



## **PROYECTO FINAL AED – Administración de Red mediante Protocolos de Enrutamiento**

**Entrega Final - MÉTODO DE LA INGENIERÍA.**

**Presentado por:** Marisol Giraldo Cobo – **Código:** A00246380

**Presentado al Profesor:** Andrés Aristizábal

### **METODO DE INGENIERÍA**

#### **PASO 1. IDENTIFICACIÓN DEL PROBLEMA**

Bananas Core es una empresa líder del mercado en diversas áreas, tales como routing y switching, comunicaciones unificadas, soluciones inalámbricas y seguridad. Con el fin de conocer el funcionamiento detallado OSPF de su red, contratan una ingeniera telemática para la administración de red de la empresa. Debido al crecimiento del tamaño de la empresa, dentro de sus labores, deberá desarrollar una aplicación que reporte las subredes a las que se encuentra conectado x dispositivo (router o switch), generar reportes de conectividad que existan entre los diferentes host o dispositivos de red, mostrar sus vecinos más cercanos y la métrica de costo.

#### **PASO 2. RECOPIACIÓN DE INFORMACIÓN**

##### **ALGORITMOS Y PROTOCOLOS DE ENRUTAMIENTO**

###### **FUNCIONES BÁSICAS**

– **Forwarding o reenvío:** Un nodo (o router) determina la interfaz por la que reenviar un paquete

– **Routing o encaminamiento:** establecimiento de la ruta (camino) más apropiada entre origen y destino, actualizando las tablas de reenvío en los nodos. Se define como la operación de buscar el mejor camino posible en una red de conmutación de paquetes.

El mejor camino posible, el más corto, será el que minimice una métrica de enrutamiento. Hay diferentes tipos de métricas: número de saltos, distancia geográfica, retardo promedio, ancho de banda, nivel de tráfico o una combinación de varias.

###### **ELEMENTOS Y CLASIFICACIÓN**

- ✦ **Métricas de rendimiento**
  - # de saltos
  - Coste
  - Retardo
  - Rendimiento (Throughput)
  
- ✦ **Fuentes de información**
  - Local
  - Nodo contiguo (vecino)
  - Nodos de la ruta
  - Todos los nodos

- ✦ **Lugar de la decisión**
  - Cada nodo
  - Nodo central
  - Nodo origen/fuente
  
- ✦ **Actualización de la información**
  - Continuo
  - Periódico
  - Cambio en la carga
  - Cambio topológico

## TÉCNICAS DE ENRUTAMIENTO

### ✦ ENRUTAMIENTO LOCAL

Esta técnica no tiene en cuenta la topología de la red y utiliza únicamente información local.

### ✦ ENRUTAMIENTO ESTÁTICO

Esta técnica tiene en cuenta la topología de la red para encaminar los paquetes. Para ello se necesita una tabla de enrutamiento donde guarda información de cada ruta. Generalmente esa información consiste en el destino, la máscara o prefijo de red, siguiente salto y costo asociado al camino.

### ✦ ENRUTAMIENTO DINÁMICO

Esta técnica además de tener en cuenta la topología de red construye las tablas de enrutamiento de manera automática, mediante el intercambio de información constante entre los enrutadores de la red. Se adapta automáticamente a los cambios en la topología de la red.

#### ○ **Protocolo de Vector-Distancia**

Cada enrutador tiene una tabla de enrutamiento con una entrada por cada posible destino de la red.

Para construir la tabla de enrutamiento los nodos intercambian periódicamente información con sus vecinos. Cada nodo envía a sus vecinos distancia que conoce, estos actualizan su tabla con esta distancia sumada a la distancia del vecino. Este método también se conoce como Algoritmo Bellman-Ford.

Por ejemplo, los protocolos que más utilizan este algoritmo son:

- **RIP, RIPv2 y RIPng**, los cuales utilizan como métrica el número de saltos, como un máximo de 1555 saltos
- **IGRP** utiliza como métrica compuesta basándose en el ancho de banda, retardo y carga del enlace.
- **BGP** es un protocolo tipo EGP, y no utiliza métricas, sino que toma decisiones de enrutamiento basándose en políticas de red.

#### ○ **Protocolos de Estado- Enlace**

Cada enrutamiento mantiene una base de datos con información sobre la topología exacta de la red.

Para construir esta base de datos, se sigue un proceso similar al definido a continuación: Cada enrutador identifica a sus vecinos y su distancia, después anuncia esta información a todos los nodos de la red por inundación. Al recibir esta información, cada nodo la guarda en la base de datos y la difunde junto a su propia información. Al final todos los nodos de la red tendrá la información de todos. Entonces, basándose en el **Algoritmo Dijkstra**, cada enrutador crea un mapa o árbol de rutas con el mismo como origen.

Los protocolos de Estado- Enlace más utilizados son:

- **OSPF, OSPFv2 y OSPFv3**, donde los nodos que lo utilizan descubren la topología de la red mediante el intercambio coordinado de unidades de información llamadas LSA. Los **LSA de tipo Router Link** son creados por cada router del sistema y definen la relación del propio router con sus interfaces. Los **LSA de tipo Network Link** son creados por el DR de cada subred y definen la relación entre la subred y todos los vecinos conectados a ella.

Mediante estos tipos de LSA, y utilizando el algoritmo de Dijkstra, cada router crea lo que se conoce como el **Shortest-Path Tree** poniéndose a sí mismo como raíz.

De esta manera cada router conoce la ruta más óptima hacia todos los integrantes de la red.

Finalmente, se actualiza la tabla de rutas del sistema indicando, principalmente, para cada ruta: el destino, el siguiente salto y el coste total.

## GRAFO

En matemáticas y ciencias de la computación, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto. Un grafo puede verse como una pareja de conjuntos de  $G = (V, A)$  donde  $V$  es el conjunto de vértices y  $A$  es un conjunto de aristas.

Muchas redes de uso cotidiano pueden ser modeladas con un grafo: una red de carreteras, una red eléctrica, una red de drenaje entre otros.

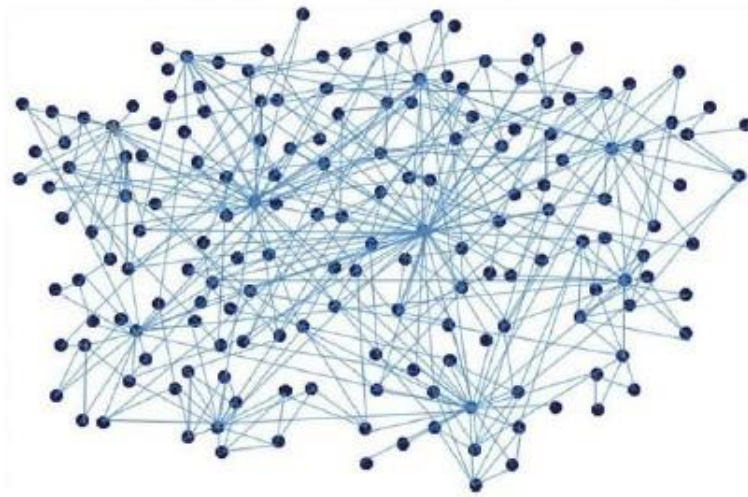


Ilustración 1 Grafo

## TIPOS DE GRAFOS

- ✦ **Dirigidos:** Es un par  $G = (V, E)$ 
  - Los enlaces son pares dirigidos de nodos
  - $V$  es un conjunto finito de Vértices (o Nodos o Puntos)
  - $E$  es un conjunto de Aristas (o Arcos) dirigidas

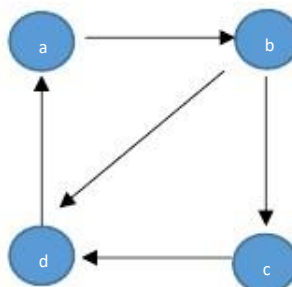


Ilustración 2 Grafo Dirigido

Aquel grafo dirigido en el que sus aristas tienen una etiqueta puede ser un nombre, costo o un valor de cualquier tipo de dato se le llama **Grafo Dirigido con Peso**. También a este grafo se le denomina red de actividades, y el número asociado al arco se le denomina factor de peso. Se usa en el modelado de problemas de la vida real; por ejemplo, al tiempo que se tardará en realizar una actividad determinada o la distancia que hay de un lugar a otro.

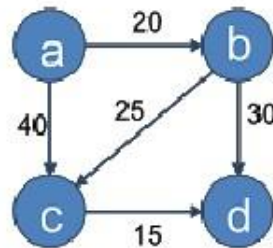


Ilustración 3 Grafo Ponderado

- ✦ **No dirigidos:** Es un par  $G = (V, E)$ 
  - No es necesario establecer un criterio de ordenación a los nodos en cada enlace
  - V es un conjunto finito de Vértices
  - E es un conjunto de Aristas no Dirigidas

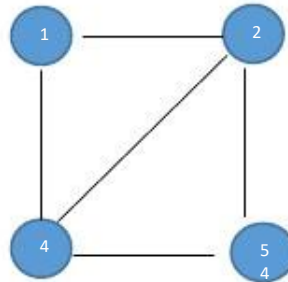


Ilustración 4 Grafo No Dirigido

## CONCATENACIONES DE ENLACES

- ✦ **Path (camino):** Una secuencia de vértices dentro de un grafo tal que exista una arista entre cada vértice y el siguiente. Se dice que dos vértices están conectados si existe un camino que vaya de uno a otro, de lo contrario estarán desconectados. Dos vértices pueden estar conectados por varios caminos. El número de aristas dentro de un camino es su longitud. Así, los vértices adyacentes están conectados por un camino de longitud 1, y los segundos vecinos por un camino de longitud 2.  
Por ejemplo: Camino desde a hasta d  $\rightarrow \langle a, b, e, c, d \rangle$ .

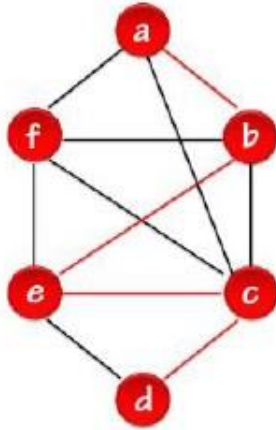


Ilustración 5 Camino

**Longitud de un camino:** Es el número de arcos del camino.  
 Por ejemplo: longitud del camino desde a hasta d  $\rightarrow \langle a, b, e, c, d \rangle$  es 4.

Se habla también de **camino hamiltoniano** si no se impone regresar al punto de partida.

✦ **Cycle (ciclo o circuito):** Es un camino que empieza y acaba en el mismo vértice.

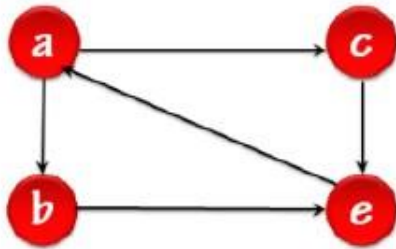


Ilustración 6 Ciclo

Por ejemplo:  $\langle a, c, e, a \rangle$  es un ciclo de longitud 3

Un **ciclo hamiltoniano** tiene además que recorrer todos los vértices exactamente una vez (excepto el vértice del que parte y al cual llega).

✦ **Bucle:** Un bucle es una arista que conecta a un vértice consigo mismo. Es un ciclo de longitud 1.

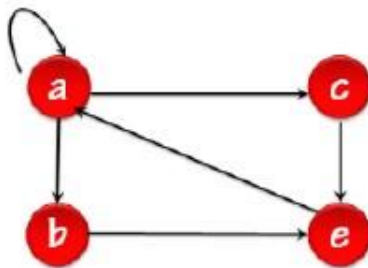


Ilustración 7 Bucle

Existen diferentes formas de almacenar grafos en una computadora. La estructura de datos usada depende de las características del grafo y el algoritmo usado para manipularlo. Entre las estructuras más sencillas y usadas se encuentran las listas y las matrices, aunque frecuentemente se usa una combinación de ambas. Las listas son preferidas en grafos dispersos porque tienen un eficiente uso de la memoria. Por otro lado, las matrices proveen acceso rápido, pero pueden consumir grandes cantidades de memoria.

## ESTRUCTURA DE LISTA

- ✦ **Lista de Incidencia:** Las aristas son representadas con un vector de pares (ordenados, si el grafo es dirigido), donde cada par representa una de las aristas.
- ✦ **Lista de Adyacencia:** Cada vértice tiene una lista de vértices los cuales son adyacentes a él. Esto causa redundancia en un grafo no dirigido (ya que A existe en la lista de adyacencia de B y viceversa), pero las búsquedas son más rápidas, al costo de almacenamiento extra.

## ESTRUCTURAS MATRICIALES

- ✦ **Matriz de incidencia:** El grafo está representado por una matriz de A (aristas) por V (vértices), donde [arista, vértice] contiene la información de la arista (1 - conectado, 0 – no conectado).
- ✦ **Matriz de adyacencia:** El grafo está representado por una matriz cuadrada M de tamaño  $n^2$ , donde n es el número de vértices. Si hay una arista entre un vértice x y un vértice y, entonces el elemento  $m_{x,y}$  es 1, de lo contrario, es 0.

## ALGORITMOS DE GRAFOS

### ✦ BFS - BREADTH FIRST SEARCH

En Ciencias de la Computación, Búsqueda en anchura es un algoritmo para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles). Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

Formalmente, BFS es un algoritmo de búsqueda sin información, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución. El algoritmo no usa ninguna estrategia heurística.

### ✦ DFS - DEPTH FIRST SEARCH

Una Búsqueda en profundidad un algoritmo que permite recorrer todos los nodos de un grafo o árbol (teoría de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

## ✦ 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 los vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos 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 coste 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).

Una de sus aplicaciones más importantes reside en el campo de la telemática, gracias a él, podemos resolver grafos con muchos nodos, los cuales serían muy complicados de hacer sin dicho algoritmo, encontrando así las rutas más cortas entre un origen y todos los destinos en una red.

## ✦ FLOYD-WARSHALL

Es un algoritmo de análisis sobre grafos que permite encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución, constituyendo un ejemplo de programación dinámica.

El algoritmo obtiene la mejor ruta entre todo par de nodos. Trabaja con la matriz D inicializada con las distancias directas entre todo par de nodos. La iteración se produce sobre nodos intermedios, o sea para todo elemento de la matriz se prueba si lo mejor para ir de  $i$  a  $j$  es a través de un nodo intermedio elegido o como estaba anteriormente, y esto se prueba con todos los nodos de la red. Una vez probados todos los nodos de la red como nodos intermedios, la matriz resultante da la mejor distancia entre todo par de nodos.

El algoritmo da sólo la menor distancia; se debe manejar información adicional para encontrar tablas de encaminamiento. Hasta no hallar la última matriz no se encuentran las distancias mínimas. Su complejidad es del orden  $n^3$ .

## ✦ BELLMAN-FORD

El algoritmo de Bellman-Ford (algoritmo de Bell-End-Ford) genera el camino más corto en un grafo dirigido ponderado (en el que el peso de alguna de las aristas puede ser negativo). El algoritmo de Dijkstra resuelve este mismo problema en un tiempo menor, pero requiere que los pesos de las aristas no sean negativos, salvo que el grafo sea dirigido y sin ciclos. Por lo que el Algoritmo Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo. Este algoritmo fue desarrollado por Richard Bellman, Samuel End y Lester Ford.

## ✦ KRUSKAL

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor total de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa). El algoritmo de Kruskal es un ejemplo de algoritmo voraz.

Un ejemplo de árbol expandido mínimo. Cada punto representa un vértice, el cual puede ser un árbol por sí mismo. Se usa el Algoritmo para buscar las distancias más cortas (árbol expandido) que conectan todos los puntos o vértices. Funciona de la siguiente manera:

Se crea un bosque B (un conjunto de árboles), donde cada vértice del grafo es un árbol separado se crea un conjunto C que contenga a todas las aristas del grafo mientras C es no vacío eliminar una arista de peso mínimo de C si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol en caso contrario, se desecha la arista Al acabar el algoritmo, el bosque tiene un solo componente, el cual forma un árbol de expansión mínimo del grafo.

#### ★ PRIM

El algoritmo incrementa continuamente el tamaño de un árbol, comenzando por un vértice inicial al que se le van agregando sucesivamente vértices cuya distancia a los anteriores es mínima. Esto significa que, en cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol. El árbol recubridor mínimo está completamente construido cuando no quedan más vértices por agregar.

### PASO 3. BÚSQUEDA DE SOLUCIONES CREATIVAS

#### SOLUCIÓN 1:

Mediante una plataforma tercerizada de gestión de red poder administrar y supervisar eficazmente la infraestructura de TI. Esta plataforma proporcionaría una amplia gama de herramientas para monitorear redes, servidores y centros de datos, y para la identificación de problemas de desempeño. Ver los dispositivos de red, servidores, entornos virtuales, en la nube e inalámbricos en contexto. Si bien tercerizar permite que la empresa se enfoque en las actividades vitales para el agregado de valor del negocio, existe el riesgo de pérdida de control de gestión y administración de la empresa, posibles problemas de comunicación con el prestador del servicio, amenaza a la confidencialidad y a la seguridad de la información interna de la empresa además del costo general (mantenimiento, disponibilidad, licencia entre otros.).

#### SOLUCIÓN 2:

En la contratación de servicios podría contratarse a personal técnico que se encargue de rastrear y monitorear la red con hardware y software especializado. Este personal podría ser propio y/o tercerizado, sin embargo, en cualquiera de las dos opciones el costo general (mantenimiento, disponibilidad, licencia entre otros) persistiría, por lo que sería necesario considerar un análisis costo-beneficio para evitar costos ocultos.

#### SOLUCIÓN 3:

La Ingeniera Telemática contratada, mediante los conocimientos adquiridos en su carrera profesional generará una aplicación java aplicando modelos basados en la teoría de grafos que pueden aplicarse a optimización de tráfico en redes y análisis de redes de comunicaciones.

Los routers y computadores serían considerados, teóricamente, como nodos, y las líneas de transmisión, como los enlaces. De modo que la red se considere matemáticamente como un conjunto de nodos y enlaces de un grafo  $G = (V, E)$ , donde  $V$  es el conjunto de nodos y  $E$  es el conjunto de los enlaces.

Al trasladar un grafo a un problema de enrutamiento:

- Los  $N$  nodos son los routers de la red
- Los  $E$  enlaces se corresponden con los enlaces físicos entre ellos

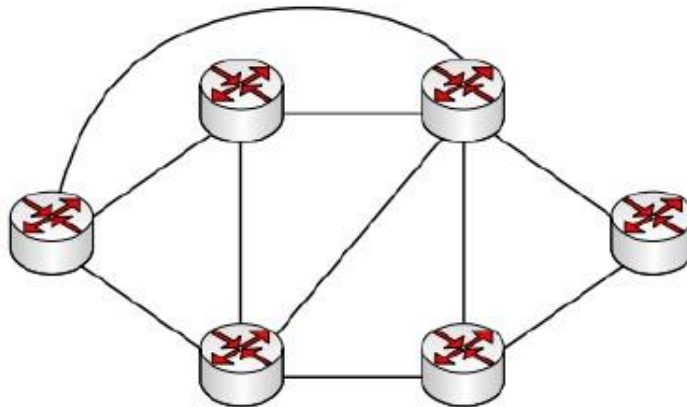


Ilustración 8 Red computacional



Cada router posee información global sobre la red: nodos y enlaces existentes

### ¿Cómo?

- ✦ Informan de sus enlaces a redes activas y con routers vecinos.
- ✦ Inundan la red con esta información.
- ✦ Todos los routers tienen una imagen (grafo) de la red. A partir de ella eligen los caminos.

Los routers son dispositivos que conectan los diferentes segmentos que conforman una red IP, seleccionando un camino para que los paquetes lleguen a su dirección de destino. Los routers actúan en la capa de red del modelo OSI por lo cual tienen acceso a la información de direccionamiento de los nodos, esta información es utilizada para determinar el camino óptimo entre el punto de origen y el punto de destino de un mensaje. Además, los routers guardan y dirigen los paquetes a los segmentos interconectados de la red sin preocuparse por su topología.

Los routers utilizan un algoritmo de enrutamiento para decidir el camino de los datos transmitidos por la red, basándose en la información contenida en una tabla de rutas y en el encabezado del paquete donde se ubica la dirección fuente y la de destino. La tabla de rutas lista las direcciones la red y los caminos entre los nodos. El algoritmo de enrutamiento, entonces, compara la información del encabezado y las direcciones de red en la tabla de rutas para determinar el mejor camino hacia el cual enviar el paquete de información.

## BÚSQUEDA DEL CAMINO MÁS CORTO

La idea principal es la de encontrar el camino con un coste mínimo entre una fuente (Source) y un destino (Destination).

Si  $G(V, E)$  se mantiene constante para todos los enlaces, la solución es la ruta de menor número de saltos.

- ✦ Algoritmos con una única fuente: encuentran el camino más corto entre S y el resto de los nodos.
  - Dijkstra
  - Bellman-Ford
- ✦ Algoritmos para toda la red: encuentran el camino más corto entre todas las posibles parejas de nodos en la red.
  - Floyd-Warshall
  - Johnson

## PASO 4. PASAR DE LA IDEA PRINCIPAL AL DISEÑO PRELIMINAR

**tipo** Grafo

**dominios** Grafo, Elemento, BOOLEAN

**generadores**

Crear :  $\rightarrow$  Grafo

Añadir\_Nodo : Grafo  $\times$  Elemento  $\rightarrow$  Grafo

Añadir\_Arista : Grafo  $\times$  Elemento  $\times$  Elemento  $\rightarrow$  Grafo

**constructores**

Borrar\_Nodo : Grafo  $\times$  Elemento  $\rightarrow$  Grafo

Borrar\_Arista : Grafo  $\times$  Elemento  $\times$  Elemento  $\rightarrow$  Grafo

**selectores**

Es\_Vacio? : Grafo  $\rightarrow$  BOOLEAN

Contiene? : Grafo  $\times$  Elemento  $\rightarrow$  BOOLEAN

Son\_Adyacentes?: Grafo  $\times$  Elemento  $\times$  Elemento  $\rightarrow$  BOOLEAN

Nodo: Grafo  $\rightarrow$  Elemento

Arista: Grafo  $\rightarrow$  Elemento  $\times$  Elemento

**auxiliares**

Hay\_Arista? : Grafo  $\rightarrow$  BOOLEAN

**precondiciones** G: Grafo n, m: Elemento

pre: Añadir\_Arista (G, n, m) = Contiene? (G, n) AND Contiene? (G, m)

pre: Nodo(G) = NOT Es\_Vacio? (G)

pre: Arista(G) = Hay\_Arista?(G)

**ecuaciones** G: Grafo; n,m,p,q: Elemento;

```
Añadir_Nodo(Añadir_Nodo(G, n), m) == SI (n = m) ENTONCES
    Añadir_Nodo(G, n);
SI NO
    Añadir_Nodo(Añadir_Nodo(G, m), n)
Añadir_Arista(G, m, n) == Añadir_Arista(G, n, m)
Añadir_Arista(Añadir_Arista(G, m, n), p, q) ==
    SI (p = m) AND (q = n) ENTONCES
        Añadir_Arista(G, m, n)
    SI NO
        Añadir_Arista(Añadir_Arista(G, p, q), m, n)
Añadir_Nodo(Añadir_Arista(G, m, n), p) == Añadir_Arista(Añadir_Nodo(G, p), m, n)
Borrar_Nodo(Crear, m) == Crear
Borrar_Nodo(Añadir_Nodo(G, n), m) == SI (n = m) ENTONCES
    Borrar_Nodo(G, m)
    SI NO
        Añadir_Nodo(Borrar_Nodo(G, m), n)
Borrar_Nodo(Añadir_Arista(G, p, q), m) == SI (m = p) OR (m = q) ENTONCES
    Borrar_Nodo(G, m)
    SI NO
        Añadir_Arista(Borrar_Nodo(G, m), p, q)
Borrar_Arista(Crear, n, m) == Crear

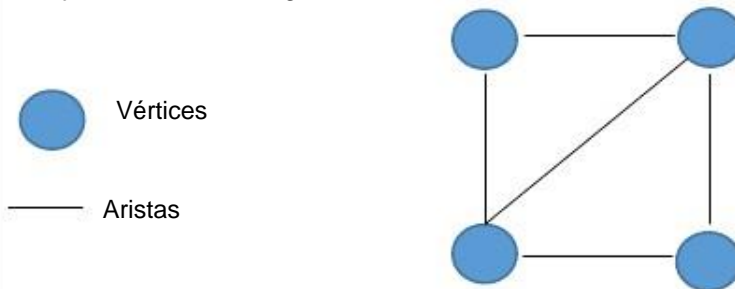
Borrar_Arista(Añadir_Nodo(G, p), n, m) == Añadir_Nodo(Borrar_Arista(G, n, m), p)
Borrar_Arista(Añadir_Arista(G, p, q), n, m) ==
    SI (n = p AND m = q) OR (n = q AND m = p) ENTONCES
        Borrar_Arista(G, n, m)
    SI NO
        Añadir_Arista(Borrar_Arista(G, n, m), p, q)
Es_Vacio?(Crear) == VERDADERO
Es_Vacio?(Añadir_Nodo(G, n)) == FALSO
Es_Vacio?(Añadir_Arista(G, n, m)) == FALSO
Hay_Arista?(Crear) == FALSO
Hay_Arista?(Añadir_Nodo(G, n)) == Hay_Arista?(G)
Hay_Arista?(Añadir_Arista(G, n, m)) == VERDADERO
Contiene?(Crear, m) == FALSO
Contiene?(Añadir_Nodo(G, n), m) == (n = m) OR Contiene?(G, m)
Contiene?(Añadir_Arista(G, p, q), m) == Contiene?(G, m)
Son_Adyacentes?(Crear, n, m) == FALSO
Son_Adyacentes?(Añadir_Nodo(G, p), n, m) ==
Son_Adyacentes?(G, n, m)
Son_Adyacentes?(Añadir_Arista(G, p, q), n, m) == (n = p AND m = q) OR
    (n = q AND m = p) OR
    Son_Adyacentes?(G, n, m)
Contiene?(G, Nodo(G)) == VERDADERO
Son_Adyacentes(G, Arista(G)) = VERDADERO
```

**fin**

## TAD GRAFO

### Representación

Sea  $G = (V, E)$  donde  $V = \{V_1, V_2, V_3 \dots V\}$  es el conjunto de vértices y  $E = \{E_1, E_2, E_3 \dots E_n\}$  es el conjunto de aristas del grafo.



### Invariante

Debe tener al menos un vértice

### Operaciones Primitivas

Nombre	Entradas	Salidas	Operación
createGraph		→ Graph	Constructor
insert	Vertex	→ Graph	Modificador
delete	Vertex	→ Graph	Modificador
isEmpty	Graph	→ Booleano	Analizador
isAdjacent	Vertex	→ setAdjacent	Analizador
vertex	Graph	→ setVertex	Analizador
edges	Graph	→ setEdges	Analizador

## PASO 5. EVALUACION Y SELECCIÓN DE LA SOLUCIÓN

Se escogieron varios ítems para evaluar las soluciones:

- Costo
- Conocimiento
- Eficiencia
- Garantía de información de la empresa

Cada uno de estos ítems se evalúa en una escala que es:

- 0 -> Alto impacto negativo
- 1 -> Poco impacto negativo
- 2 -> Poco impacto positivo
- 3 -> Alto impacto positivo

Con base a lo anterior se hace la siguiente tabla:


Idea / Ítem	Costo	Conocimiento	Eficiencia	Garantía de la Información	Total
<b>Solución 1</b>	0	1	3	1	5
<b>Solución 2</b>	0	2	1	1	4
<b>Solución 3</b>	3	3	3	3	12

Por puntuación, se escoge la **SOLUCIÓN 3**, es decir una aplicación java que aplicando modelos de teorías de grafos permita administrar la red de la empresa Banana Cores implementada por la Ingeniería Telemática contratada. En cada ítem, esta opción tiene una relación costo- beneficio viable, puesto que el costo de la aplicación hace parte del contrato laboral pactado con la Ingeniera Telemática, en conocimiento sería una profesional que estaría aportando sus habilidades y competencias a los objetivos de la empresa, sería eficiente pues sería una herramienta automatizada y la información sería vigilada dentro de la misma organización.

La opción de la **SOLUCIÓN 1**, si bien es eficiente genera unos costos adicionales importantes y no aporta ningún valor de habilidad y/o competencia para los empleados de la empresa, además de los cuidados de ceder información propia a un tercero.

La opción de la **Solución 2**, no es tan eficiente pues requiere contratación de un equipo técnico adicional que realice el monitoreo de la red de forma manual, esto generaría un aporte relevante en conocimiento a la empresa, sin embargo, el costo adicional por cada empleado del equipo técnico y la garantía de la información al ser accedida por miembros tercerizados, convierten esta opción en una solución inviable.

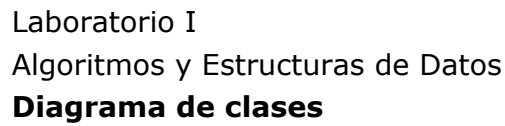
## PASO 6. PREPARACIÓN DE INFORMES Y ESPECIFICACIONES

	Laboratorio I Algoritmos y Estructuras de Datos <b>Requerimientos Funcionales</b>
Nombre del Proyecto:	<b>– Administración de Red mediante Protocolos de Enrutamiento</b>
Presentado por:	Marisol Giraldo Cobo – Código: A00246380
Presentado a:	Andrés Aristizábal
Fecha:	Mayo 17 del 2020

<b>Nombre</b>	<b>R1 – Ingresar información</b>
<b>Resumen</b>	La aplicación debe permitir ingresar la información de la infraestructura de red
<b>Entradas</b>	
- Un archivo de texto con el formato de la red	
<b>Resultados</b>	
- La infraestructura visualizada en una GUI	


<b>Nombre</b>	<b>R2 – Generar reporte de subredes</b>
<b>Resumen</b>	La aplicación debe permitir generar un formato con las subredes (Nombre, No de dispositivos, Categorías de los dispositivos, IP de los dispositivos) que contiene un router en cada una de sus interfaces de red.
<b>Entradas</b>	
- El host solicitado	
<b>Resultados</b>	
- Un archivo con un formato para las subredes	

<b>Nombre</b>	<b>R3 – Generar ruta más optima</b>
<b>Resumen</b>	La aplicación debe permitir generar un reporte de la ruta óptima de un paquete ICMP entre dos hosts, si la red fue configurada con el protocolo de enrutamiento OSPF.
<b>Entradas</b>	
- Los dos hosts seleccionados	
<b>Resultados</b>	
- Los dispositivos que hay en su ruta	

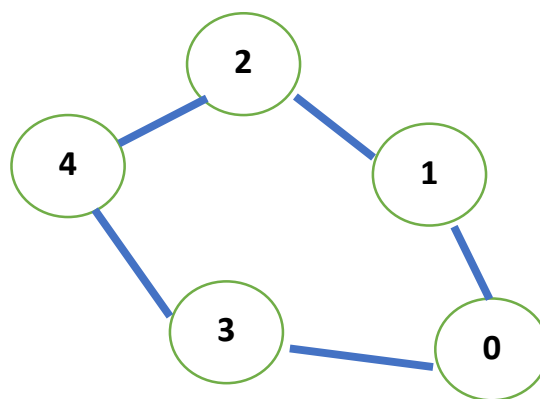


Nombre del Proyecto:	<b>– Administración de Red mediante Protocolos de Enrutamiento</b>
Presentado por:	Marisol Giraldo Cobo – Código: A00246380
Presentado a:	Andrés Aristizábal
Fecha:	Mayo 17 del 2020



	Laboratorio I Algoritmos y Estructuras de Datos <b>Diseño de Pruebas</b>
Nombre del Proyecto:	<b>– Administración de Red mediante Protocolos de Enrutamiento</b>
Presentado por:	Marisol Giraldo Cobo – Código: A00246380
Presentado a:	Andrés Aristizábal
Fecha:	Mayo 17 del 2020

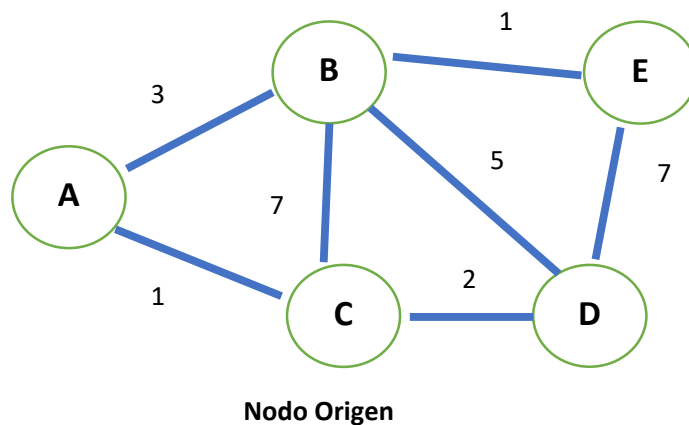
### GRAFO DE PRUEBA No 1



<b>Clase</b>	<b>Principal</b>
<b>Método</b>	BFSAdjacencyMatrix
<b>Caso de Prueba</b>	
1	
<b>Objetivo</b>	
Probar que el método retorna una búsqueda de los nodos por medio del recorrido de anchura	
<b>Entradas</b>	
NodoDeLaRed y grafo de prueba No 1	
<b>Salidas</b>	
Una cadena con el siguiente recorrido: 2, 4, 1, 3, 0	
<b>Procedimiento</b>	
La estrategia que usaremos para garantizar este recorrido es utilizar una cola que nos permita almacenar temporalmente todos los nodos de un nivel, para ser procesados antes de pasar al siguiente nivel hasta que la cola esté vacía.	

<b>Clase</b>	<b>Principal</b>
<b>Método</b>	DFSAdjacencyMatrix
<b>Caso de Prueba</b>	
2	
<b>Objetivo</b>	
Probar que el método retorna una búsqueda de los nodos por medio del recorrido por profundidad	
<b>Entradas</b>	
NodoDeLaRed, un arreglo de nodos visitados que está vacío y grafo de prueba No 1	
<b>Salidas</b>	
Una cadena con el siguiente recorrido: 2, 4, 3, 0	
<b>Procedimiento</b>	
La estrategia que usaremos para garantizar este recorrido es recorrer desde la raíz hasta los nodos extremos u hojas por cada una de las ramas. En este caso los niveles de cada nodo no son importantes.	

## GRAFO DE PRUEBA No 2



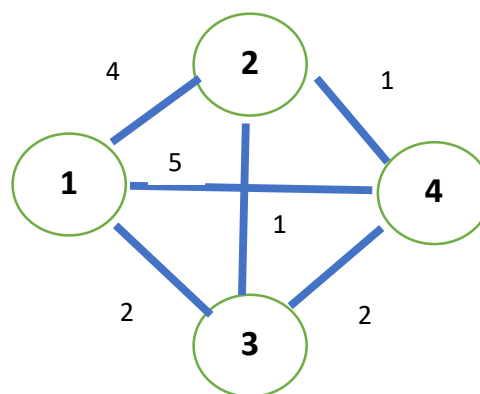
<b>Clase</b>	<b>Principal</b>
<b>Método</b>	DijkstraAdjacencyMatrix
<b>Caso de Prueba</b>	
3	
<b>Objetivo</b>	
Probar que el método retorna la ruta más corta desde un nodo origen hacia los demás nodos.	
<b>Entradas</b>	
NodoDeLaRed y grafo de prueba No 2	
<b>Salidas</b>	
Una cadena con el siguiente recorrido: C->C=0, C -> A = 1, C -> D= 2, C ->B = 4, C->E =5	
<b>Procedimiento</b>	



Durante la ejecución del algoritmo, iremos marcando cada nodo con su **distancia mínima** al nodo C (nuestro nodo elegido). Para el nodo C, esta distancia es 0. Para el resto de los nodos, como todavía no conocemos esa distancia mínima, empieza siendo infinita ( $\infty$ ): También tendremos un **nodo actual**. Inicialmente, el nodo actual será C (nuestro nodo elegido).

1. Se marca el nodo inicial que elegiste con una distancia actual de 0 y el resto con infinito.
2. Se establece el nodo no visitado con la menor distancia actual como el nodo actual A.
3. Para cada vecino  $\nabla$  de tu nodo actual A: suma la distancia actual de A con el peso de la arista que conecta a A con  $\nabla$ . Si el resultado es menor que la distancia actual de  $\nabla$ , establéclo como la nueva distancia actual de  $\nabla$ .
4. Marca el nodo actual A como visitado.
5. Si hay nodos no visitados, ve al paso 2.

### GRAFO DE PRUEBA No 3

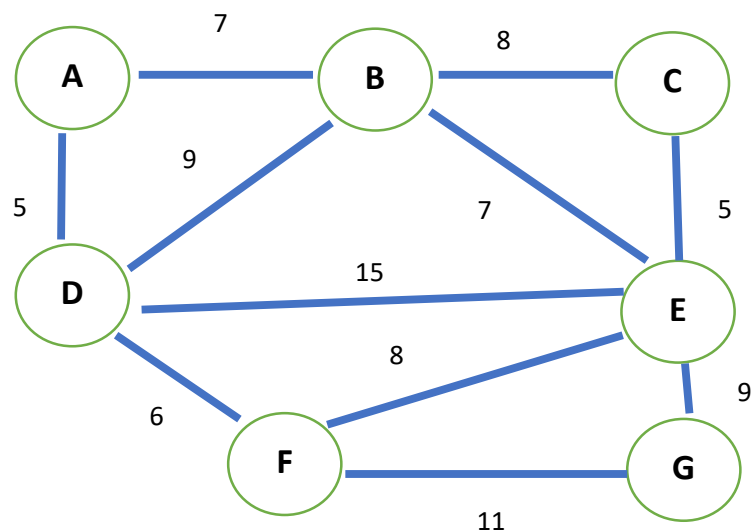


Clase	Principal																									
Método	FloydWarshallAdjacencyMatrix()																									
Caso de Prueba																										
4																										
Objetivo																										
Probar que el método compara todos los posibles caminos entre cada par de nodos. Esto se consigue al ir mejorando un estimado de la distancia entre dos nodos, hasta que el estimado es óptimo.																										
Entradas																										
No necesita entrada. Utiliza toda la estructura de datos. Grafo de Prueba No 3																										
Salidas																										
Una matriz:																										
<table><tr><td></td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>-</td><td>2</td><td>3</td><td>4</td></tr><tr><td>2</td><td>1</td><td>-</td><td>1</td><td>4</td></tr><tr><td>3</td><td>1</td><td>2</td><td>-</td><td>4</td></tr><tr><td>4</td><td>1</td><td>2</td><td>3</td><td>-</td></tr></table>			1	2	3	4	1	-	2	3	4	2	1	-	1	4	3	1	2	-	4	4	1	2	3	-
	1	2	3	4																						
1	-	2	3	4																						
2	1	-	1	4																						
3	1	2	-	4																						
4	1	2	3	-																						

### Procedimiento

La estrategia que usaremos para garantizar este recorrido es: A partir de una tabla inicial compuesta de 0's (no hay correspondencia inicial en el grafo) y 1's (hay una correspondencia, llamase "flecha", entre nodos), obtiene una nueva matriz denominada "Matriz de Clausura Transitiva" en la que se muestran todas las posibles uniones entre nodos, directa o indirectamente. Es decir, si de "A" a "B" no hay una "flecha", es posible que si haya de "A" a "C" y luego de "C" a "B". Luego, este resultado se verá volcado en la matriz final.

### GRAFO DE PRUEBA No 4

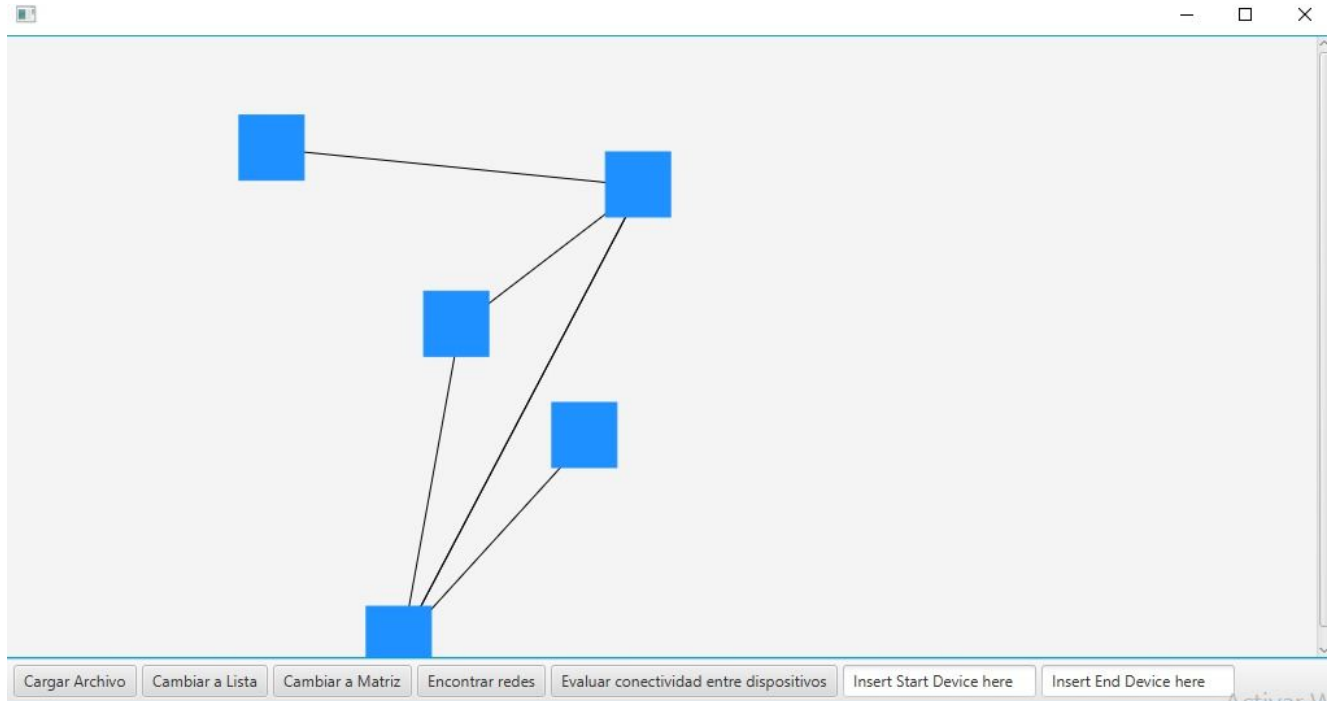


Clase	Principal
Método	PrimAdjacencyMatrix
Caso de Prueba	
5	
Objetivo	
Probar que el método retorna un árbol de expansión mínima. Es decir, es capaz de encontrar un subconjunto de las aristas que formen un árbol que incluya todos los vértices del grafo inicial, donde el peso total de las aristas del árbol es el mínimo posible.	
Entradas	
No necesita entrada. Utiliza toda la estructura de datos. Grafo de Prueba No 4	
Salidas	
Una cadena del árbol de expansión mínima con un peso de 39.	
Procedimiento	
<ol style="list-style-type: none"><li>1. Se marca un vértice cualquiera. Será el vértice de partida.</li><li>2. Se selecciona la arista de menor peso incidente en el vértice seleccionado anteriormente y se selecciona el otro vértice en el que incide dicha arista.</li></ol>	

3. Repetir el paso 2 siempre que la arista elegida enlace un vértice seleccionado y otro que no lo esté. Es decir, siempre que la arista elegida no cree ningún ciclo.
4. El árbol de expansión mínima será encontrado cuando hayan sido seleccionados todos los vértices del grafo.

<b>Clase</b>	<b>Principal</b>
<b>Método</b>	KruskalAdjacencyMatrix
<b>Caso de Prueba</b>	
6	
<b>Objetivo</b>	
Probar que el método retorna un árbol de expansión mínima. Es decir, es capaz de encontrar un subconjunto de las aristas que formen un árbol que incluya todos los vértices del grafo inicial, donde el peso total de las aristas del árbol es el mínimo posible.	
<b>Entradas</b>	
No necesita entrada. Utiliza toda la estructura de datos. Grafo de Prueba No 4	
<b>Salidas</b>	
Una cadena del árbol de expansión mínima con un peso de 39.	
<b>Procedimiento</b>	
<ol style="list-style-type: none"> <li>1. Se selecciona, de entre todas las aristas restantes, la de menor peso siempre que no cree ningún ciclo.</li> <li>2. Se repite el paso 1 hasta que se hayan seleccionado <math> V  - 1</math> aristas. Siendo V el número de vértices.</li> </ol>	

## PASO 7. IMPLEMENTACIÒN DEL DISEÑO



## **BIBLIOGRAFÍA**

### **ALGORITMOS DE ENCAMINAMIENTO**

Recuperado de: [https://ocw.unican.es/pluginfile.php/301/course/section/239/tema\\_02.pdf](https://ocw.unican.es/pluginfile.php/301/course/section/239/tema_02.pdf)

### **IMPLEMENTA UN GRAFO DE CIUDADES EN JAVA**

Recuperado de: <https://devs4j.com/2017/11/24/implementa-un-grafo-de-ciudades-en-java/>

### **CLASE 2: CAMINOS, CIRCUITOS EULERIANOS Y HAMILTONIANOS**

Recuperado de:

[https://www.icesi.edu.co/moodle/Grafos\\_Caminos\\_y\\_conectividad%20\(1\).pdf](https://www.icesi.edu.co/moodle/Grafos_Caminos_y_conectividad%20(1).pdf)

### **PROTOCOLOS DE ENRUTAMIENTO SIMULADOR DE TRÁFICO DE REDES**

Recuperado de: <https://www.monografias.com/trabajos-pdf/enrutamiento-trafico-redes/enrutamiento-trafico-redes2.shtml>

### **TEORIA DE GRAFOS**

Recuperado de: <https://medium.com/@matematicasdiscretaslibro/cap%C3%ADtulo-11-teoria-de-grafos-3b00228dd81c>

### **GRAFOS**

Recuperado de: <https://w3.ual.es/~btorreci/tr-grafos.pdf>

### **ALGORITMOS DE RESOLUCIÓN DE PROBLEMAS DE CAMINOS MÍNIMOS EN GRAFOS**

Recuperado de: <http://knuth.uca.es/moodle/mod/page/view.php?id=4108&forceview=1>