UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Redes


# Laboratorio 3 - Primera parte

# Algoritmos de Enrutamiento

MANUEL ALEJANDRO ARCHILA MORAN 161250

DIEGO JOSE FRANCO PACAY 20240

JUAN DIEGO AVILA SAGASTUME 20090

**Descripción de la práctica**

El objetivo de la práctica era la elaboración e implementación de tres algoritmos de enrutamiento, los cuales fueron: Flooding, Link State Routing y Distance Vector, para el envió de mensajes. Para cada uno de estos algoritmos se realizó una clase que simulará el comportamiento y organizará el traspaso de mensajes.

El primer algoritmo que se realizó fue Flooding. Este algoritmo consiste en enviar el mensaje desde el nodo inicial a todos sus vecinos. Los nodos vecinos realizan el mismo proceso de enviar el mensaje a todos sus nodos vecinos, así sucesivamente hasta llegar al nodo receptor. Cuando el mensaje alcanza el nodo deseado el mensaje se presenta en pantalla.

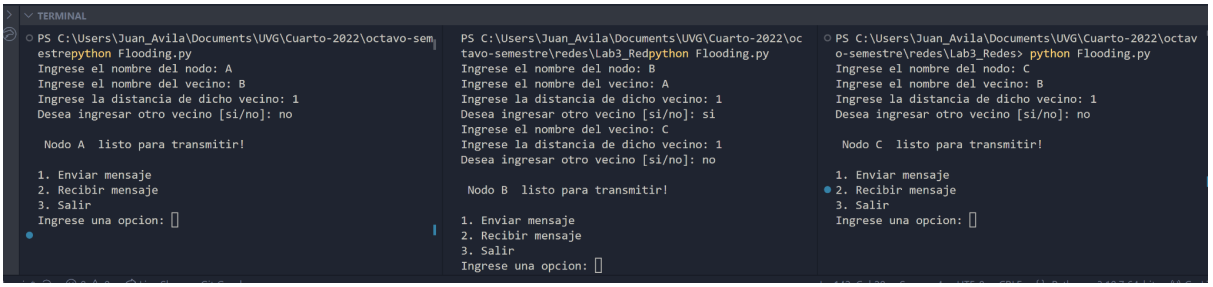
El segundo algoritmo implementado fue Link State Routing. Este algoritmo para funcionar correctamente necesita conocer la topología de la red. Es por esto que al empezar el algoritmo se solicita el nombre de todos los nodos y las distancias entre dichos nodos. Luego para enviar el mensaje, el nodo emisor calcula el mejor camino del nodo actual al receptor usando el algoritmo de Dijkstra. Este nodo actual se encarga de enviar el mensaje al nodo más cercano que se encuentra en el camino óptimo. Como siguiente paso, el nodo que acaba de recibir el mensaje hace este cálculo nuevamente para poder realizar el envío al siguiente nodo en el camino, así sucesivamente hasta llegar al destinatario y poder mostrar el payload.

Por último, se realizó el algoritmo de Distance Vector. Este algoritmo se basa en el intercambio continuo de información entre nodos vecinos para determinar las rutas óptimas en una red. Al iniciar el algoritmo, cada nodo posee una tabla que contiene la distancia estimada hacia otros nodos en la red, junto con el próximo salto necesario para alcanzar esos destinos. Cuando un nodo detecta cambios en su tabla de distancia o en la información recibida de nodos vecinos, actualiza su tabla y envía copias actualizadas a sus nodos vecinos. Estos nodos, a su vez, repiten el proceso, lo que lleva a una propagación de la información de enrutamiento en toda la red. Si el nodo que está recibiendo la información detecta que su tabla de enrutamiento ya no se modifica a pesar de recibir información de sus vecinos debe de indicar que ya converge. Este algoritmo tiene la peculiaridad de que es necesario que todos los nodos indiquen su convergencia para poder empezar a enviar y recibir mensajes. Para este proceso el nodo emisor consulta su tabla de distancia para determinar el próximo salto hacia el destino deseado. Este enlace se elige en función de las distancias y los saltos almacenados en la tabla. El mensaje se envía al nodo seleccionado, y este proceso se repite a medida que el mensaje se dirige hacia su destinatario final.

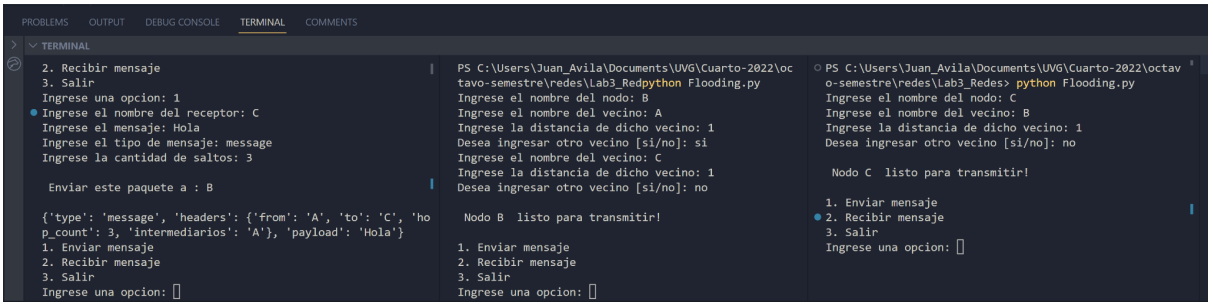
**Resultados**

Flooding:

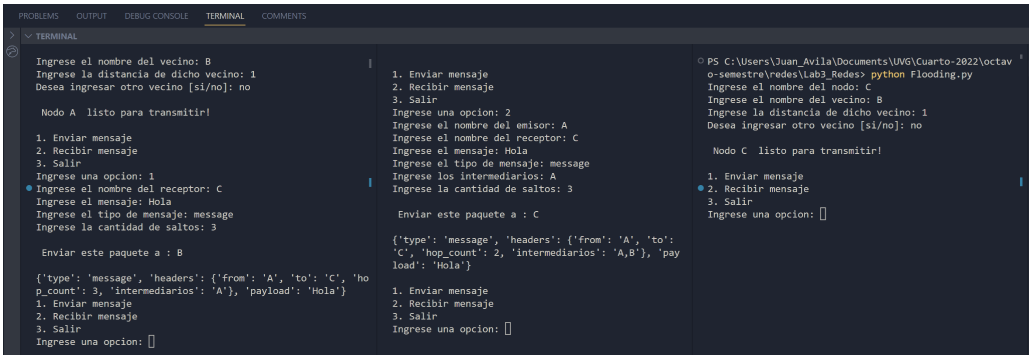
Instancia de nodos




Envío de mensaje de A a C:



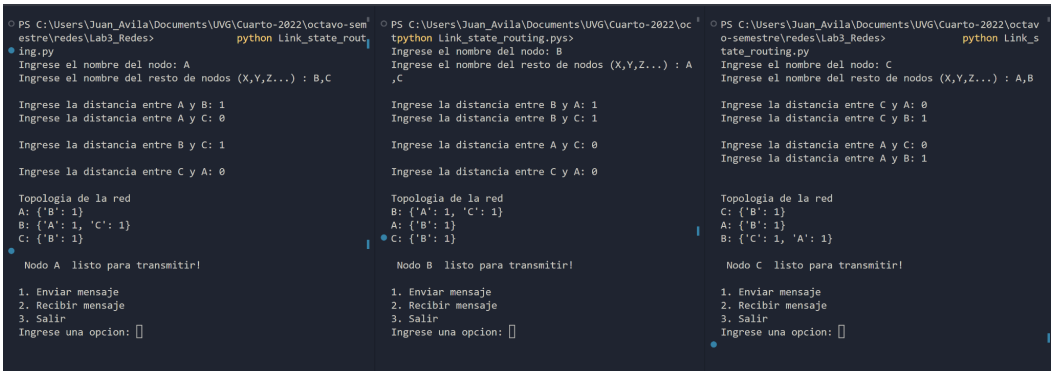

Recibiendo mensaje con B para transmitir:



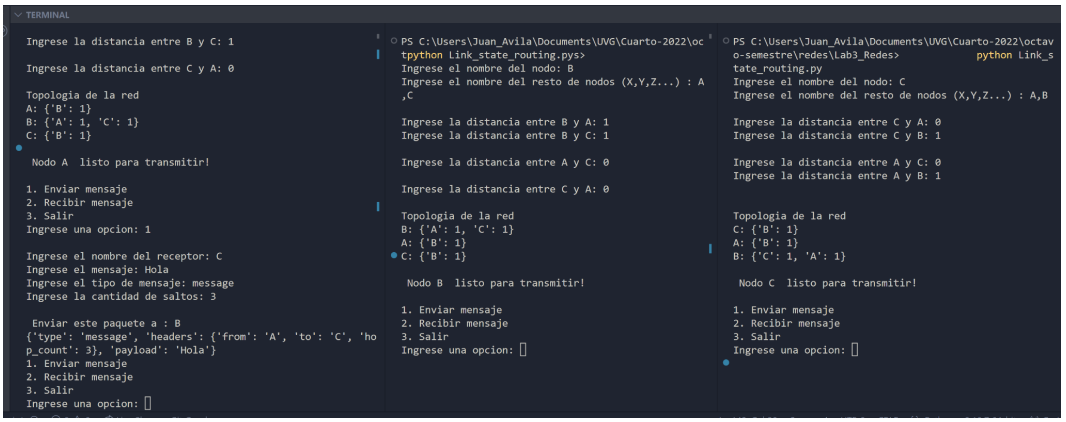

Recibiendo el mensaje en el nodo C:


## Link State Routing

Creación de la topología de red e instancia de nodos:

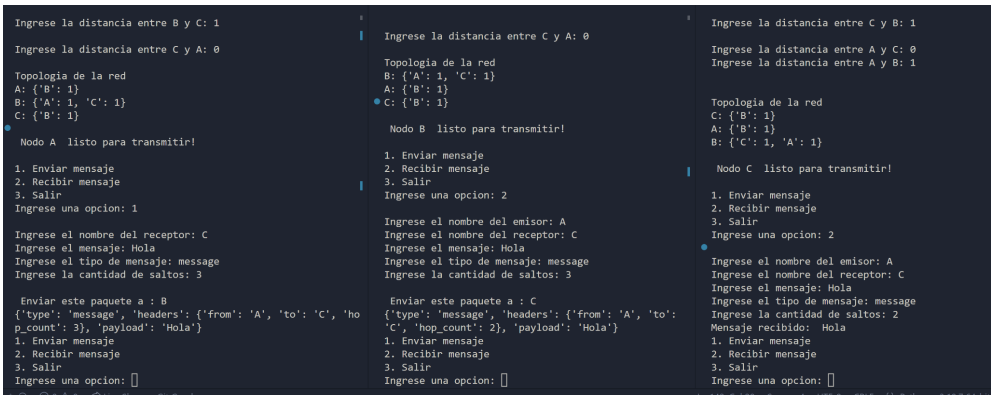



Envío de mensaje desde el nodo A al nodo C:



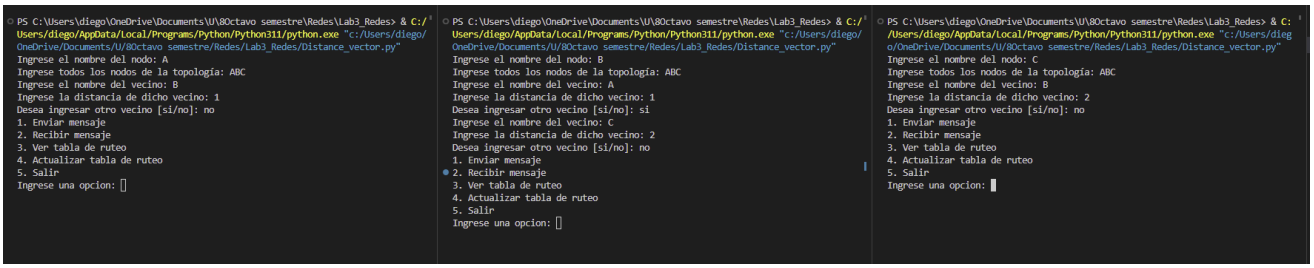

Recibiendo mensaje desde B para transmitir:


Recibiendo mensaje desde C:

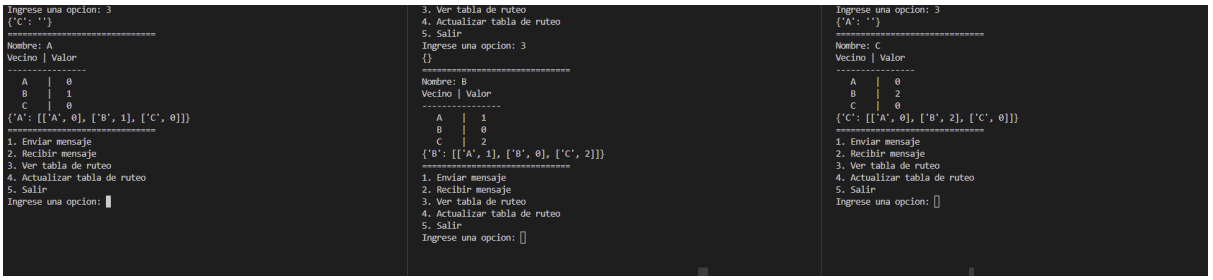



## Distance Vector:

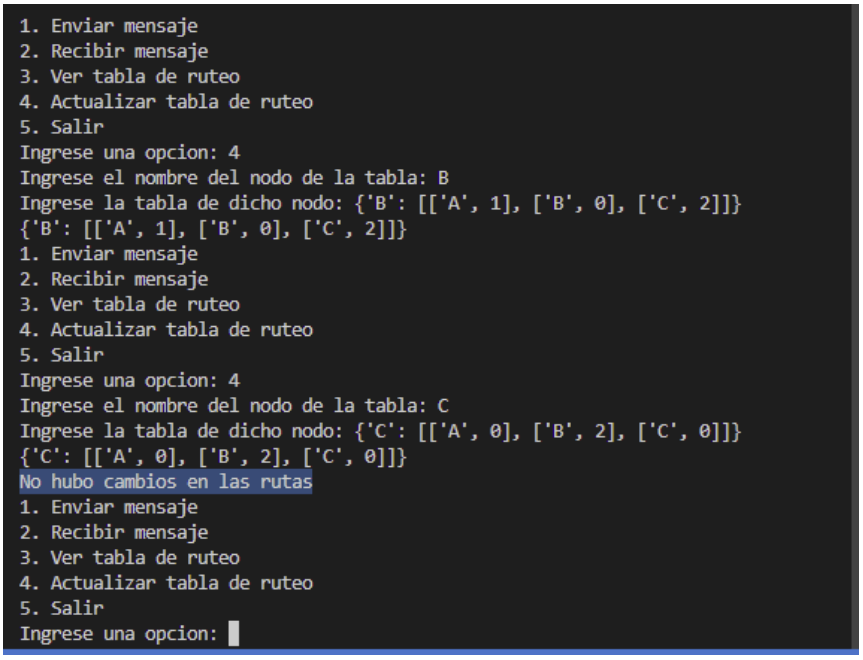
Instanciar los nodos:



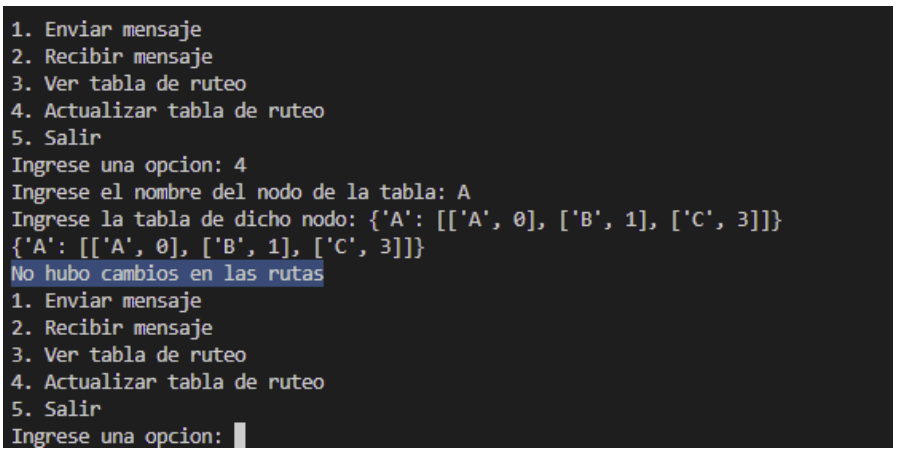
Imprimir tabla de ruteo para cada nodo:




Actualizar la tabla de A hasta que ya no hayan cambios:



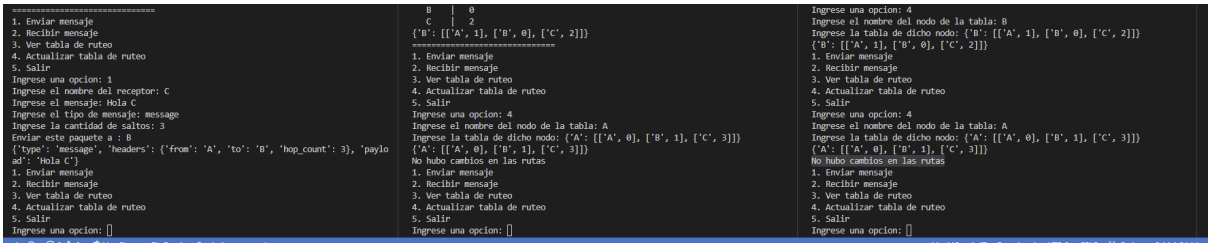

Actualizar la tabla de B hasta que ya no hayan cambios:



Actualizar la tabla de C hasta que ya no hayan cambios:

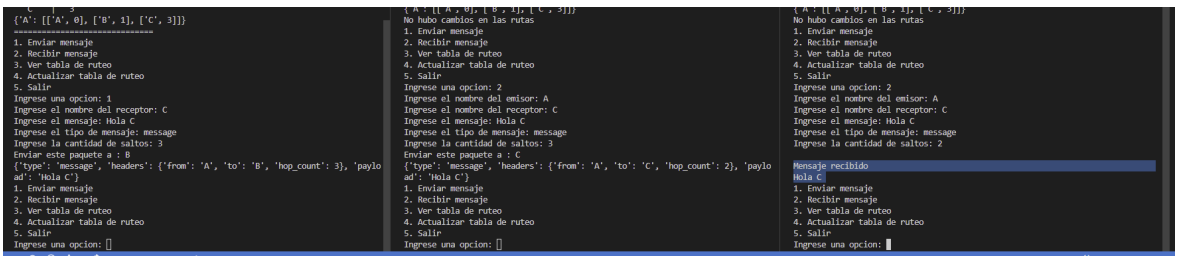
```
============================
1. Enviar mensaje
2. Recibir mensaje
3. Ver tabla de ruteo
4. Actualizar tabla de ruteo
5. Salir
Ingrese una opcion: 4
Ingrese el nombre del nodo de la tabla: B
Ingrese la tabla de dicho nodo: {'B': [['A', 1], ['B', 0], ['C', 2]]}
{'B': [['A', 1], ['B', 0], ['C', 2]]}
1. Enviar mensaje
2. Recibir mensaje
3. Ver tabla de ruteo
4. Actualizar tabla de ruteo
5. Salir
Ingrese una opcion: 4
Ingrese el nombre del nodo de la tabla: A
Ingrese la tabla de dicho nodo: {'A': [['A', 0], ['B', 1], ['C', 3]]}
{'A': [['A', 0], ['B', 1], ['C', 3]]}
No hubo cambios en las rutas
1. Enviar mensaje
2. Recibir mensaje
3. Ver tabla de ruteo
4. Actualizar tabla de ruteo
5. Salir
Ingrese una opcion:
```

Envío de mensaje de A a C:



Recibiendo mensaje con B para transmitir:

Recibiendo el mensaje en el nodo C:



**Discusión**

Para empezar la práctica se empezó con el algoritmo de Flooding. Este aparentaba ser el algoritmo más sencillo debido a que no se necesita ningún tipo de información acerca de la topología de la red. Sin embargo, al momento de probar con diferentes topologías de red, se encontraron algunos errores que hacían que la transmisión de mensajes no funcionara como se deseaba. Esto se debe a que en el momento de experimentación se presenciaban bucles infinitos, ya que cada uno de los nodos enviaban el paquete a todos sus vecinos y como no se verifica si un nodo ya recibió el paquete se hacía infinitamente o hasta que se acabara el hop count. Por esto se decidió implementar un nuevo parámetro a la estructura del paquete, el cual era un arreglo con los nodos visitados por el paquete. De esta manera el paquete llevaba un registro de los nodos visitados, sabiendo a qué nodos no había llegado el mensaje para transmitir la información. Este problema de creación infinita de paquetes es uno de los problemas que se presenta en el algoritmo. Es por esto que no se recomienda el uso de Flooding para comunicación de mensajes uno a uno. A pesar de estas complicaciones se pudo realizar correctamente el envío de mensajes usando este algoritmo. (Herramientas Web, 2023)

El segundo algoritmo que se implementó fue el Link State Routing. Para este algoritmo era necesario conocer la topología de la red, presentando un poco más de complejidad. Además, al igual que Flooding a cada nodo creado se le proporcionaba también la distancia a sus vecinos. Esto con el fin de tener datos con el cual se podría usar el algoritmo de Dijkstra para armar correctamente las tablas de enrutamiento. Al tener las tablas correctamente armadas, el algoritmo pudo encontrar la ruta más corta y enviar el mensaje sin problemas.

Finalmente, se realizó el algoritmo de Distance Vector. Este algoritmo era el que más información necesitaba, ya que aparte de recibir la topología completa, los nodos y sus vecinos junto con sus distancias, se necesitaba el envió del vector de distancia de cada uno de los nodos. Esto implicaba una etapa de reconocimiento en el que cada nodo se iba actualizando con cada una de la información que recibía, hasta que todos los nodos converjan y no se pueda actualizar más las rutas. Distance Vector resultó ser más difícil que cualquier otra implementación debido al cálculo de las rutas que hacía el programa. Al investigar acerca de posibles acercamientos, se encontró que se podía implementar el algoritmo de Bellman-Ford. Al utilizar este otro algoritmo en nuestro programa los mensajes se pudieron enviar de forma correcta.

## Comentario grupal

Es muy interesante ver como aunque hacen lo mismo, es decir, se están transmitiendo mensajes entre nodos, hay tantos acercamientos diferentes según la necesidad de la red como se puede ver con los algoritmos implementados. Algunos desean enviar el mensaje a través de la mejor ruta, otros simplemente desean hacer llegar un mensaje a un receptor sin importar por quien pase y esto define no sólo somo se envían los mensajes sino afectan la implementación completa del algoritmo, calculando para algunos la topología completa para cada nodo o teniendo en cuenta únicamente a los vecinos.

Aunque el más complejo de implementar fue el distance vector, para los tres algoritmos tuvimos que tomar en cuenta muchas consideraciones a parte de comprender el algoritmo ya que hay casos que se deben tomar en cuenta y se deben prevenir para el correcto funcionamiento y aplicación del mismo.

## Conclusiones

El algoritmo Flooding, a pesar de ser bien sencillo al basarse en el envío indiscriminado de mensajes a todos los nodos vecinos, lo que garantiza la entrega eventual, pero puede dar lugar a un uso ineficiente del ancho de banda, además de que puede causar problemas a la hora de toparse con loops infinitos.

Por otro lado, el enrutamiento mediante Link State Routing requería un conocimiento previo de la topología de la red, lo que implicaba un paso inicial de recopilación de información. A pesar de esto, el uso del algoritmo de Dijkstra para calcular caminos óptimos permitió una transmisión más eficiente al elegir rutas específicas en función de las distancias.

Para el algoritmo de distance vector el cual fue el más complicado de implementar, la actualización de tablas de ruteo hasta que cada uno de los nodos haya convergido puede significar una demora en el envío de los mensajes pero una vez todas las tablas estén actualizadas no hay forma de que el envío del paquete se pierda en algún loop al cual no debe de entrar.

Por lo tanto cada uno de los algoritmos utilizados en este laboratorio poseen distintos enfoques que traen consigo ventajas y desventajas específicas para las situaciones que se están buscando. Haciendo que la elección del algoritmo adecuado dependería de las necesidades y características de la red en cuestión.

**Referencias**

Herramientas Web. (2023). Inundación. Retrieved August 30, 2023, from Lcc.uma.es website: https://neo.lcc.uma.es/evirtual/cdd/tutorial/red/inunda.html#:~:text=El%20principal%20inconveniente%20que%20plantea,establece%20alguna%20forma%20de%20limitaci%C3%B3n.