Computer network project

Diaz S, Álvarez S y Velandia N. Macc

***Abstract* — Link-state algorithms are complex routing protocols that share information with different routers to create a comprehensive overview of the overall computer network. By doing so, these protocols can fully describe the possible routes with their respective costs, along said network. One of many link-state algorithms is the renowned Dijkstra's Algorithm, which computes the least-cost path from one node (the source) to all other nodes in the network in an iterative fashion.**

The focus of this research paper is to study link-state routing protocols and with this information design a program, with the aid of Python, that can calculate the shortest path based on Dijkstra's Algorithm of a network (a random graph) composed of a maximum of 50 nodes.

***Index Terms* — Link-state Algorithm, Routing protocols, Computer Network, Least-cost path, Dijkstra's Algorithm**

I. INTRODUCCIÓN

CON la revolución y el desarrollo de internet han surgido diversos problemas a lo largo del crecimiento de la red. En los últimos años la red ha crecido mucho más rápido de lo que lo hacía antes, por lo que cada día se necesita que navegar en la red sea un proceso más fácil y rápido. De aquí nace el problema de enrutamiento, en el que se necesita que un enrutador acceda a los servidores de la manera más ágil posible. Para llegar al servidor no solo existe una ruta posible sin que hay distintas con diferentes tiempos y costos de conexión. Por lo tanto el problema a tratar se basa en encontrar la ruta más corta hacia distintos nodos de la red, para generar una posible mejoría en la calidad de servicio que puedan prestar distintos ISP(Internet service provider).

Durante el desarrollo de este escrito, se tiene el propósito de presentar un proyecto para la materia redes de computadores, el cual consiste en programar un algoritmo que sea capaz de generar grafos de manera aleatoria o mediante un archivo CSV. En los grafos generados, los nodos representan los enrutadores, y las aristas sus conexiones con sus respectivos costos.

II. ESTADO DEL ARTE

Grafo:

El estudio de los grafos ha dado solución a múltiples problemas tales como problema de rutas, que es exactamente el mismo problema al cual se enfrenta este proyecto. Para estos problemas existen varias soluciones, pero la más común es el uso de algoritmos, y hay varios para este problema:

-Dijkstra
-Floyd warshall
-Bellman-Ford
-Kruskal
-Prim
-Ford-Fulkerson

Algoritmo de Dijkstra:

También llamado el algoritmo de caminos mínimos, consiste en ir explorando los caminos más cortos, e ir almacenando los datos en una tabla, posteriormente se compara esta tabla y con esta se puede hallar el camino más corto desde un nodo origen hacia el resto de los nodos, también dice el peso total de cada camino escogido. Es un algoritmo bastante eficiente y confiable, por lo que se vuelve la mejor opción a la hora de buscar la ruta más corta en un grafo.

Grafos en Python:

En Python existe una librería llamada "Networkx", que sirve para el manejo de grafos. Esta librería nos permite crear grafos con la función "nx.Graph()" y Dígrafos con la función "nx.DiGraph()"adicionalmente la función "add_nodes_from()" o "add_edges_from()" permiten agregar nodos y aristas respectivamente, por lo cual el trabajo de creación de un grafo desde cero está solucionado, lo que facilita realizar el enfoque en otros aspectos de la problemática.

Matrices en Python:

A lo largo del desarrollo se tiende a tener el problema del uso de matrices en el código. Python no es un lenguaje que se lleve muy bien con estas al no estar bien definidas ni son trabajables. Por tanto, como primera solución se pensó en usar listas cuyo contenido fueran otras listas. Esta idea fue descartada dado que ya existe una librería especializada en trabajar con matrices llamada "numpy", con lo cual el trabajo de usar matrices también se facilitó.

Lectura de archivos csv:

La lectura de un archivo CSV se puede realizar también desde la librería "NetworkX", dado que desde la función "read_edge_list()" se puede leer el archivo. De esta forma se minimiza el trabajo de buscar librerías especializadas en la lectura de archivos CSV.

.

III. METODOLOGIA

Para una mejor comprensión de todo el seudocódigo que hay por detrás del planteamiento y solución del problema a desarrollar, utilizar diagramas de flujo puede clarificar mejor el proceso. La idea general es la siguiente.

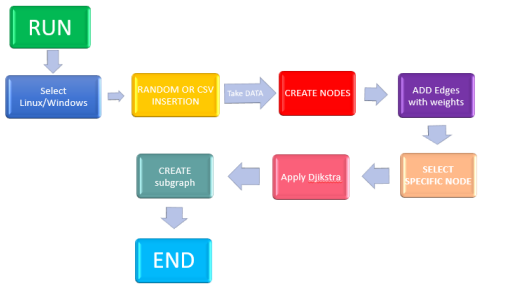

Figura 1. Proceso general.

Como se puede apreciar en la imagen anterior. El proceso no es tan complejo. Es únicamente sacar datos, plantear un problema mediante grafos y luego ejecutar un algoritmo de caminos mínimos para hallar las rutas de menor costo. Para los procesos de creación de nodos, se explica mejor con el siguiente esquema.

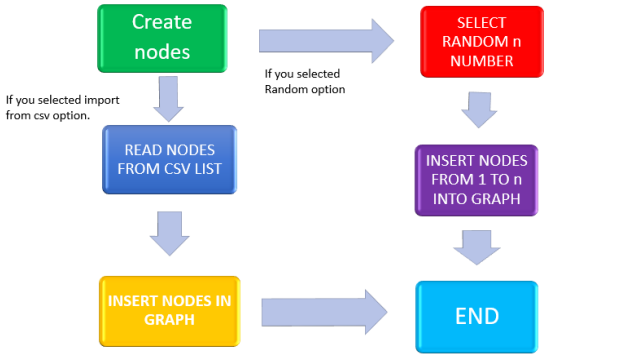

Figura 2. Creación de nodos.

En la figura 2, simplemente se generan los nodos que van a representar nuestros enrutadores dentro del grafo. Si se escoge la opción de aleatoriedad, se genera un numero n aleatorio y luego se crean los nodos del 1 al n. Si de lo contrario se escogió datos mediante CSV, entonces toma los nodos ingresados en la matriz de dispersión.

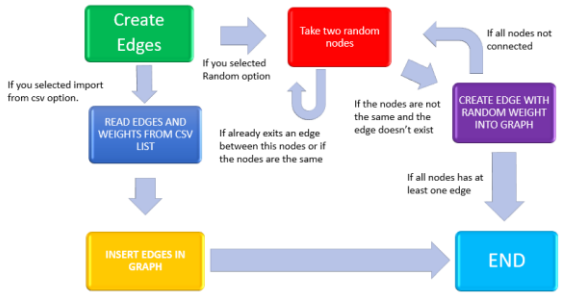

Figura 3. Creación de enlaces.

Note que, al plantear una red de redes en la vida real, se puede encontrar con la problemática de la posibilidad de que haya nodos (routers) aislados, es decir si ningún tipo de conexión hacia la red principal. Esto es algo que se quiere evitar por lo cual el diagrama de la figura 3, perfectamente soluciona este inconveniente.

Dado que, en el proceso general, previamente se tuvo que escoger si los datos eran aleatorios o importados mediante CSV, el programa debe tener en cuenta estas 2 alternativas. Si son importados mediante un CSV, cae toda la responsabilidad del programador en si ocurren este tipo de situaciones de nodos aislados, mientras que, si se ha incurrido a la opción de generación aleatoria, simplemente se van añadiendo aristas con extremos y costos aleatorios hasta que no haya nodos aislados. Si se da el caso donde los nodos ya tienen un enlace entre sí, o si los nodos escogidos son el mismo, es decir, se hace un bucle, simplemente no se agrega.

IV. RESULTADOS Y DISCUSIÓN

Los tiempos de ejecución del programa son relativamente cortos, incluso cuando se aumenta el número de nodos el tiempo de ejecución casi no varía, y si se incrementa el peso en las conexiones el tiempo de ejecución se mantiene igual.

En general los gráficos son fáciles de interpretar siempre y cuando el número de nodos sea bajo, pero si el número de nodos aumenta, el grafo va a ser más difícil de interpretar. Ahora un posible aspecto a mejorar en trabajos futuros sería el hecho de hacer el algoritmo iterativo.

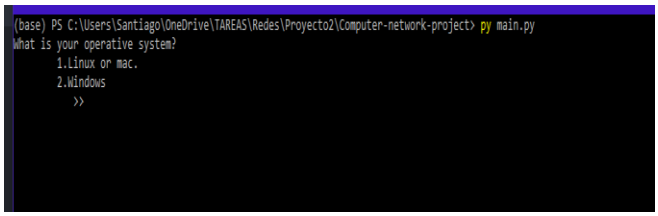

Figura 4. Ejecución del programa

Como se puede observar en la figura 4, al iniciar, el programa solicita al usuario ingresar la opción en la cual escoge que sistema operativo está corriendo. Esto es únicamente con fines de evitar usar comandos no permitidos debido a que se está trabajando con la consola.

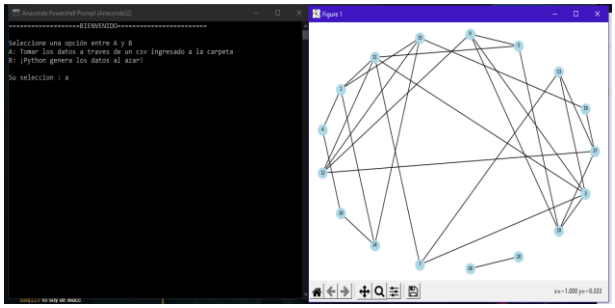

Figura 5. Opciones permitidas y grafo del problema.[1]

Luego en la figura 5 se puede apreciar como el programa brinda dos opciones para continuar con el desarrollo de la problemática. En la opción "A" se tiene la posibilidad de ingresar un archivo CSV. En este se debe tener tres columnas. La primera es del nodo de origen, la segunda el destino y en la última el costo de utilizar ese enlace. Básicamente seria un matriz de dispersión. En la opción "B" Python se encargará de todo, desde generar nodos hasta los enlaces con sus pesos. Luego el código arroja una visualización del grafo.

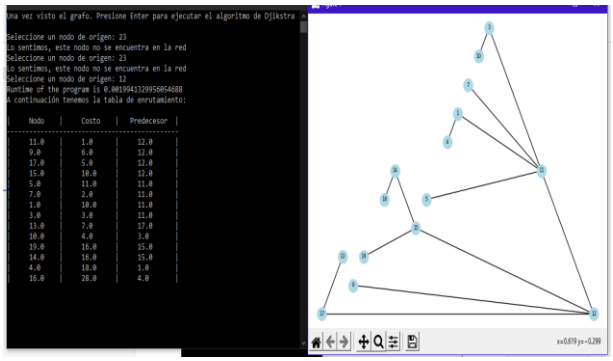

Figura 6. Tabla de enrutamiento.[2]

Una vez se cierra la ventana del grafo, viene la parte de encontrar las rutas de menor costo. Para esto el programa solicita un nodo de origen, luego aplica un algoritmo de Dijkstra a dicho nodo para posteriormente hacer un análisis de lo ocurrido. Como se puede ver en la figura 6, la consola da información como el tiempo tomado por la ejecución del algoritmo, la tabla de enrutamiento[3] de ese nodo y por último una imagen de un árbol de expansión que nos permite visualizar aquellas rutas de menor costo desde el nodo escogido hacia los demás enrutadores.

V. CONCLUSIONES

Los tiempos de ejecución del programa son relativamente cortos, incluso cuando se aumenta el número de nodos el tiempo de ejecución casi no varía, y si se incrementa el peso en las conexiones el tiempo de ejecución se mantiene igual.

En general los gráficos son fáciles de interpretar siempre y cuando el número de nodos sea bajo, pero si el número de nodos aumenta, el grafo va a ser más difícil de interpretar.

Ahora un posible aspecto a mejorar en trabajos futuros sería el hecho de hacer el algoritmo iterativo.

VI. REFERENCIAS

- *Computer networking, a top down approach 7th edition*
- *Admministrator. (2021,November 9). Link State Routing Protocols. Firewall.cx. https://www.firewall.cx/networking-topics/routing/rounting-protocols/183-link-state-routing.html*
- *Dehghan, S., & Rahmani, A. M. (2010). A new intelligent link state routing algorithm based on Credit Base-CMAC neural network. 2010 IEEE 12th International Conference on Communication Technology. https://doi.org/10.1109/icct.2010.56886*

[1] Las opciones se ingresan simplemente con una "A" o "B". No se permiten otros símbolos, el programa no discrimina entre mayúsculas y minúsculas.

[2] Hay que destacar que en esta parte se debe ingresar un enter vacío a la consola previo y luego ingresar un nodo valido.

[3] La tabla de enrutamiento posee 3 columnas. La primera es nodo destino, la segunda es cuanto cuesta llegar y la tercera es desde que nodo se precede