**Universidad Autónoma Metropolitana**
**Unidad Azcapotzalco**

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

# *"BIBLIOTECA EN LENGUAJE C PARA RESOLUCIÓN DE PROBLEMAS DE DISEÑO DE REDES"*

Marco Antonio Cervantes Aparicio 206309616

Asesor: Dr. Pedro Lara Velázquez

# Contenido

Resumen ..... 2
Introducción ..... 2
Marco Teórico ..... 3
Problemas a resolver y algoritmos para su solución. ..... 3
Ruta más corta ..... 3
Ruta crítica ..... 4
Árbol de expansión mínima ..... 6
Implementación de los algoritmos ..... 7
Algoritmo para encontrar la ruta más corta de una red ..... 7
Algoritmo para encontrar el árbol de expansión mínima de una red ..... 8
Algoritmo para encontrar la ruta critica de una red ..... 10
Casos de Prueba ..... 11
Pruebas al algoritmo de la ruta más corta ..... 12
Pruebas al algoritmo para generar un árbol de expansión mínima ..... 15
Pruebas al algoritmo que define la ruta crítica de un proyecto ..... 18
Conclusiones ..... 21
Bibliografía ..... 23

## Resumen

El presente proyecto está enfocado al desarrollo de herramientas informáticas de un tema en específico de la investigación de operaciones (IO), las redes de optimización. Las herramientas contenidas en el presente proyecto son algunos algoritmos que utilizan estructuras de red para resolver problemas de optimización.

Los algoritmos que conforman el proyecto son los siguientes:

- Algoritmo para encontrar la ruta más corta en una red.
- Algoritmo para encontrar el árbol de expansión mínima en una red.
- Algoritmo para crear la ruta crítica en una red.

Este proyecto incluye una explicación de la estructura, funcionamiento, además de un conjunto de pruebas que mostrarán, de manera más clara, el funcionamiento de estos algoritmos dentro de la computadora. Todo esto con el fin de facilitar su comprensión y futuros usos que le pueda dar la comunidad universitaria.


## Introducción

La IO se dedica al planteamiento, aplicación y solución de problemas mediante modelos matemáticos, siempre en busca de la optimización, es decir, el uso eficiente de los recursos, es por esta razón la importancia de su estudio detallado. La IO cubren una gran cantidad de tópico, dentro de los cuales se encuentran los problemas que al ser planteados con un modelo en red son más fáciles de solucionar. Estos problemas que usan modelos de redes surgen en una gran variedad de situaciones y son usados de manera amplia en áreas tan diversas como producción, distribución, planeación de proyectos, localización de instalaciones, administración de recursos y planeación financiera por mencionar algunos ejemplos.

Si bien es verdad que existen algoritmos y paquetes computacionales muy sofisticados que se usan en forma rutinaria para resolver problemas de gran tamaño que no se habrían podido manejar hace dos o tres décadas, la mayor parte de estos están "incrustados" en el software sólo para desempeñar tareas específicas y algunos otros se encuentran bajo algún tipo de licencia que no permite el uso directo de los algoritmos para resolver problemas en red. Esta clase de software es muy útil, sin embargo, para la innovación, desarrollo, investigación o estudio de las redes, dichas aplicaciones al ser fijas o al formar parte de un grupo de aplicaciones pierden claridad para aquellos que desee conocer su funcionamiento.

Este proyecto está dirigido a parte de la comunidad universitaria, específicamente aquellos que tenga relación con la IO (tanto docentes como alumnos). El objetivo del proyecto es proporcionar un conjunto de algoritmos usados en la resolución de problemas con un modelo en red dentro de la IO que cuente con su documentación y pruebas correspondientes para clarificar su funcionalidad y uso. Esto con el fin de

facilitar el acceso y disposición de dichos algoritmos para ser empleados y adaptados a las necesidades de quien desee utilizarlos.

## Marco Teórico

Winston, W. nos menciona que una red, también llamada *gráfica*, se define mediante dos conjuntos de símbolos: nodos y arcos. Para diferenciar estos elementos primero definiremos un conjunto "N" que se encargue de contener los nodos que tiene la red y un conjunto "A" que contenga los arcos de la red.

Los nodos se definen como puntos extremos dentro de la red, mientras que los arcos se definen como un par ordenado de puntos extremos y representa una posible dirección de movimiento que podría ocurrir entre puntos extremos.

Los arcos al ser un par ordenado tendrán el siguiente formato: si existe un arco (j, k), entonces existe movimiento posible entre el nodo j y el nodo k. Simultáneamente para el arco (j, k), se dirá que el nodo *j* es el nodo inicial y el nodo *k* es el nodo terminal.

Se dice que un arco está dirigido u orientado si permite un flujo positivo entre los nodos sólo en una dirección. Una *red dirigida* debe tener todos los arcos dirigidos. Una red *no dirigida* solo tiene arcos que no presentan restricción en su dirección de flujo.

Una secuencia de arcos tal que cada arco tiene exactamente un nodo en común con el arco previo, se llama una *cadena*. Una *trayectoria* o *ruta* es una cadena en la que el nodo terminal de cada arco es idéntico al nodo inicial del arco siguiente. Una ruta forma un *ciclo* o *bucle* si conecta un nodo de vuelta a sí mismo a través de otros nodos.

Se dice que una red está *conectada* si cada dos nodos distintos están conectados en al menos una ruta. Un árbol es una red conectada libre de *ciclos* compuesta de un subconjunto de todos los nodos (Winston 2005).

## Problemas a resolver y algoritmos para su solución.

Los problemas que se resolvieron y los algoritmos que se utilizaron se describen a continuación:

### Ruta más corta

El problema de la ruta más corta supone una red *conectada* y *dirigida* en la cual cada arco tiene un costo (longitud, gasto, requerimiento, etc.) asociado. El algoritmo inicia en un nodo particular que llamaremos "Origen" a partir del cual se busca llegar a un nodo que llamaremos "Destino" formando una *ruta* cuyo costo sea el mínimo.

Para resolver este problema el algoritmo de *Floyd* es una buena alternativa. Taha, H. menciona que en esencia el algoritmo de Floyd incluye al algoritmo de *Dijkstra* por eso es más general que éste último. El algoritmo de Floyd se encarga de determinar el costo de la ruta entre dos nodos cualquiera dentro de la red. Usualmente el algoritmo representa los costos de una red de *n* nodos con una matriz cuadrada "C" de *n* filas (que representaran los *n* nodos que pueden ser tomados como Origen) y *n* columnas (que

representaran los *n* nodos que pueden ser tomados como Destino). En la red, si existe un arco (i, j) su costo, al cual llamaremos "$c_{ij}$" será finito y guardado en C(i, j), si el arco (i, j) no existe el costo "$c_{ij}$" será infinito y guardado igualmente en C(i, j).

La idea del algoritmo de Floyd se basa en una *operación triple* de costos. Dado un arco (i, j) con costo "$c_{ij}$" tomado como posible arco de la ruta más corta, esta puede ser remplazado por una *cadena* formada por un arco (i, k) y un arco (k, j) cuyos costos "$c_{ik}$" y "$c_{kj}$" cumplen que "$c_{ik} + c_{kj} < c_{ij}$". Si tenemos una red de *n* nodos podemos crear nuestra matriz "C" y definirla fila *k* y la columna *k* como *fila pivote* y *columna pivote* (con $0 < k \leq n$) y para cada *i* y *j* aplicamos la *operación triple* a su correspondiente "$c_{ij}$", siempre y cuando se cumpla lo siguiente: "$c_{ik} + c_{kj} < c_{ij}$" $i \neq k$, $j \neq k$ y $i \neq j$.

Si aplicamos la operación triple a todos los elementos de la matriz "C" después de *n* pasos podremos determinar el costo de la ruta más corta entre cualquier nodo *i* y nodo *j* de la red (Taha 2011).

## Ruta crítica

La necesidad de determinar de la ruta crítica surge en programación de proyectos formados por actividades interrelacionadas, en las cuales, su realización consume tiempo y recursos. El objetivo es programar el proyecto de tal manera que todas las actividades que lo forman sean realizadas en el menor tiempo posible. Para lograrlo podemos usar como herramienta, los modelos de red para representar el proyecto.

Winston (2005) agrega que:

> Para cada actividad, hay un conjunto de actividades (llamadas *predecesores* de la actividad) que deben completarse antes que comience la actividad. Un proyecto de red se utiliza para representar las relaciones de precedencia entre actividades. En este análisis, las actividades se representan con arcos directos, y los nodos se utilizan para representar la terminación de un conjunto de actividades. (Por esta razón, a menudo se hace referencia a los nodos del proyecto como *eventos*) Este tipo de red de proyecto de llama red AOA (por sus siglas en inglés-actividad en arco).

Para generar la red dirigida de un proyecto Winston, W. señala que es necesario contar con una lista de actividades y predecesores. Ya teniendo dichas listas se podrá construir la red si se sigues las siguientes reglas:

- El nodo 1 representa el inicio del proyecto, por lo que todas las actividades que no tengan predecesores irán unidas a este nodo.
- Se debe incluir en la red un nodo (llamado nodo de terminación) que represente el fin del proyecto.
- Se debe numerar los nodos de manera que el nodo que representa el final de una actividad siempre tenga un número más grande que el nodo que representa el inicio de una actividad.
- Una actividad no se debe representar por más de un arco en la red.
- Dos nodos pueden estar conectados por más de un arco.

En ocasiones para lograr cumplir las últimas dos reglas se requiere el uso de *actividades ficticias* que no consumen tiempo ni recursos, pero ayudan a crear relaciones con actividades que tiene más de un predecesor (Winston 2005).

Por otro lado Taha, H. explica que el objetivo del algoritmo CPM es realizar un cronograma en base a la duración total del proyecto y sabiendo cual actividad del proyecto es *crítica* y cual es *no crítica*. Una actividad es crítica si sus tiempos de inicio y terminación están predeterminados (fijos). Una actividad no crítica es aquella que puede ser programada en un espacio de tiempo mayor a su duración. Es decir tiene tiempos de inicio y terminación flexibles (dentro de los límites). Por esta razón si retrasamos la realización de una actividad crítica se generará una demora en la terminación del proyecto, sin embargo, una demora en una actividad no crítica puede que no afecte la fecha de término de proyecto ya que esta cuenta con cierta holgura. Se considera que el proyecto llega a su fin cuando se terminan todas las actividades
(Taha 2011).

Para Winston (2005)

> Los modelos de red se pueden utilizar como una ayuda en la programación de proyectos complejos de gran tamaño que consisten de muchas actividades. Si la duración de cada actividad se conoce con certeza, entonces el ***método de la ruta crítica*** (CPM por sus siglas en inglés) se utiliza para determinar la longitud del tiempo requerido para completar el proyecto. El CPM también se utiliza para determinar cuánto se puede retardar cada actividad del proyecto sin retrasar la terminación del mismo. Investigadores de DuPont y Sperry Rand desarrollaron el CPM a finales de la década de 1950.

Para realizar el algoritmo CPM, Winston, W. comenta que existen dos componentes clave que deben ser calculados sobre todos los *eventos* (nodos) de la red, dichos conceptos son conocidos como tiempo inicial (ET) y tiempo tardío (LT) del evento. Es decir, el tiempo inicial del evento para el nodo *i*, representado por ET(*i*), es el primer momento en que ocurre el evento que corresponde al nodo *i*. Y el tiempo tardío del evento para el nodo *i*, representado por LT(*i*), es el último momento en que puede ocurrir un evento que corresponde al nodo *i* sin retrasar la terminación del proyecto.

Para calcular el tiempo inicial (ET) de cada nodo en la red del proyecto, primero hay que definir el tiempo inicial del nodo 1 que no gasta tiempo ni recursos, es decir, el ET(1) = 0. Después de esto se calcula en secuencia ascendente ET(2), ET(3),…, ET(n), con n = número total de nodos. Para calcular el tiempo inicial del nodo *i* o ET(*i*) se siguen los siguientes pasos:

- Identificamos los predecesores inmediatos (nodos que están conectados por un arco y ocurren antes del nodo *i*) del nodo *i*.
- Para cada predecesor del nodo *i* sumamos su ET con el costo del arco (actividad) que lo une al nodo *i*.
- El valor de ET(*i*) es el valor máximo de los valores calculados anteriormente.

Una vez calculados los tiempos iníciales (ET) de todos los nodos de la red, hay que calcular el tiempo de evento tardío de cada nodo. Para calcular cualquier LT(i), debemos estar posicionados en el último nodo y hacer un recorrido en retroceso hasta llegar al nodo 1 o nodo de inicio del proyecto realizando lo siguiente:

- Identificamos a los sucesores inmediatos (nodos que están conectados por un arco y ocurren después del nodo *i*) del nodo *i*.
- Para cada LT de los sucesores inmediato al nodo *i*, se resta la duración de la actividad que los une al nodo *i*.
- El valor del LT(i) es el valor mínimo de los valores calculados anteriormente.

Una vez contando con el tiempo inicial (ET) y el tiempo de evento tardío (LT) de cada nodo, podemos definir qué actividades son críticas y cuáles no, esto con ayuda del concepto de *tiempo libre total* (TF), este tiempo representa el tiempo total en que podríamos retrasar el inicio de una actividad sin retrasar con ello el término del proyecto.

El tiempo libre total se calcula restando al tiempo de evento tardío del nodo, el tiempo inicial del mismo nodo. Si una actividad tiene un tiempo libre total de cero, significará que ésta no cuanta con holgura para realizarse por lo que se considera una actividad crítica.

Una vez calculados los TF de cada nodo podemos identificar las actividades críticas y formar una ruta con ellas que inicie en el nodo 1 y concluya en el nodo final n (Winston 2005).

## Árbol de expansión mínima

El problema que presenta la construcción de un árbol de expansión mínima es para Hillier, F. y Liberman, G., comparable con el problema de la ruta más corta. Para una red *no dirigida* y *conectada* en la cual existe una longitud positiva (distancia, costo, tiempo, etc.) asociada a los arcos de la red al igual que el algoritmo de la ruta más corta se deben seleccionar un conjunto de arcos cuya sumatoria total sea la menor posible. Sin embargo, a diferencia del algoritmo de la ruta más corta entre dos nodos cualesquiera, el algoritmo que genera el árbol de expansión mínima busca que los arcos seleccionados garantice una ruta entre cada par de nodos en la red. (Hillier y Liberman 2010)

Un árbol de expansión, es un grupo de n-1 arcos que conectan a todos los nodos de la red y no contienen bucles (ciclos). Por lo que un árbol de expansión cuyo costo total es el mínimo posible es conocido como *árbol de expansión mínima* (MST por sus siglas en ingles).

Para construir un árbol de expansión mínima el algoritmo de Kruskal ofrece una buena opción. Agüero, R. menciona que el algoritmo construye el MST incorporando paulatinamente los arcos que existen en dos componentes diferentes (dos subredes no conectadas entre sí); los nodos de la red que no forman parte del árbol de expansión mínima y los nodos de la red que ya forman parte del árbol de expansión mínima.

El algoritmo realiza lo siguiente para formar el árbol de expansión mínima:

- Recorre todos los arcos de la red que no forman parte del árbol de expansión mínima e identifica el arco (i, j) con menor costo.
- Revisa que el nodo *i* y el nodo *j* no estén en el mismo árbol, en caso contrario descarta el arco.
- Crea un árbol “C” uniendo el árbol al que pertenece el nodo i y el árbol al que pertenece el nodo j.

- Posteriormente se marca el nodo *i* y el nodo *j* como pertenecientes a un mismo árbol con una raíz en común.
- Se marca al arco (i, j) como parte del árbol de expansión mínima para evitar que se vuelva a usar.
- El proceso se repite hasta cubrir todos los nodos (Agüero 2012).

# Implementación de los algoritmos

## Algoritmo para encontrar la ruta más corta de una red

El algoritmo para encontrar la ruta más corta de una red funciona a través de la función llamada "GenRutaCorta" localizada en el archivo biblioteca "RutaCorta.h". La función "GenRutaCorta" recibe tres parámetros; el primero es el nombre del archivo ASCII donde se localizan datos de los nodos y arcos que formarán la red; el segundo es el número del nodo que deseamos tomar como "Origen" y el tercer parámetro es el número del nodo "Destino" al que deseamos llegar. El pseudocódigo del algoritmo se describe a continuación:

*GenRutaCorta (NombreArchivo, Origen, Destino)*

*Si Origen >= Destino entonces regresa "No hay camino"*

*Si no Tabla = Extraer y Ordenar ("NombreArchivo", Origen, Destino)*

*Si Tabla ≠ Nulo entonces*

*Revisa Factibilidad (Tabla, Origen, Destino)*

*CostosMinimos=Floyd (Tabla, Origen, Destino)*

*CaminoBuscado = CaminoCorto (CostosMinimos, Origen, Destino)*

*Regresa Caminobuscado*

*Si no regresa "No hay camino"*

El archivo ASCII debe contener los datos de la red con el formato siguiente:

- La primera línea del archivo ASCII debe contener el número de arcos de la red, un espacio en blanco como separador y después el número de nodos que existen en la red. Por ejemplo, si en la primera línea está escrito "26 15", esto significará que la red está formada por 15 nodos conectados por 26 arcos.
- Las líneas siguientes del archivo deben contener los datos de cada arco (nodo inicial, nodo final, costo del arco). Los datos deberán seguir el siguiente orden: primero el número del nodo inicial donde comienza el arco, un espacio en blanco como separador seguido por el número del nodo destino donde termina el arco, un espacio en blanco como separador y por último el costo del arco. Por ejemplo, si escribimos "1 2 10", esto significará que existe un arco que une al nodo 1 con el nodo 2 y tiene un costo de 10. Cabe mencionar que cada terna de datos debe escribirse en renglones independientes, ordenados de manera

ascendente de preferencia, para mantener un orden al momento de extraer los datos al programa.

Ya explicada la estructura del archivo ASCII de entrada, a continuación se explica cómo funciona el algoritmo implementado.

El programa hace uso de nueve funciones (sin contar la función principal "GenRutaCorta"), para generar un vector de salida de datos donde se muestra el camino más corto encontrado en la red previamente especificada. A continuación, en la tabla, se pueden observar las funciones existentes en el código y una breve explicación de su función.

| Función | Descripción |
|---|---|
| Datos | Recibe el nombre del archivo ASCII donde se encuentran los datos de la red, extraer los datos del archivo y los coloca en una matriz para su uso en el programa. |
| OrdenaDatos | Se encarga de ordenar los datos de la matriz que contiene los elementos de la red. Esta función sólo garantiza el orden de los datos para el funcionamiento del algoritmo. Puede omitirse si se tiene la certeza que los datos ya están ordenados. |
| Factibilidad | Revisa la matriz de los datos de la red y determina si puede existir o no una ruta que inicie en el nodo "*Origen*" y termine en el nodo "*Destino*". |
| CaminoCorto | Valiéndose del resultado del algoritmo de Floyd explorar las rutas posibles y localiza la ruta más corta del nodo "*Origen*" al nodo "*Destino*". |
| Floyd | Crea un arreglo de tamaño "*n*", donde "*n*" es el número de nodos, posteriormente aplica el algoritmo de Floyd a la red y llena el arreglo solo con los costos mínimos de las rutas formadas de todos los nodos de la red hacia el nodo "*Destino*". |
| GeneraCamino | Se encarga de extraer de la red y almacenar en una matriz los datos de los arcos que forman el camino más corto |
| GuardarArchivo | Da la opción al usuario de guardar los resultados obtenidos por el programa. |
| Libera | Libera la memoria utilizada por las matrices creadas de manera dinámica. |
| Minimo | Compara dos números y regresa el más pequeño. |

Para mayores detalles acerca de las funciones se incluyeron en la biblioteca comentarios que pormenorizan las variables involucradas y como se usan en cada función.

## Algoritmo para encontrar el árbol de expansión mínima de una red

El Algoritmo para encontrar el árbol de expansión mínima de una red funciona a través de la función "ArbolExMin" localizada en el archivo biblioteca "ArbolExpMinima.h". La función recibe como único parámetro el nombre del archivo ASCII donde se

localizan datos de los nodos que formaran la red. El pseudocódigo de este algoritmo es el siguiente:

*ArbolExMin (NombreArchivo)*

*Matriz = Extraer y Ordenar ("NombreArchivo")*

*Si Matriz = Nulo entonces regresa Nulo*

*ArbolMin = Kruskal (Matriz, NNodos, NArcos)*

*Regresa ArbolMin*

En cuanto al archivo ASCII, esté debe contener los siguientes datos de la red con el formato siguiente:

- La primera línea del archivo ASCII debe contener el número de arcos de la red, un espacio en blanco como separador y consecuentemente el número de nodos que existen en la red. Por ejemplo, si en la primera línea está escrito "14 9", esto significara que la red está formada por 9 nodos conectados por 14 arcos
- Las líneas siguientes del archivo deben contener los datos de cada arco (nodo inicial, nodo final, costo del arco). Los datos debe estar ordenados de la siguiente manera: primero el número del nodo inicial donde comienza el arco, un espacio en blanco como separador seguido por el número del nodo destino donde termina el arco, un espacio en blanco como separador y por último el costo del arco. Por ejemplo, si escribimos "3 6 9", esto significara que existe un arco que conecta el nodo 3 con el nodo 6 y tiene un costo de 9. Cabe mencionar que cada terna de datos debe escribirse en renglones independientes forzosamente ordenados ascendentemente para mantener un orden al momento de extraer los datos al programa.

Ya explicada la estructura del archivo ASCII de entrada, procederemos a explicar cómo funciona el algoritmo implementado.

El programa hace uso de ocho funciones (sin contar la función principal "ArbolExMin"), para generar un vector de salida donde se almacenaran los arcos con sus nodos y costo que forman un árbol de expansión mínima. A continuación en la tabla, se pueden observar las funciones existentes en el código y una breve explicación de su función en el mismo.

| Función | Descripción |
|---|---|
| MatrizAdy | Recibe el nombre del archivo ASCII donde se encuentran los datos de la red, extraer los datos del archivo y los coloca en una matriz. |
| Mezcla | Función que usa el algoritmo de ordenamiento por mezcla (merge sort) para ordenar la matriz que contiene los datos de la red en base al costo del arco. |
| Mezclar | Función auxiliar del algoritmo de ordenamiento por mezcla que se encarga de integrar partes de la matriz en orden. |
| Raiz | Busca el nodo raíz de una estructura de árbol ya formada. |

| | |
|---|---|
| UnionArbol | Une los nodos raíz de dos estructuras de árbol diferentes en un solo árbol. |
| MostrarArbol | Función encargada de mostrar en pantalla los resultados y preguntar si se desean guardar. |
| GuardarArchivo | Da la opción al usuario de guardar los resultados obtenidos por el programa en un archivo. |
| Kruskal | Se encarga de buscar el arco con menor costo en la red y añadirlo al árbol de expansión mínima sólo si el nodo inicial y nodo final no pertenecen a un mismo árbol. Este proceso se repite hasta cubrir todos los arcos. |

Para mayores detalles acerca de las funciones se incluyó en las bibliotecas comentarios que pormenorizan las variables involucradas y como se usan en cada función.

## Algoritmo para encontrar la ruta crítica de una red

Este algoritmo funciona a través de la función principal "GenRutaCritica" la cual recibe como único parámetro el nombre del archivo ASCII donde se localizan datos de los nodos que formaran la red. El Pseudocódigo del algoritmo es el siguiente:

*GenRutaCritica (NombreArchivo)*

*Tabla = Extraer y Ordenar ("NombreArchivo")*

*Si Tabla = Nulo entonces regresa "No hay datos"*

*Crea nodo "InicioProyecto"*

*Para i = 0 hasta NumerodeArcos haz*

*ActProyecto= AñadirActividad (NombreActividad, ActAntecesora, Costo)*

*Calcula ET de Actividad*

*Calcula LT (ActProyecto)*

*RutaCritica = GeneraCamino (ActProyecto)*

*Regresa RutaCritica*

El archivo ASCII para la entrada, debe contener los siguientes datos de la red en con el formato siguiente:

- La primera línea del archivo ASCII debe contener el número total de arcos o actividades en la red del proyecto. Por ejemplo, si en la primera línea está escrito "17", esto significara que la red del proyecto está formada por 17 actividades.
- Las líneas siguientes del archivo deben contener los datos de cada arco con las siguientes reglas: primero una letra mayúscula que identifique una actividad dentro del proyecto, un espacio en blanco como separador, seguido de una posible actividad antecesora, es decir, si la actividad escrita al principio tiene una actividad antecesora dentro del proyecto, entonces se escribe una segunda letra mayúscula que identifique a la actividad antecesora, en caso que no exista

dicha actividad se debe colocar un guion "-" para indicar que no existen antecesores, después, un espacio en blanco como separador y por último el costo de realización de la actividad. Por ejemplo, si escribimos "A - 14", esto significara que existe una actividad "A" que no tiene actividades antecesoras y que realizarla nos cuesta "14", otro ejemplo seria "B A 5", esto significara que existe la actividad "B", la cual tiene como antecesor a la actividad "A" y que realizar la actividad "B" nos cuesta "5". Cabe mencionar que cada terna de datos debe escribirse en renglones independientes para mantener un orden al momento de extraer los datos al programa.

Ya explicada la estructura del archivo ASCII de entrada, procederemos a explicar cómo funciona el algoritmo implementado.

El programa hace uso de diez funciones (sin contar la función principal "GenRutaCritica"), para generar un vector de salida con la secuencia de las actividades que forman la ruta crítica del proyecto. A continuación en la tabla, se pueden observar las funciones existentes en el código y una breve explicación de su función en el mismo.

| Función | Descripción |
|---|---|
| Datos | Recibe el nombre del archivo ASCII de entrada, extrae e inserta los datos de las actividades en una estructura para su posterior inserción en la estructura de red del proyecto. |
| OrdMezcla | Función que usa el algoritmo de ordenamiento por mezcla (merge sort) para ordenar la matriz que contiene los datos de la red en orden alfabético en base a la letra de cada actividad. |
| MezclaResultado | Función auxiliar del algoritmo de ordenamiento por mezcla que se encarga de integrar partes de la matriz en orden. |
| AnadirActividad | Recibe una a una las actividades (arcos) del proyecto y las inserta en la estructura de red del proyecto en el lugar donde les corresponda. |
| BuscarNodo | Busca y rectifica en la estructura de red la existencia de algún nodo relacionado con una actividad en la red. |
| Max | Compara dos valores y regresa el más grande |
| Min | Compara dos valores y regresa el más pequeño |
| GeneraCamino | Genera la ruta crítica eligiendo las actividades cuyo tiempo libre total (TF) sea igual a 0. |
| MostrarResultado | Función encargada de mostrar en pantalla los resultados y preguntar si se desean guardar. |
| GuardarArchivo | Da la opción al usuario de guardar los resultados obtenidos por el programa en un archivo. |

Para mayores detalles acerca de las funciones se incluyó en las bibliotecas comentarios que pormenorizan las variables involucradas y como se usan en cada función.

## Casos de Prueba

En esta sección se explicará el plan de pruebas que se siguió para garantizar el cumplimiento de los requisitos planteados para el proyecto, los cuales son:

- Generar una estructura que represente la red.
- Aplicar el algoritmo correspondiente a la estructura de red
- Generar una matriz resultado del algoritmo.
- Guardar en un archivo los resultados obtenidos.

Cada biblioteca debe cumplir con estos requisitos que se acordaron en el proyecto para la evaluación de su funcionamiento.

## Pruebas al algoritmo de la ruta más corta

En la biblioteca "RutaCorta.h" la función que se encarga de generar y almacenar la estructura de red es la función "Datos", a la cual le aplicamos los siguientes casos de prueba para garantizar su funcionalidad:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP001 | Datos | Marcar error al recibir un archivo ASCII de entrada vacío. | -Contar con un archivo ASCII de entrada en blanco (vacío). | Detener el programa al no encontrar datos en el archivo ASCII de entrada. | Excepción en acceso a memoria por índice no iniciado. | Reparado |
| CP002 | Datos | Verificar que existan los suficientes datos del archivo de entrada. | -Contar con un archivo ASCII que contenga datos de la red (Número de arcos y nodos, además de los datos de cada arco) incompletos. | Reportar que no se tienen todos los datos de la red. | OK | Correcto |
| CP003 | Datos | Verificar que todos los datos del archivo de entrada estén almacenados en una matriz. | -Contar con un archivo ASCII que contenga datos de la red (Número de arcos y nodos, además de los datos de cada arco) correctos. -Haber creado y una matriz de *n**3, donde "*n*" es el número de arcos. | Todos los datos de la red deben estar almacenados en la estructura de datos matricial. | OK | Correcto |

El algoritmo que se usó para encontrar la ruta con menor costo es el algoritmo de Floyd, el cual es realizado por la función "Floyd" dentro del programa. A esta función se le aplicaron los casos de prueba siguientes:

| Caso de Prueba | Función a probar | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP004 | Floyd | Asegurarse que el algoritmo calcula el costo mínimo de las rutas de todos los nodos de la red hasta el nodo "*Destino*". | -Haber extraído y guardado los datos de los arcos de la red en una matriz.<br>-Tener Definido los nodos "*Origen*" y "*Destino*" de la ruta buscada. | Matriz con los costos mínimos de las rutas de todos los nodos al nodo destino. | OK | Correcto |
| CP005 | Floyd | Asegurarse que el algoritmo guarda el costo mínimo calculado en un arreglo en la posición correspondiente (tomando como índice el número del nodo). | -Haber extraído y guardado los datos de los arcos de la red en una matriz.<br>-Tener Definido los nodos "*Origen*" y "*Destino*" de la ruta buscada.<br>-Haber creado un arreglo para guardar los costos mínimos. | Para cualquier elemento *"i"* en la matriz debe contener el costo mínimo del nodo *"i"* al nodo "*Destino*". | OK | Correcto |

Ahora, para generar la matriz de resultados se diseñó la función "GeneraCamino" que hace uso de la matriz de costos mínimos que genera el algoritmo de Floyd y otras matrices derivadas de la misma. Las pruebas que se contemplaron fueron las siguientes.

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP006 | Genera Camino | Verificar que la matriz que contenga los arcos que forman el camino más corto tenga una secuencia entre los nodos. | -Haber calculado las rutas del nodo "*Origen*" al nodo "*Destino*" con ayuda de la matriz de costos de Floyd.<br>-Haber identificado y marcado la ruta con menor costo. | Una matriz que contenga los arcos que forman el camino más corto entre el nodo "*Origen*" y el nodo "*Destino*" en secuencia congruente. | Una matriz con los arcos que forman el camino más cortó con secuencia descontinúa en los nodos en una parte de la matriz. | Reparado |
| CP007 | Genera | Verificar que | -Haber | Una matriz | OK | Correcto |

| | | | | | | |
|---|---|---|---|---|---|---|
| | Camino | solo los arcos marcados como perteneciente s al camino corto sean guardados en la matriz de resultados. | calculado las posibles rutas del nodo "*Origen*" al nodo "*Destino*" con ayuda de la matriz de costos de Floyd.<br>-Haber identificador y marcado la ruta con menor costo. | que en cada fila contenga sólo la información de los arcos (origen, destino, costo) que forman el camino más corto. El último elemento de la matriz debe ser NULO. | | |

Una vez cumplidos los requisitos anteriores solo falta asegurar que podernos guardar el archivo, lo cual es trabajo de la función "GuardarArchivo" que dará la opción de guardar los resultados en un archivo ASCII para cualquier uso que el usuario desee darle.

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP008 | Guardar Archivo | Asegurar que el archivo en donde se pretende guardar los datos es creado correctamente. | -Tener el nombre para el nuevo archivo.<br>-Contar con la matriz de resultados y la sumatoria de sus costos. | Un nuevo archivo ASCII cuyo contenido contendrá los arcos que forman la ruta más corta y el costo total de la misma. | OK | Correcto |
| CP009 | Guardar Archivo | Verificar que sólo la información de los resultados sea insertada en el archivo, sin ninguna clase de información basura. | -Contar con la matriz de resultados y la sumatoria de sus costos.<br>-La matriz de resultados debe tener su último elemento apuntando a NULO. | El contenido del archivo debe estar conformado sólo por los arcos del camino más corto y el costo total de este (sin la existencia de caracteres basura). | OK | Correcto |

## Pruebas al algoritmo para generar un árbol de expansión mínima

La biblioteca "ArbolExpMinima.h" usa la función "MatrizAdy" para extraer los datos de la red del archivo ASCII de entrada para poder trabajar sobre ellos, aquí los casos de prueba usados para esta función:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP001 | MatrizAdy | Marcar error al recibir un archivo ASCII de entrada vacío. | -Contar con un archivo ASCII de entrada en blanco (vacío). | Detener el programa al no encontrar datos en el archivo ASCII de entrada. | Excepción en acceso a memoria por índice no iniciado. | Reparado |
| CP002 | MatrizAdy | Verificar que existan los suficientes datos del archivo de entrada. | -Contar con un archivo ASCII que contenga datos incompletos de la red (Número de arcos y nodos, además de los datos de cada arco). | Reportar que no se tienen todos los datos de la red. | OK | Correcto |
| CP003 | MatrizAdy | Verificar que todos los datos del archivo de entrada estén guardados en una matriz. | -Contar con un archivo ASCII que contenga datos completos de la red (número de arcos y nodos, además de los datos de cada arco).<br>-Haber creado y una matriz de *n**3, donde "*n*" es el número de arcos. | Todos los datos de la red deben estar almacenados en la estructura de datos matricial. | OK | Correcto |

Para el correcto funcionamiento del algoritmo los datos extraídos del archivo de entrada se deben orden de una manera especial usando el algoritmo de ordenamiento por mezcla. Se debe realizar un ordenamiento en base al costo de los arcos de menor a mayor. El algoritmo de ordenamiento por mezcla está formado por dos funciones "Mezcla" y "Mezclar" a las cuales se les aplicaron las pruebas siguientes:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| CP004 | Mezcla | Verificar que la matriz de datos sea dividida hasta llegar al caso base de un solo elemento. | -Haber extraído y guardado en una matriz los datos de los arcos de la red. | Tener en cierto punto del programa divisiones de bloques de la matriz de tamaño uno. | OK | Correcto |
| CP005 | Mezclar | Verificar que, dados dos bloques de la matriz de datos de la red, al mezclarlos los elementos del bloque resultante se encuentren ordenados. | -Haber extraído y guardado en una matriz los datos de los arcos de la red.<br>-Haber mezclado previamente ambos bloques los bloques. | Un solo bloque de datos ordenados que contenga todos los datos de los dos bloques mezclados. | OK | Correcto |

Ahora bien, el algoritmo que se usó para generar el árbol de expansión mínima de la red fue el algoritmo de Kruskal. Este algoritmo es realizado por la función "Kruskal" de la biblioteca, la cual a cada paso va formando la matriz resultado que contiene los arcos que forman el árbol de expansión mínima. Las pruebas fueron las siguientes:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP006 | Kruskal | Comprobar que el árbol de expansión mínima en la matriz resultado realmente tenga el costo mínimo. | -Contar con la matriz de datos de los arcos de la red.<br>-Haber creado una matriz de *m**3, donde "*m*" es el número de nodos de la red. | Una matriz con los arcos de la red que forman el árbol de expansión con el costo más bajo. | OK | Correcto |
| CP007 | Kruskal | Asegurarse que el conjunto de arcos de la matriz resultado realmente son un árbol. | -Haber extraído y almacenado en una matriz los datos de los arcos de la red.<br>-Haber creado una matriz de *m**3, donde "*m*" es el | Una matriz que contenga m-1 arcos ("*m*" igual al número de nodos), de los cuales, al menos debe aparecer una vez cada uno de los nodos de la red en | OK | Correcto |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | número de nodos de la red. | algún arco. | | |
| CP008 | Kruskal | Verificar que sólo existan *n*-1 arcos dentro de la matriz de resultados, donde "*n*" es el número de nodos en la red. | -Haber extraído y almacenado en una matriz los datos de los arcos de la red.<br>-Haber creado una matriz de *m**3, donde "*m*" es el número de nodos de la red. | Una matriz de "*m*" elementos ("*m*" igual al número de nodos), que contenga solo *n*-1 arcos de la red que no forman ciclos y cuya sumatoria total de costo sea la mínima. El elemento "*m*" debe apuntar a NULO. | OK | Correcto |

Entonces solo falta cumplir el requisito de guardar en un archivo ASCII. Para probar la función "GuardarArchivo" la cual nos permite guardar la matriz resultado en un archivo ASCII para cualquier uso que el usuario desee darle se consideró prudente realizar las siguientes pruebas:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP009 | Guardar Archivo | Asegurar que el archivo en donde se pretende guardar los datos es creado correctamente. | -Se debe saber el nombre para el nuevo archivo.<br>-Contar con la matriz de resultados y la sumatoria de sus costos. | Un nuevo archivo ASCII cuyo contenido contendrá los arcos que forman el árbol de expansión mínima y el costo total del mismo. | OK | Correcto |
| CP010 | Guardar Archivo | Verificar que sólo información de los resultados sea insertada en el archivo, sin ninguna clase de información basura. | -Contar con la matriz de resultados y la sumatoria de sus costos.<br>-La matriz de resultados debe tener su último elemento apuntando a | El contenido del archivo conformado sólo por los arcos del árbol de expansión mínima y el costo total de este (sin la existencia de | OK | Correcto |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | NULO. | caracteres basura). | | |

## Pruebas al algoritmo que define la ruta crítica de un proyecto

Los primeros casos de prueba fueron aplicados a dos funciones, en primer lugar la función "Datos" la cual sólo extrae los datos del archivo de entrada y la función "AnadirActividad" la cual forma la estructura de red para el proyecto. Las pruebas fueron las siguientes:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP001 | Datos | Debe registrarse un error al recibir un archivo ASCII de entrada vacío. | -Contar con un archivo ASCII de entrada vacío. | Detener el programa al no encontrar datos en el archivo ASCII de entrada. | Excepción en acceso a memoria por índice no iniciado. | Reparado |
| CP002 | Datos | Verificar que existan los suficientes datos del archivo de entrada. | -Contar con un archivo ASCII que contenga datos incompletos de la red (número de arcos y los datos de cada arco). | Reportar que no se tienen todos los datos de la red. | OK | Correcto |
| CP003 | Datos | Verificar que todos los datos del archivo de entrada estén guardados en una matriz. | -Contar con un archivo ASCII que contenga los datos completos de la red (número de arcos y los datos de cada arco)<br>-Haber creado y una matriz de *n**3, donde "*n*" es el número de arcos. | Todos los datos de los arcos de la red deben estar almacenados en una estructura de datos matricial. | OK | Correcto |
| CP004 | Anadir Actividad | Asegurarse que las actividades sin antecesor sean colocadas al inicio del proyecto | -Haber extraído y guardado en una matriz los datos de la red.<br>- Asegurarse que existir algún arco que tenga como antecesor el inicio del proyecto (actividad "–"). | Una estructura de red en la que los arcos que no tienen actividad antecesora (actividad "–") inicien su arco en el "evento" marcado por el nodo 1 de la | OK | Correcto |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | red. | | |
| CP005 | Anadir Actividad | Verificar que las actividades con algún antecesor tengan un arco que inicie en el antecesor y termine en un nodo "*evento*" que indique la finalización de la actividad. | -Haber extraído y guardado en una matriz los datos de la red.<br>-Asegurarse que exista alguna actividad (arco) que tenga como antecesor otra actividad que no sea el inicio del proyecto. | Una estructura de red en la que una actividad *"y"* que tienen como antecesor una actividad *"x"*, se encuentre conectada por un arco que inicie en el nodo donde concluye la actividad "*x"* y termine en un nuevo nodo donde concluirá la actividad "*y"*. | OK | Correcto |
| CP006 | Anadir Actividad | Verificar la creación correcta de actividades ficticias en el proyecto | -Haber extraído y guardado en una matriz los datos de la red.<br>-Debe existir alguna actividad (arco) con más de una actividad antecesora. | Una estructura de red en la que una actividad *"y"* que tienen como antecesor más de una actividad, creara un nodo antecesor y su arco representará una actividad ficticia que para crear una relación transitiva entre la actividad *"y"* y sus antecesores. | Una estructura de red en la que una actividad *"y"* que tienen como antecesor más de una actividad, usa un nodo con un arco que representan una actividad ficticia por cada predecesor de la actividad para usarla como una conexión transitiva entre sus antecesores. | Reparado |

Una vez creada la estructura de red, el algoritmo para generar la ruta crítica, el cual es conocido como CPM (CriticalPathMethod por sus siglas en inglés) es realizado por dos funciones, por una parte, el cálculo del tiempo inicial (ET) es calculado al generar la estructura por la función "AnadirActividad". Mientras que la función "CalculaLT" calcula el tiempo de evento tardío (LT). A estas dos funciones se les realizaros las siguientes pruebas:

| Caso de Prueba | Función a probar | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP007 | Anadir Actividad | Verificar que los tiempos iníciales (ET) sean calculados correctamente. | -Haber extraído y guardado en una matriz los datos de la red. | Cada nodo (evento) correspondiente a una actividad debe tener su tiempo inicial calculado correctamente con respecto a todos sus nodos antecesores al momento de ser insertada a la estructura. | Algunos nodos calculaban su tiempo inicial sólo con parte de sus antecesores, ignorando algunos de ellos. | Corregido |
| CP008 | CalculaLT | Asegurarse que los tiempos de evento tardío (LT) se calculen correctamente. | -Haber extraído y guardado en una matriz los datos de la red. | Cada nodo (evento) correspondiente a una actividad debe tener su tiempo de evento tardío calculado correctamente con respecto a sus nodos sucesores. | Algunos nodos calculaban su tiempo de evento tardío sólo con parte de sus sucesores, ignorando algunos de ellos. | Corregido |

Para probar la generación de la matriz de resultados, hacemos uso de la función "GeneraCamino" la cual guardará las actividades críticas del proyecto en un arreglo. Las pruebas fueron las siguientes:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP009 | Genera Camino | Asegurar que las actividades con tiempo libre igual a cero son tomadas como actividades críticas. | -Haber extraído y guardado en una matriz los datos de la red.<br>-Cada nodo debe tener su tiempo inicial (ET) y tiempo de evento tardío (LT) calculado. | Una arreglo de caracteres que tenga concatenados los nombres de las actividades cuyo tiempo libre total (TF) es igual a cero. | OK | Correcto |
| CP010 | Genera Camino | Vigilar que los nombres de las actividades críticas sean concatenados correctament | -Haber extraído y guardado en una matriz los datos de la red.<br>-Cada nodo debe tener su tiempo inicial (ET) y | Una arreglo de caracteres que tenga concatenados los nombres de las actividades | OK | Correcto |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | e dentro del arreglo resultado. | tiempo de evento tardío (LT) calculado.<br>-Haber definido cuales actividades (arcos) son consideradas como actividades críticas en base a su tiempo libre total (TF). | criticas sólo separadas por un espacio en blanco. | | |

Entonces sólo falta cumplir el requisito de guardar en un archivo ASCII. Para probar la función "GuardarArchivo" la cual nos permite guardar la matriz resultado en un archivo ASCII para cualquier uso que el usuario desee darle se consideró prudente realizar las siguientes pruebas:

| Caso de Prueba | Función a probar. | Descripción del caso | Pre requisitos | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|---|
| CP011 | Guardar Archivo | Asegurar que el archivo en donde se pretende guardar los datos es creado correctamente. | -Se debe saber el nombre para el nuevo archivo.<br>-Contar con la matriz de resultados y la sumatoria de sus costos. | Un nuevo archivo ASCII cuyo contenido contendrá las actividades que son críticas el costo total de la ruta crítica. | OK | Correcto |
| CP012 | Guardar Archivo | Verificar que sólo información de los resultados sea insertada en el archivo, sin ninguna clase de información basura. | -Contar con la matriz de resultados y la sumatoria de sus costos.<br>-La matriz de resultados debe tener su último elemento apuntando a NULO. | El contenido del archivo debe estar conformado sólo por los arcos del camino más corto y el costo total de la ruta crítica (sin la existencia de caracteres basura). | OK | Correcto |

## Conclusiones

La realización del presente proyecto tuvo como objetivo exponer algunos de los algoritmos usados en el área de investigación de operaciones (IO) para resolver problemas que pueden ser representados con un diseño de red.

El tema de los problemas con un diseño de red, involucra varios algoritmos y problemas de los cuales sólo se eligieron tres para su codificación. Los problemas elegidos fueron:

- Localizar la ruta más corta en una red.
- Formar el árbol de expansión mínima de una red.
- Determinar la ruta crítica en una red de proyecto.

Dichos problemas fueron plantearon como una red, es decir, con una estructuras donde existen dos conjuntos de símbolos conocidos como nodos y arcos. Los nodos se definieron como puntos extremos de la red y a los arcos se les definió como un posible movimiento entre dos nodos (nodo inicial y un nodo terminal). También se mencionó la existencia de algunas redes con características útiles para plantear cada problema, por ejemplo las redes dirigidas, no dirigidas y árboles.

Una vez planteados los problemas como una estructura de red, se eligieron los algoritmos que se usarían para darles solución. Estos algoritmos fueron codificados y estructurados como bibliotecas para el lenguaje de programación C, con el fin de brindar una opción documentada a los miembros de la comunidad universitaria que tenga alguna relación con la IO y que tengan interés en dar uso de algunos de estos algoritmos en algún otro proyecto.

Cada algoritmo que se eligió fue investigado en fuentes bibliográficas y electrónicas antes de su implementación, para poder apreciar su alcance, sus límites y su utilidad actual. Una vez realizada la investigación se consideró la importancia de implementar los algoritmos como bibliotecas para tener acceso a ellos de manera fácil y rápida mediante una simple llamada a función. Los algoritmos elegidos en relación a los problemas son:

- El de Floyd para el problema de localizar la ruta más corta de una red.
- El de Kruskal se usó para resolver el problema de construir el árbol de expansión mínima.
- Mientras que el de CPM (Critical Path Method) se utilizó para determinar la ruta crítica de un proyecto.

Todos los algoritmos fueron expuestos y explicados con base bibliográfica para su total comprensión y soporte.

Parte del proyecto era mostrar que existen múltiples interpretaciones que se les puede atribuir a las estructuras de redes y que dependiendo del problema que representan, los conceptos asociados a los nodos y arcos toman diferentes significados. Por ejemplo, un nodo representa un “lugar” o “localización” dentro de una red para localizar la ruta más corta, mientras que un nodo en la red de un proyecto al que se le desea determinar la ruta crítica representa un “evento” que marca la culminación de una actividad.

Todos los programas codificados con los algoritmos fueron diseñados para recibir los datos de entrada mediante un archivo de texto ASCII, en el cual los datos deben

encontrarse con un formato para que el programa pueda leerlos, extraerlos y trabajar con ellos de manera correcta. El formato de los datos debía indicar en la primera línea el número de arcos y el número de nodos y en las líneas consecuencias los datos de cada arco, su nodo inicial, nodo terminal y su precio, es ese orden. El formato de los datos dentro del archivo ASCII fue pensado para hacerlo intuitivo y fácil de entender para el usuario.

Una particularidad dentro de todos los programas codificados es que los algoritmos trabajan sólo si los datos de entrada, en este caso los datos de los arcos (nodo origen, nodo destino, costo), tienen un orden particular para cada algoritmo. Por lo que se incluyó un algoritmo de ordenamiento en cada algoritmo.

Parte de lo propuesto dentro del proyecto fue añadir la capacidad de guardar los resultados en un archivo ASCII independiente. Esta opción se implementó en el programa sin ningún problema y de manera que el usuario pueda decidir si desea guardar los resultados o no.

Una vez finalizada la codificación de los programas, a las funciones principales de cada uno de estos, se les aplicó una serie de casos de prueba (prestando principal atención a las funciones que forman cada algoritmo) para asegurar su correcto funcionamiento. Dichas pruebas fueron diseñadas también con el fin de mostrar como los algoritmos codificados cumplían con las especificaciones que se plantearon al inicio del proyecto.

Finalmente se corrigieron los problemas detectados por los casos de prueba, tras lo cual se aplicaron nuevamente estos y se consideraron correctos los programas (que incluyen los algoritmos) en virtud de los resultados obtenidos. Por lo que se da por concluido el proyecto que tenía como finalidad generar las bibliotecas documentadas en lenguaje C para su uso como herramientas de trabajo para la comunidad universitaria que tenga algún interés en aplicarlas en algún otro proyecto.

## Bibliografía


- Agüero C., Ramón (2012), Tema 4 – Algoritmos y protocolos de encaminamiento. Recuperado el 20 de enero de 2013, de http://www.tlmat.unican.es/siteadmin/submaterials/679.pdf.

- Hillier, Frederick S. y Liberman, Gerard J. (2010), *Introducción a la investigación de operaciones*. (9ª ed.). México: McGraw Hill.

- Taha, Hamdy A. (2011). *Investigación de Operaciones*, (9ª ed.) México: Pearson.

- Winston, Wayne L. (2005). *Investigación de operaciones - Aplicaciones y algoritmos*, (4ª ed.). México: CENGAGE Learning.