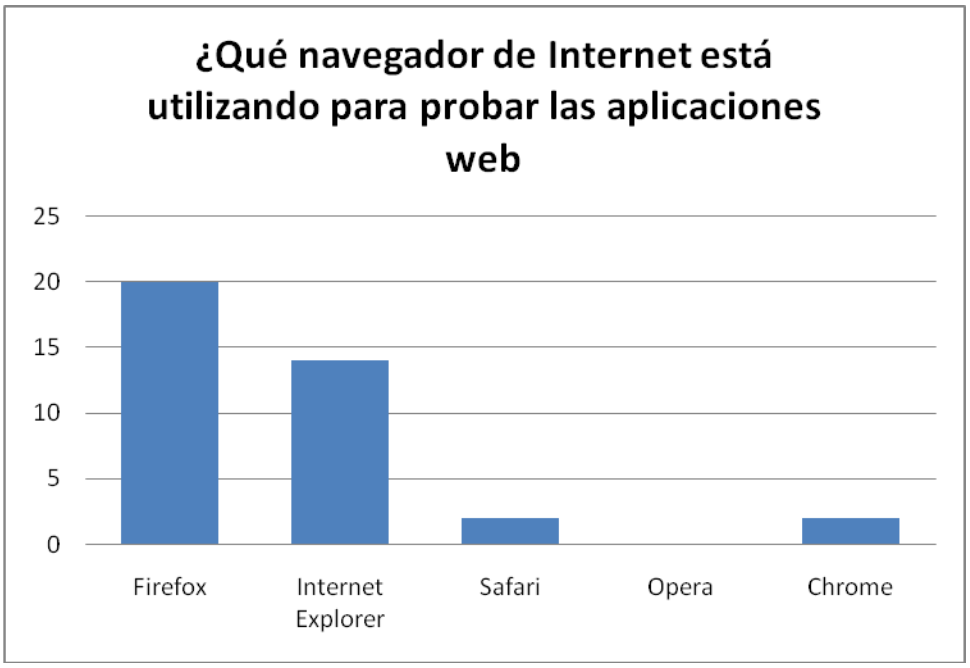


Ilustración 17 Encuesta: Búsqueda A

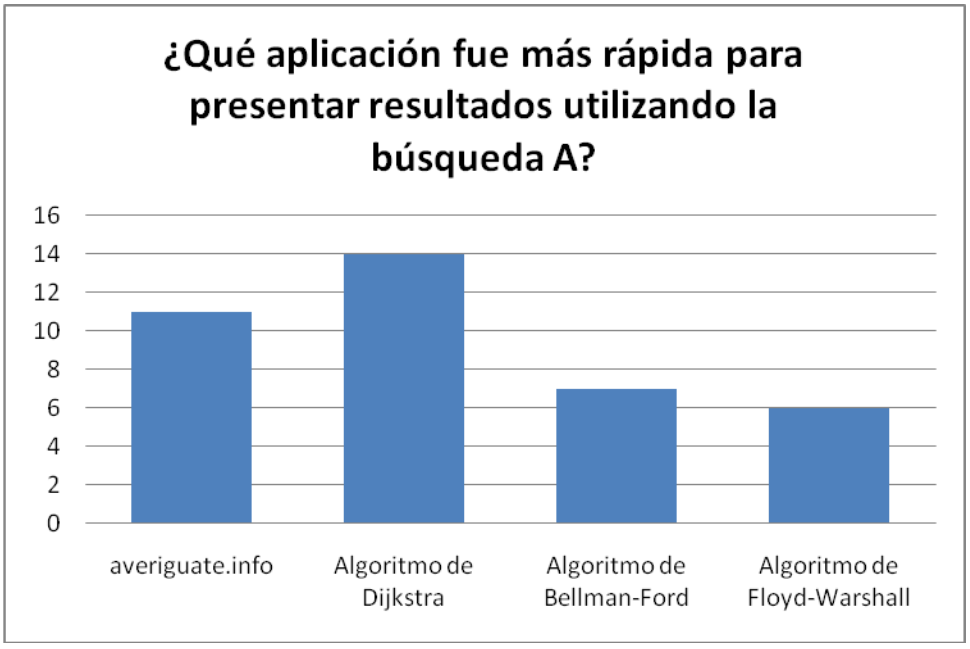


Ilustración 18 Encuesta: Búsqueda B

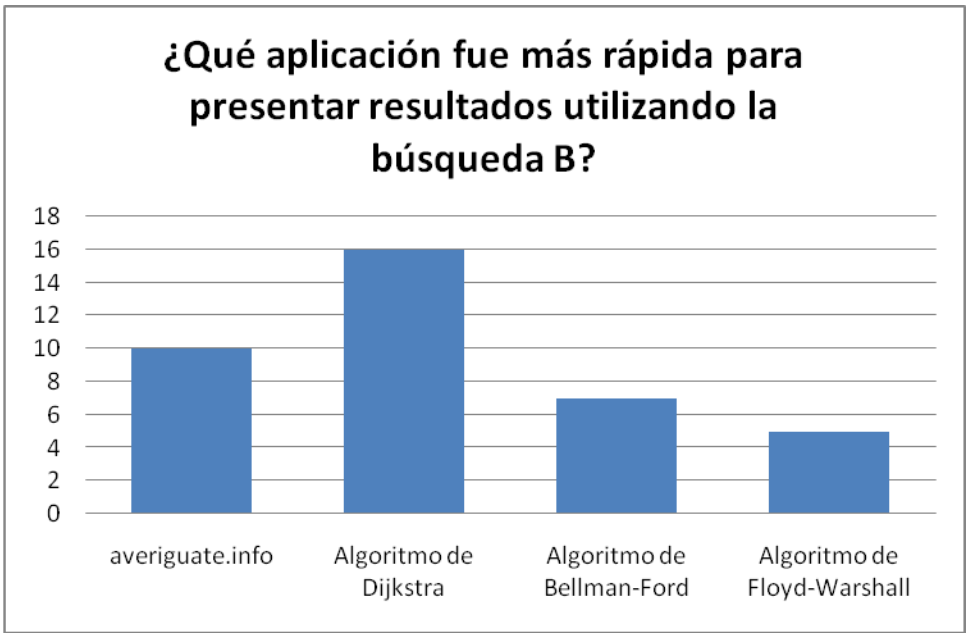


Ilustración 19 Encuesta: Búsqueda personalizada