Nombre: Josue Salvador Cano Martinez ***Matrícula:*** A00829022

Programación de estructuras de datos y algoritmos fundamentales

Módulo: Estructura de Datos de Red (Grafos)

Actividad 4.3: Actividad integral de Grafos (Evidencia de competencia)

Investigación y reflexión individual

Fecha de entrega: 18 de noviembre de 2020

Objetivo

Realizar una investigación y reflexión en forma individual de la importancia y eficiencia del uso de Grafos en una situación problema de esta naturaleza.

Parte 1: Investigación

Grafos

Un gráfico (figura 1.0) es una estructura de datos no lineal que consta de nodos y bordes. Los nodos a veces también se denominan vértices y los bordes son líneas o arcos que conectan dos nodos cualesquiera en el gráfico.

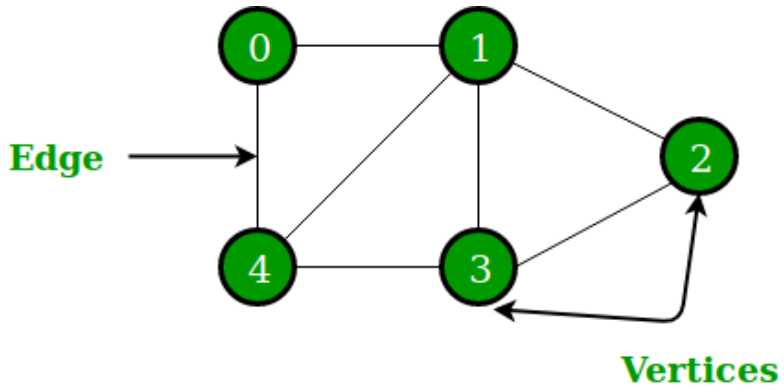


Figura 1.0: representación gráfica de un grafo.

Los gráficos se utilizan para representar redes. Las redes pueden incluir trayectos en una ciudad o una red telefónica o una red de circuitos. Los gráficos también se utilizan en redes sociales como LinkedIn, Facebook. [1]

Representaciones

Matriz de adyacencia:

La matriz de adyacencia (figura 2.0) es una matriz 2D de tamaño V x V donde V es el número de vértices en un gráfico. Sea la matriz 2D adj [] [], una ranura adj [i] [j] = 1 indica que hay un borde desde el vértice i al vértice j. La matriz de adyacencia para grafos no dirigidos es siempre simétrica. La Matriz de adyacencia también se usa para representar gráficos ponderados. Si adj [i] [j] = w, entonces hay una arista desde el vértice i al vértice j con peso w.

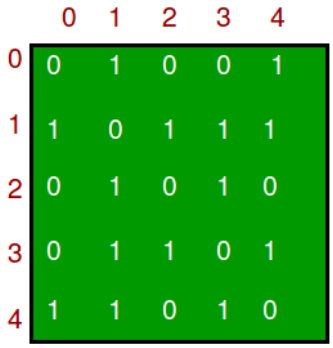


Figura 2.0: representación gráfica de una matriz de adyacencia.

Ventajas: La representación es más fácil de implementar y seguir. La eliminación de un borde lleva O (1) tiempo. Consultas como si hay un borde desde el vértice 'u' al vértice 'v' son eficientes y se pueden hacer O (1).
Contras: Consume más espacio O(V^2). Incluso si el gráfico es escaso (contiene menos aristas), consume el mismo espacio. Agregar un vértice es el tiempo O(V^2). Consulte esto para ver una implementación de Python de muestra de la matriz de adyacencia.

Lista de adyacencia:

Se utiliza una matriz de listas (figura 3.0). El tamaño de la matriz es igual al número de vértices. Sea la matriz una matriz []. Una matriz de entrada [i] representa la lista de vértices adyacentes al i-ésimo vértice. Esta representación también se puede utilizar para representar un gráfico ponderado. Los pesos de las aristas se pueden representar como listas de pares. A continuación, se muestra la representación de la lista de adyacencia del gráfico anterior.

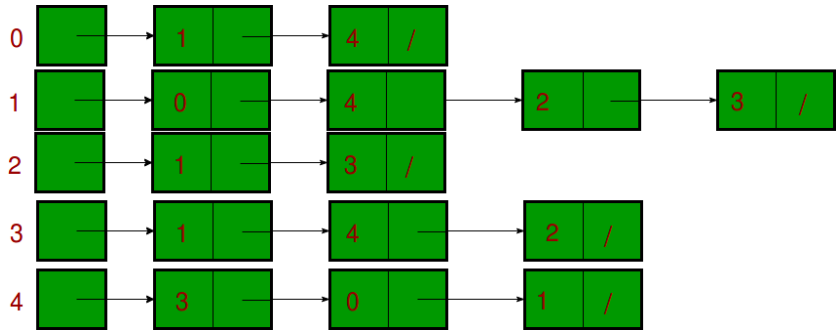


Figura 3.0: representación gráfica de una matriz de adyacencia.

[2]

Tipos de nodos

- *Nodo raíz:* el nodo raíz es el antepasado de todos los demás nodos de un gráfico. No tiene antepasado. Cada gráfico consta exactamente de un nodo raíz. Generalmente, debe comenzar a recorrer un gráfico desde el nodo raíz.
- *Nodos de hoja:* en un gráfico, los nodos de hoja representan los nodos que no tienen sucesores. Estos nodos solo tienen nodos ancestros. Pueden tener cualquier número de bordes entrantes, pero no tendrán ningún borde saliente.

Tipos de gráficos

- *No dirigido:* Un gráfico no dirigido (figura 4.0) es un gráfico en el que todos los bordes son bidireccionales, es decir, los bordes no apuntan en ninguna dirección específica.

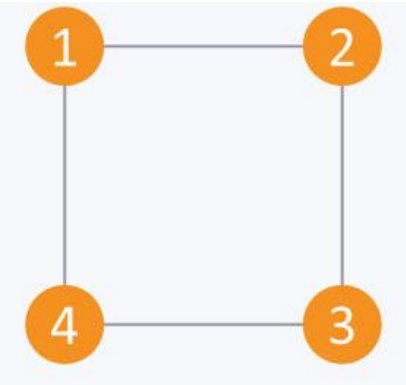


Figura 4.0: representación de un grafo no dirigido.

- *Dirigido:* Un gráfico dirigido (figura 5.0) es un gráfico en el que todos los bordes son unidireccionales, es decir, los bordes apuntan en una sola dirección.

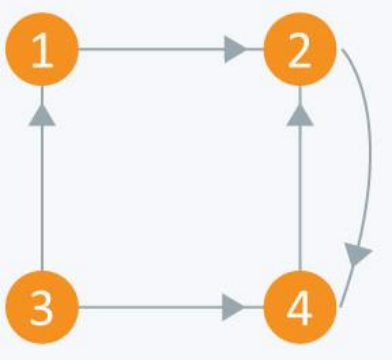


Figura 5.0: representación de un grafo dirigido.

- *Cíclico:* un gráfico es cíclico si el gráfico comprende una ruta que comienza en un vértice y termina en el mismo vértice. Ese camino se llama ciclo. Un gráfico acíclico es un gráfico que no tiene ciclo. Un árbol es un gráfico no dirigido (figura 6.0) en el que dos vértices cualesquiera están conectados por un solo camino.

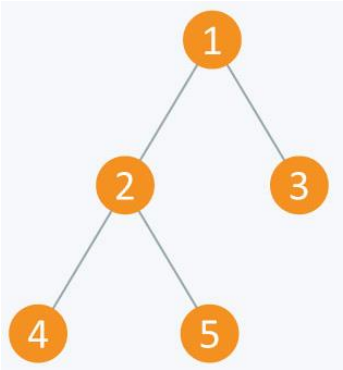


Figura 6.0: representación de un árbol.

[3]

Complejidad computacional

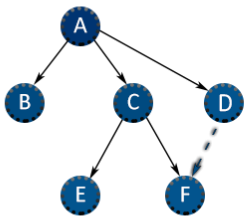
Data Structure	Time Complexity					
	Storage	Add Vertex	Add Edge	Remove Vertex	Remove Edge	Query
Adjacency list	O(|V|+|E|)	O(1)	O(1)	O(|V| + |E|)	O(|E|)	O(|V|)
Incidence list	O(|V|+|E|)	O(1)	O(1)	O(|E|)	O(|E|)	O(|E|)
Adjacency matrix	O(|V|^2)	O(|V|^2)	O(1)	O(|V|^2)	O(1)	O(1)
Incidence matrix	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|E|)

BFS y DFS

	BFS	*DFS*
1.	BFS son las siglas de Breadth First Search.	DFS son las siglas de Depth First Search.
2.	BFS (Breadth First Search) utiliza la estructura de datos de la cola para encontrar la ruta más corta.	DFS (Depth First Search) utiliza la estructura de datos Stack.
3.	BFS se puede utilizar para encontrar la ruta más corta de una sola fuente en un gráfico no ponderado, porque en BFS, llegamos a un vértice con un número mínimo de aristas desde un vértice de origen.	En DFS, podríamos atravesar más bordes para llegar a un vértice de destino desde una fuente.
4.	BFS es más adecuado para buscar vértices que están más cerca de la fuente dada.	DFS es más adecuado cuando hay soluciones fuera de la fuente.
5.	BFS considera a todos los vecinos primero y, por lo tanto, no es adecuado para árboles de toma de decisiones utilizados en juegos o rompecabezas.	DFS es más adecuado para problemas de juegos o rompecabezas. Tomamos una decisión, luego exploramos todos los caminos a través de esta decisión. Y si esta decisión lleva a una situación ganadora, nos detenemos.
6.	La complejidad de tiempo de BFS es O (V + E) cuando se usa Adjacency List y O (V ^ 2) cuando se usa Adjacency Matrix, donde V significa vértices y E significa bordes.	La complejidad de tiempo de DFS también es O (V + E) cuando se usa Adjacency List y O (V ^ 2) cuando se usa Adjacency Matrix, donde V significa vértices y E significa bordes.

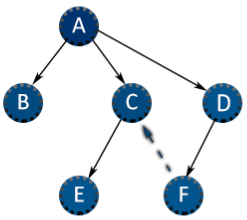
[4]

BFS



A B C D E F

DFS

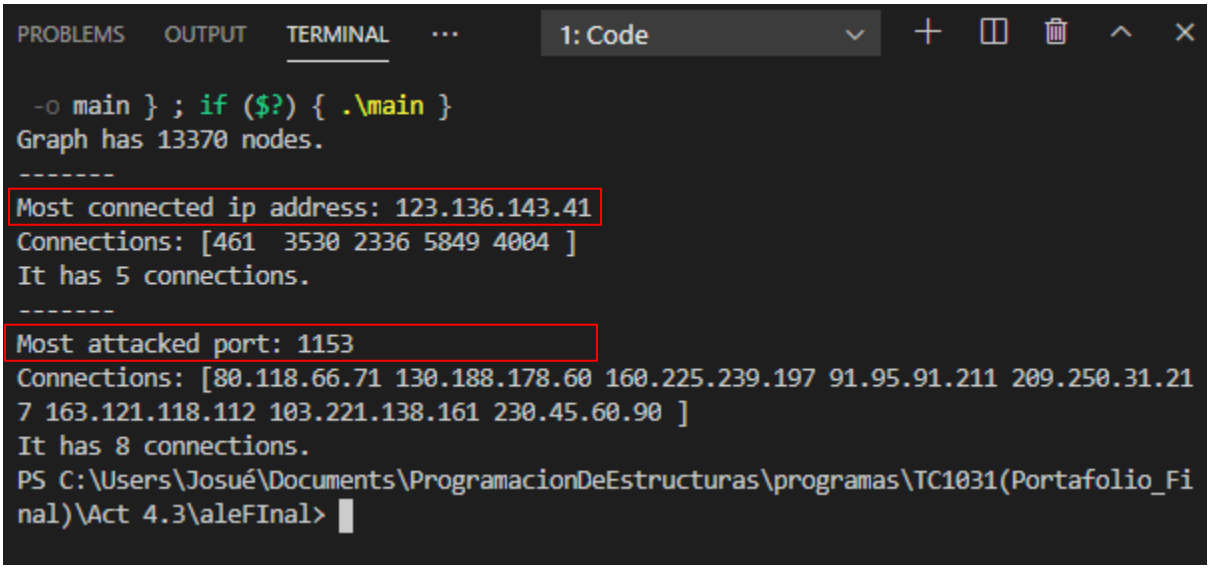


A D F C E B

Parte 2: Reflexión

En esta actividad integradora se llevó a cabo la implementación de un grafo para la estructuración de los datos con el fin de dar solución a la pregunta planteada. En la situación problema se pretende determinar el puerto más atacado y la IP con el mayor número de conexiones salientes para posteriormente determinar la dirección IP en la que presumiblemente se encuentra el boot master. Lo anterior es posible de determinar al llevar a cabo un análisis de los registros que se tienen acordes a los accesos que se han tenido en la red.

El hecho de que en una dirección IP se encuentre un boot master resulta como consecuencia de una serie de ataques que se hacen desde una misma unidad, por lo tanto, es posible concluir que el análisis de las IP de los dispositivos utilizados para acceder a la red permite identificar si alguna de ellas se presenta con un elevado número de repeticiones en el acceso, es decir, si en el registro se tiene una IP repetida un gran número de veces es posible determinar que el dispositivo perteneciente a dicha IP resulta ser el medio que se utilizó para pretender un ataque contra la red. Se podría pensar que dicha tarea resulta simple y sencilla, sin embargo, cuando se tienen una enorme cantidad de datos correspondientes a los registros de los accesos que se han tenido resulta difícil su análisis y procesamiento, por lo que es necesario estructurarlos para poder obtener la respuesta a los planteamientos que buscan resolverse. Particularmente en esta entrega, al trabajar con más de 16,000 registros, lo que se procedió a efectuar fue la estructuración de los datos en un grafo, estructura que, como se puede argumentar en la investigación previa, posibilita la interconexión de nodos a manera de redes. La estrategia efectuada para determinar si la red había sido o no infectada consistió en crear un grafo con las direcciones IP y con los puertos de cada registro contenido en el fichero de datos. Para conseguirlo, primero se procedió a iterar el fichero “bitácora.txt” para extraer los registros correspondientes a las IP’s y a los puertos, posteriormente se llama a una función que verifica que la IP y el puerto no se encuentren en un vector que contiene los registros únicos, en caso de no encontrarse se agregan a dichos vectores, de lo contrario un atributo llamado “counter” aumenta en 1, indicando que dicha IP o que dicho puerto han aparecido una vez más; al final, se iteran los vectores y de ellos se extraen la IP y el puerto que resultan tener el valor mayor para el contador. Los resultados de la ejecución del programa son los siguientes:



Como es posible de apreciar, la IP que tiene el mayor número de conexiones salientes es: 123.136.143.41 (con 5 conexiones), y el puerto que resulta ser más atacado es el: 1153 (con 8 conexiones). Tomando en cuenta el análisis anterior, es posible decir que el boot master se encuentra en la IP que resulta tener más conexiones (una mayor cantidad de ataques son provenientes de dicha IP), la cual, como se mencionó, resulta ser: 123.136.143.41

Referencias

[1] GeeksforGeeks, «GeeksforGeeks,» 2016. [En línea]. Available: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/. [Último acceso: 18 11 2020].

[2] GeeksforGeeks, 29 06 2020. [En línea]. Available: https://www.geeksforgeeks.org/graph-and-its-representations/. [Último acceso: 18 11 2020].

[3] Hackerearth, «https://www.geeksforgeeks.org/graph-and-its-representations/,» 2020. [En línea]. Available: https://www.hackerearth.com/practice/algorithms/graphs/graph-representation/tutorial/. [Último acceso: 18 11 2020].

[4] GeeksforGeeks, «GeeksforGeeks,» 03 07 2020. [En línea]. Available: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/#:~:text=BFS(Breadth%20First%20Search)%20uses,edges%20from%20a%20source%20vertex.. [Último acceso: 18 11 2020].