UNIVERSIDAD DE LOS ANDES

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE SISTEMAS

DEPARTAMENTO DE SISTEMAS COMPUTACIONALES

DISEÑO Y ANÁLISIS DE ALGORTIMOS

BITACORA DE AVANCE DEL PROYECTO: DISTRIBUIDOR ÓPTIMO DE DISTANCIAS EN RUTAS INTERURBANAS

Control+2

González Jesús

Mejías Julio

Obari Anabel

Plaza Miguel

# Presentación

Nos movemos en un mundo que cambia constantemente y más allá de la constancia, puede sorprendernos la manera en que lo hace, cada vez es más rápido, diferente y en algunos entornos podríamos decir que incluso más agresivo.

El cambio en esta nueva era supone nuevas exigencias y cada exigencia es en sí misma un uevo reto. Estos retos, cuando se asumen, pueden concluir en ideas innovadoras que generan gran impacto en la manera en que, no solo vemos al mundo sino como nos movemos en él.

Tal ha sido la innovación que ha desarrollado el ser humano que, entes intangibles asumen grandes roles dentro del desarrollo mismo del ser. Algunos pesimistas podrían asegurar firmemente que la supervivencia se ha visto mermada por la dependencia de los *gadgets* que nos rodean y otros verán oportunidades, irónicamente donde “no las hay”. La ironía del no haber radica en que actualmente es posible ver cosas como que la tienda más grande del mundo (amazon) no tiene establecimientos físicos, la red más grande de amigos no posee un lugar de encuentro (Facebook), uno de los servicios de hospitalidad en auge actualmente no cuenta con ningún hospedaje (airbnb) propio y por último, el tema que nos ocupa, la red más grande de transporte privado no tiene ni un solo vehículo, si, hablamos de Uber.

Uber es un sistema (todo es un sistema) de transporte privado que conecta al cliente con el conductor, organiza recogidas, lleva alimentos y un par de servicios más que han nacido a lo largo del tiempo con la creación de la empresa.

Uno de los servicios interesantes que ofrecen es UberPOOL, que es un servicio que utiliza los asientos disponibles del vehículo para llevar más pasajeros que tengan destinos diferentes en función de la ubicación más cercana. Una de las preguntas frecuentes que incluso salen en la página web es la siguiente:

*Llegaron por mí primero. ¿Por qué llegamos a mi destino después de dejar al otro usuario?*

Y la respuesta que ofrecen es la siguiente:

*El orden en el que se llega por un usuario o el orden de los destinos, depende de su ubicación a lo largo de la ruta, no de quién subió primero al auto.*

Nuestra propuesta radica entonces en la optimización de la ruta para un servicio de uberPOOL. ¿por qué? Sencillo, se nos ocurre esto dado que, si un conductor está en el viaducto sucre y tiene un pasajero en Santa Bárbara, otro en el hotel el Serrano y otro en alguna urbanización más arriba del centro comercial Buganvillas, la lógica del servicio nos dice que el segundo pasajero es el del punto más cercano, sin embargo físicamente es inviable ya que, al salir de Santa Bárbara el conductor debe, por obligación, subir por la Avenida los Próceres hasta encontrar el retorno más cercano y bajar hasta e Hotel el Serrano para luego volver a buscar un retorno y subir hasta la ubanización X.

Es un problema de pathfinding en el que se deben tomar en cuenta además de los trayectos, las direcciones de estos. Al ser éste un sistema extenso para facilitar su estudio y aplicación, ha de delimitarse la frontera del mismo, se limita entonces Casco central de la ciudad de Mérida pudiendo extenderse hasta el Municipio Libertador.

# Primer Avance:

Se requiere un pequeño modelo del grafo incluyendo el tipo de dato en el nodo y en el arco, una explicación breve de la solución a ofrecer y un poco acerca de los algoritmos a utilizar.

Se explicará la propuesta utilizando el método de las 5 W comenzando con ¿qué?

## ¿Qué?

Crear un aplicación capaz de encontrar la manera óptima de realizar viajes para dos o más personas (cuatro personas es la cantidad deseada) dentro de la ciudad de Mérida. Esto es, se desea llevar de dos a cuatro personas buscadas en (posiblemente) puntos distintos a cuatro puntos (posiblemente) distintos, utilizando la menor distancia.

## ¿Cómo?

Creando una aplicación que utiliza un algoritmo de búsqueda óptima y, por ende, asegurar la menor distancia no solo en el primer tramo origen-destino sino en todo el recorrido, tanto para la búsqueda de los pasajeros como en su entrega.

## ¿Cuándo?

El servicio comienza cuando el conductor acepta la solicitud y la cantidad de pasajeros apilada. Se recomienda comenzar con al menos dos que, probablemente vayan a dos sitios distintos (no se nota el cambio si ambos van al mismo lugar, pero no es limitante). Casos como estos abundan en las noches juveniles de la ciudad.

## ¿Quién?

Todas las personas que estén dispuestas a cambiar el modelo de negocio tradicional de transporte privado y que, además, tomen en cuenta las ventajas de hacer más viajes en menos distancia.

## ¿Dónde?

En cualquier ciudad que desee su implementación. La idea es dejar el algoritmo para que pueda ser implementado en cualquier ciudad objetivo.

Para el tipo de grafo se propone (con probables cambios) un grafo dirigido con pesos, cuyo valor viene dado por la distancia entre dos nodos, en metros. Otras consideraciones tomadas son el bloqueo de una de las calles pertenecientes a una ruta, entonces se introduce el atributo de mayor valor posible, esto para alterar el resultado de búsqueda de camino mínimo.

El algoritmo de búsqueda que se probará en principio será el algoritmo de Dijkstra. Nos enfrentamos ante el primer problema de desarrollo y es buscar la manera de obtener una solución global y no varias locales. Se pide una pequeña muestra con el Dijkstra iterativo por destinos locales para el siguiente avance.

# Segundo Avance

Para la entrega del segundo avance se debe ejecutar el algoritmo de Dijkstra, saliendo del punto donde se asume se recogen los paquetes o “misiones” y entregando en un orden aleatorio y recalculando el siguiente destino partiendo del destino anterior. Para ello se establece una parte del mapa, creando el grafo con sus nodos y arcos correspondientes. Haciendo uso de la biblioteca Aleph- para los nodos se propone:

En una cadena de caracteres se almacenará la información concerniente al nodo. Por cuestiones de diseño, inicialmente se almacena un texto que permita identificar qué parte de la ruta identifica ese nodo. Por tanto, se establecerá que un nodo corresponde a una intersección de dos o más vías. Por otro lado, los arcos almacenaran en un número real la distancia entre nodos y la dirección, lo que nos conduce a pensar que, en efecto, estamos trabajando con un grafo dirigido con pesos cuyos nodos guardan el nombre de la intersección y el arco la distancia entre ellos, no habiendo, por lógica física, distancias negativas en esta aplicación. El arco entonces se define como:

El cálculo de distancias se realiza mediante la herramienta Google Maps; el nombre de los nodos se hace usando tanto un mapa de la ciudad proporcionado por la corporación merideña de turismo como también la herramienta Google Maps. El grafo utilizado se implementa mediante listas enlazadas pertenecientes al archivo tpl\_Graph.H, el grafo utilizado se define entonces como:

La primera etapa utilizada para las primeras pruebas comprende una porción de la zona norte del municipio libertador, delimitada por:

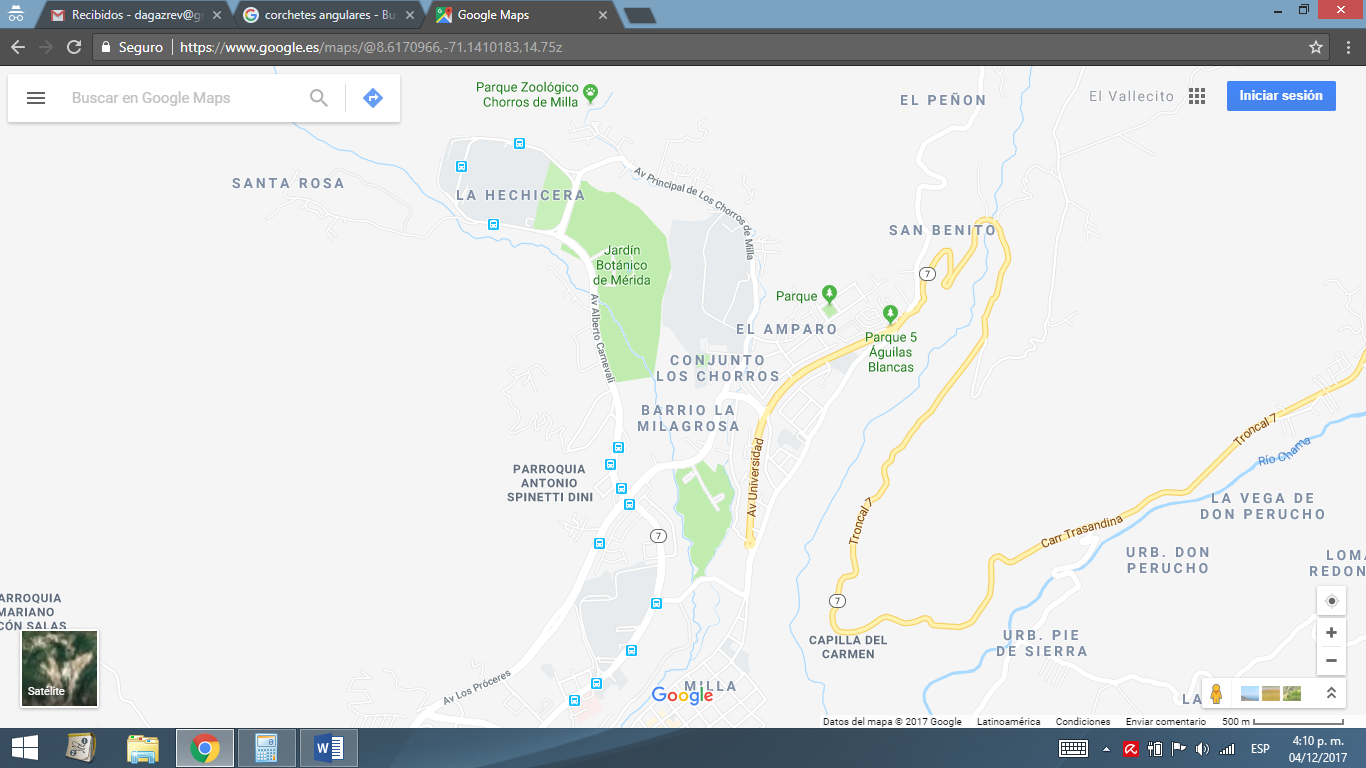


Ilustración Zona delimitada de la ciudad para la primera prueba

La primera parte de la ciudad tiene un total de 205 nodos conectados entre sí mediante sus respectivos arcos. El archivo con los nodos iniciales se encuentra en el archivo prueba1.C. Las rutinas de impresión se basan en un archivo de prueba llamado ex1.C

La ejecución del algoritmo de Dijkstra se encuentra en el archivo prueba\_de\_dijkstra.C, es una porción de código que debe añadirse al archivo prueba1.C y ejecutarse. El archivo final es Proyect.C; Al ejecutarse ésta versión se encuentra el error de que no se puede hallar un árbol abarcador mínimo. Se debe verificar y depurar el código para la siguiente etapa.

# Tercer Avance

La ejecución anterior resultó fallida dado que el algoritmo de dijkstra entregaba un heap vacío. Se procede a depurar el código no sin antes agregar más nodos y arcos agrandando el mapa, la primera depuración se hace sobre el mapa resultante consiguiendo grafos inconexos.

El nuevo mapa resulta:

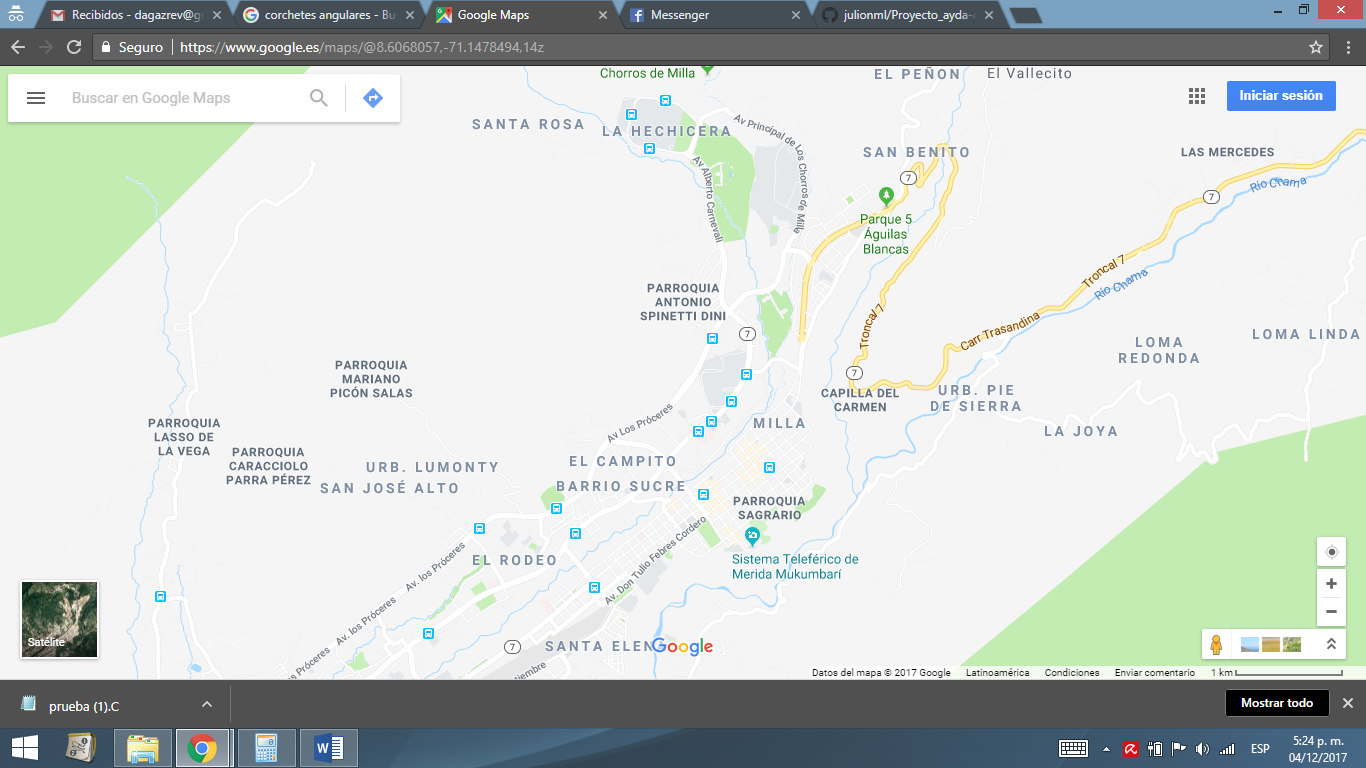


Ilustración Nueva área delimitada por los arcos y nodos agregados

Este mapa incluye el centro de la ciudad de Mérida hasta el viaducto de la calle 26, viaducto Campo Elías.

Dado que no se puede ejecutar el algoritmo de dijkstra que cree un árbol abarcador mínimo y cubrir todos los destinos, se utiliza el algoritmo que busca el camino mínimo entre dos nodos: el nodo *start* que es el principio y el nodo *end*, que es el destino y almacena el resultado en una variable tipo *PATH* que almacena la ruta entre ambos. Tal prueba su instrumenta de la siguiente manera:

Donde:

Graph es el grafo origen

Start es el lugar de donde parte el recorrido

End es el destino

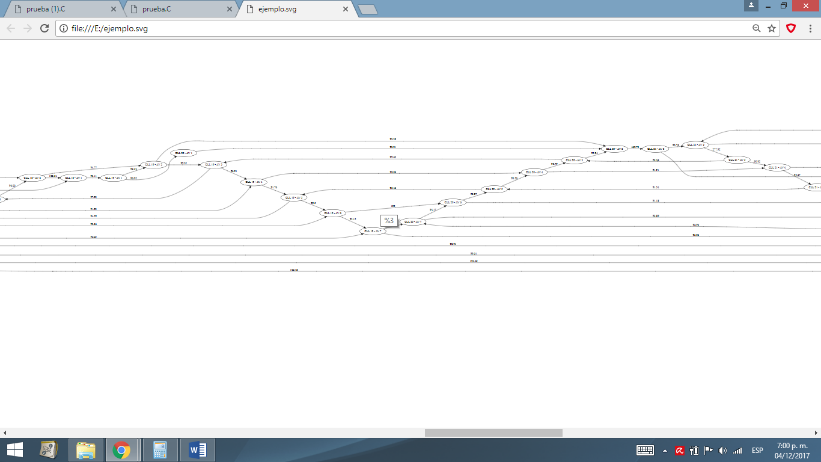
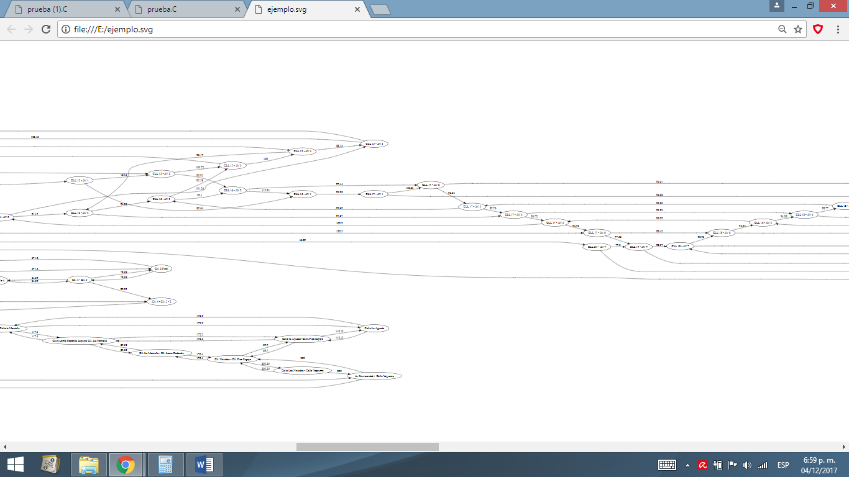
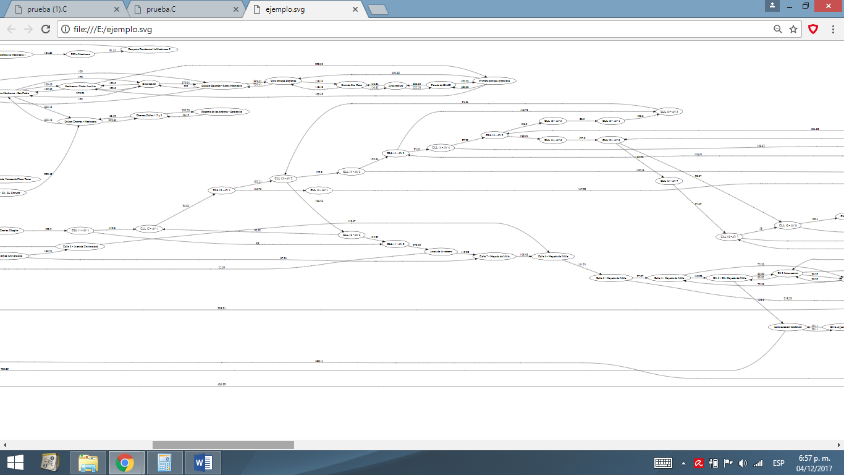
Path es el recorrido almacenado

Al ejecutar esa prueba entre dos de los nodos pertenecientes al grafo, se encuentra un camino mínimo, lo que conduce a pensar que, en efecto, el grafo no es completamente conexo y por ello encontrará caminos de acuerdo a la conectividad, no siendo posible entregar un árbol abarcador mínimo. Dado que se trata de un dígrafo, se realiza la prueba de Tarjan para verificar que el grafo sea fuertemente conexo. La instrumentación e implementación de Tarjan dentro de la biblioteca aleph- se encuentran en el archivo Tarjan.H.

Al ejecutar la prueba de tarjan mediante Tarjan\_Connected\_Components().test\_connectivity(graph) se obtiene como resultado que el grafo no es fuertemente conexo, lo que complicaba la ejecución del algoritmo anterior.

Se ejecuta una prueba visual utilizando la herramienta GraphViz, pensando en la depuración, se considera más fácil hacerlo viendo las conexiones que revisando cada línea de código por lo extenso del archivo; el resultado de GraphViz entrega como resultado un grafo bastante denso e imposible de depurar por lo que se regresa a la depuración manual, resultando tediosa, pero entregando mejores resultados. El resultado completo está en el archivo ejemplo.svg. Se complementa entonces pregunta si el camino generado está vacío con path.is\_empty() que retorna verdadero si el camino está vacío y falso en el caso contrario. Para verificar el camino, se instrumenta un iterador sobre la clase Path similar al iterador sobre grafos visto en el segundo avance. Las modificaciones se encuentran en prueba2.C

Ilustración : Una parte del archivo ejemplo.svg



# Cuarto Avance

Se comienza con extender lo que se tiene del mapa, resultando en el área delimitada por la línea roja en la siguiente figura:

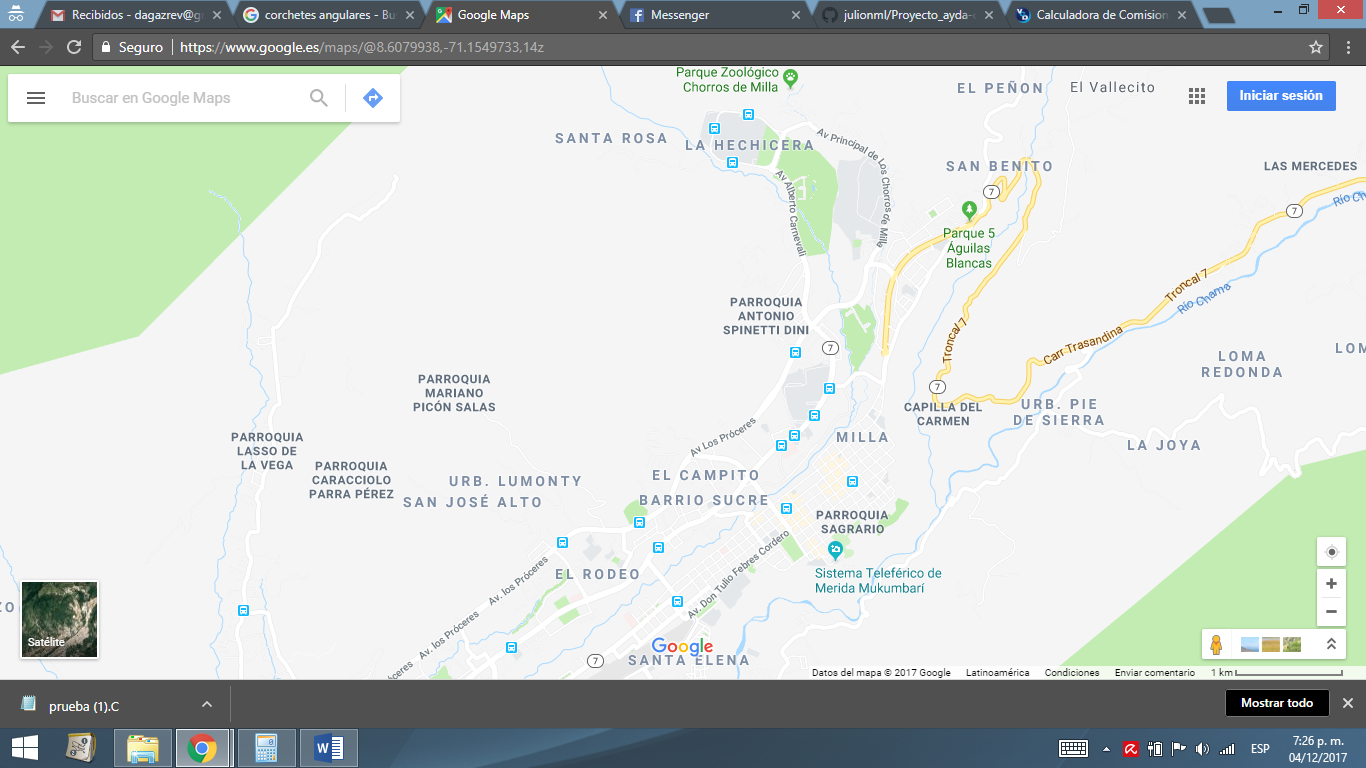


Ilustración área delimitada por la nueva inserción de nodos y arcos

Se ejecutan varias pruebas de camino para depurar la conectividad del grafo, se encuentran varios errores y se corrigen.

Ahora se piensa en la manera de entregar cuatro paquetes en la mejor manera posible. Utilizando combinatoria, cuya fórmula se expresa de la siguiente manera:

Se obtiene que, si hay 4 destinos posibles, combinando de 1 en 1 en los 4 destinos posibles, hay 24 posibles combinaciones en total. El algoritmo debe indicar cuál de las 24 rutas posibles es la menor; tales combinaciones se implementan con un ciclo de repetición anidado que modele todas las posibilidades. Se basa en el incremento que hace cada una de las variables. En primera instancia se almacena por separado el valor de cada iterador y se imprime todo como una secuencia. Cuando se verifica que el código instrumentado genera todas las secuencias sin errores, se almacena cada elemento de la secuencia en una matriz de 24 filas y 4 columnas. La instrumentación de dicha matriz, de momento, es solo una manera de ordenar las combinaciones, no obstante, posteriormente se le dará utilidad al momento de estudiar los caminos posibles y el camino definitivo.

Inicialmente se utiliza cuatro pares del tipo para poder indexar un pequeño conjunto de los nodos destino junto a la posición que deben asumir en el orden establecido por la rutina que calcula la combinatoria, sin embargo esto complicaba la instrumentación, por ello, pensando en un diccionario o en la manera de mapear, se utiliza el tipo de dato map ofrecido por C++, instrumentado de la forma donde el tipo de dato entero es la clave asociado al tipo de dato *Node\** , de manera que si ordeno el diccionario en el orden que corresponde al menor recorrido, solo debo seguir esa ruta.

Para la búsqueda de qué camino es el más corto, se realiza una sumatoria de las distancias acumuladas desde un destino al otro, se hace todo sobre un mismo camino aprovechando el hecho de que para cada ejecución, éste se reinicia y no se concatena, esto se hace en el orden de las 24 combinaciones posibles y al final se almacena el resultado de la distancia total de cada combinación en un vector. Este vector se imprime y se inspecciona visualmente en búsqueda de la menor distancia. La posición de la menor distancia debe coincidir con la el número de fila en la matriz de combinaciones creada previamente, esto indica que están de algún modo mapeadas, circunstancia a la que se le sacará provecho posteriormente. Resta entonces una manera de ordenar esas distancias. Los avances se encuentran en el archivo prueba3.C y prueba4.C

# Quinto Avance

Antes de ordenar el vector de distancias se observa que se tiene la menor distancia asumiendo que se parte desde uno de los destinos, cosa que en la vida real carece de sentido pues si uno de los destinos es un punto de partida en la primera iteración, ese paquete no tendría necesidad de bien ser entregado o ser buscado. Se tiene entonces la necesidad de calcular la distancia desde un punto llamado nodo origen que representa el lugar donde estaría el conductor al recoger todos los paquetes pues la menor distancia de entrega depende de la posición. Esto se soluciona calculando la distancia desde el nodo origen hasta el primer nodo de cada combinación justo antes de entrar al ciclo iterativo de cálculo de distancias instrumentado en el avance anterior, volviendo a aprovechar el hecho del reinicio del camino.

Una vez que se obtiene la verdadera menor distancia, resulta más atractivo ordenar de algún modo las distancias y sus recorridos para facilitar el acceso y comparar, si así se quisiese. Se instrumenta el ordenamiento de burbuja por dos simples razones:

-Fue fácil

- Se parte del hecho de que para que un problema sea optimizado, se debe garantizar su existencia. Tal como uno de los problemas de grafos, una vez se tiene el problema, se ha de determinar si existe un grafo que lo solucione.

Como se ordena el vector de distancias que está mapeado a la matriz de combinaciones, el ordenamiento del vector funge como referencia para ordenar las combinaciones, visto gráficamente:

Combinaciones Distancias

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 0 | 2 |
| 2 | 0 | 1 | 3 |
| 3 | 0 | 2 | 1 |
| 1 | 2 | 3 | 0 |
| 0 | 3 | 2 | 1 |
| 0 | 1 | 2 | 3 |
| 2 | 1 | 3 | 0 |
| 3 | 2 | 1 | 0 |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |

|  |
| --- |
| 4560.97 |
| 5674.38 |
| 4896.21 |
| 8907.65 |
| 9006.52 |
| 4555.98 |
| 3789.43 |
| 8765.43 |
| . |
| . |
| . |
| . |

Partiendo de 0, se observa que la menor distancia corresponde a la posición 6, lo que indica que de los cuatro nodos destino, asignados en un orden cualquiera pues no depende del momento del pedido de su ubicación, ha de buscarse primero al 2, luego al 1, luego al 3, y por último al cero. Todo se calcula en base al nodo origen, que es la posición actual del conductor.

Finalmente se obtiene un algoritmo que, dados un nodo origen y cuatro destinos, calcula la menor distancia de todas las combinaciones posibles y luego ordena para entregar la menor posición y la menor ruta en la primera fila de la matriz correspondiente.

Para la impresión en pantalla se vuelve a hacer el mismo recorrido tomando en cuenta solo el nodo perteneciente a la ruta e imprimiéndolo en pantalla.

Finalmente se requiere entonces una interfaz que sea intuitiva, de fácil manejo y que muestre en un mapa real de la ciudad de Mérida la ruta establecida para complementar el proyecto dodri-optravel.(los detalles de impresión se encuentran en prueba.C)

# Sexto Avance

Para el último avance se añade coordenadas que en principio son para la parte gráfica. Se crea un nuevo tipo de nodo que almacene el nombre del nodo y dos números en punto flotante para hacer las veces de la latitud y la longitud. Así mismo, se crea una rutina que exporta en un txt las coordenadas para que sea aprovechado por la aplicación creada en QT. Hasta el momento, se tiene todo lo necesario para enviar los datos a la parte gráfica quedando pendiente la conexión entre ambas partes.