Reflexión Actividad 4.3

Programación de estructuras de datos y algoritmos fundamentales (TC1031.570)
Fausto Alejandro Palma Cervantes (A01639224)
27 de julio de 2022

Continuando con los temas que abarca la unidad de formación *Programación de estructuras de datos y algoritmos fundamentales*, este cuarto bloque de actividades se centró en torno a las estructuras de datos de tipo red. Más específicamente, este bloque se centró en en un tipo de este tipo de estructuras: los Grafos. Los grafos son estructuras de datos no lineales conformadas por nodos (elementos) y arcos (conexiones nodo a nodo). (*Graph Data Structure And Algorithms*, 2022) A diferencia de los árboles binarios, los elementos de este tipo de estructura no lineal tienen una cantidad ilimitada de conexiones. Ya que cada elemento puede también estar conectado a una cantidad ilimitada de nodos, esto prohíbe la existencia de relaciones "padre-hijo". Dentro de la Ingeniería en Tecnologías Computacionales, este tipo de estructura de datos es de gran utilidad. Los grafos nos permiten estructurar conjuntos de información cuyos datos tienen una relación de "muchos a muchos" (N:M), lo cual no es posible en ninguna de las estructuras de datos previamente. (Instituto Tecnológico y de Estudios Superiores de Monterrey, 2020). La siguiente figura muestra el diagrama de un grafo con cinco nodos (también conocidos como vértices o *vertices* en inglés) y siete arcos (aristas o *edges*):

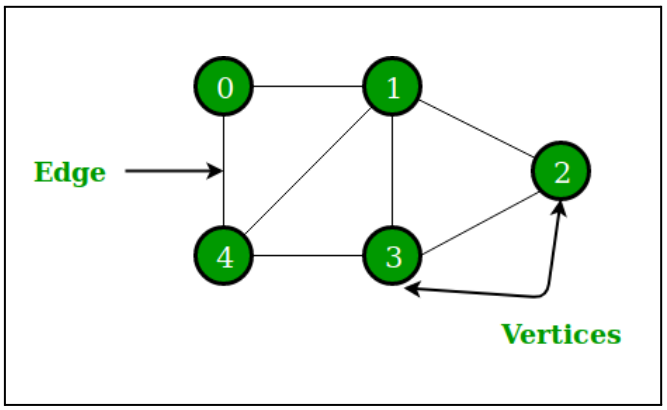

Figura 1. Diagrama de un grafo.
(*Graph Data Structure And Algorithms*, 2022)

Es importante mencionar que existen dos propiedades que comúnmente se utilizan en los arcos de un grafo. La primera es el valor o peso y normalmente se usa cuando se quiere considerar la prioridad de los arcos en el análisis del grafo. Generalmente, dado que esta estructura se utiliza para representar un conjunto de información real, el valor de un arco depende de una propiedad física como peso, distancia o costo. La segunda es la dirección y normalmente se utiliza cuando se le quiere dar cardinalidad a los arcos. Mientras que un arco con dirección tiene un nodo origen que conecta con un nodo destino, el arco sin dirección puede usar cualquiera de sus nodos como origen o destino. El poder determinar si se usan arcos con dirección en un grafo ayuda a poder distinguir entre un Grafo No-Dirigido y un

Grafo Dirigido (sin y con arcos direccionados respectivamente). (Instituto Tecnológico y de Estudios Superiores de Monterrey, 2020)

Independientemente de las propiedades presentes en los arcos de un grafo, durante este bloque se exploraron tres diferentes maneras de representar un grafo y dos distintos algoritmos para recorrerlo. A pesar de no ser las únicas, son algunas de las más eficientes y más utilizadas hoy en día para el análisis de grafos. A continuación la diferenciación entre cada uno de estos procesos:

1. **Representaciones**
    a. **Matriz de adyacencia:** Arreglo bidimensional cuadrado de tamaño correspondiente al número de nodos en el grafo dentro de la cual se indican la existencia (y valores a medida de lo posible) de las conexiones entre nodos. (GeeksforGeeks et al, 2022, *Graph and its representations*)
    b. **Lista de adyacencia:** Arreglo unidimensional de listas en el cual cada una almacena los nodos conectados a un nodo en específico. (GeeksforGeeks et al, 2022, *Graph and its representations*)
    c. **Lista de arcos:** Arreglo unidimensional de listas en el cual cada una almacena los arcos específicos a un nodo (de las tres es la representación es más compleja de implementar). (Instituto Tecnológico y de Estudios Superiores de Monterrey, 2020)
2. **Recorridos**
    a. **Breadth-First Search:** (BFS) Recorre el grafo recorriendo los nodos conectados a cada nodo desde un nodo inicial y asignándoles un status. Utiliza un *queue* dentro de su recorrido por lo cual le da prioridad de procesamiento al primer nodo recorrido (First In First Out) (GeeksforGeeks et al, 2022, *Breadth First Search or BFS for a Graph*)
    b. **Depth-First Search:** (DFS) Recorre el grafo recorriendo los nodos conectados a cada nodo desde un nodo inicial y asignándoles un status. Utiliza un *stack* dentro de su recorrido por lo cual le da prioridad de procesamiento al último nodo recorrido (Last In First Out) (GeeksforGeeks et al, 2022, *Depth First Search or DFS for a Graph*)

Para la entrega de la Actividad 4.3 (*Actividad Integral de Grafos*), se implementó la organización de registros interconectados dentro una bitácora a través de un grafo con dos propósitos principales: encontrar el número de conexiones de cada registro para identificar el registro con el mayor número de conexiones y posteriormente encontrar el camino más corto entre dicho registro y cada otro registro de la bitácora. Considerando los objetivos mencionados, se decidió utilizar un grafo representado en una lista de adyacencias y se utilizaron tres métodos para cumplir con los propósitos de la actividad. El primer método se utilizó para importar la información de cada uno de los registros y sus conexiones y con ello poder posteriormente crear una bitácora. Este método requirió una complejidad computacional $O(V + E)$ (siendo V nodos y E arcos) ya que se necesitó extraer y hacer las operaciones correspondientes con un archivo listando cada registro (nodo) y conexión (arco) de la bitácora. Siguiendo con el segundo método, este se centró en encontrar el número de

conexiones de cada registro y encontrar el *boot master* (registro con mayor cantidad de conexiones). El segundo método se llevó a cabo utilizando una complejidad computacional $O(V + E)$ ya que, con ayuda del primer método, únicamente se necesitó pasar por cada lista dentro de la representación del grafo (lista de adyacencias) y conseguir el número de elementos en la lista (ya que cada conexión equivale a un elemento). Finalmente para el tercer método se encontró el camino más corto entre el registro con el mayor número de conexiones y todos los otros registros de la bitácora y logró con una complejidad computacional $O(E \log V)$. Al hablar de la complejidad de este segundo método, es importante destacar que eso se debe porque se utiliza una variación del algoritmo de recorrido Breadth-Search First.

Recordando que la situación problema de esta unidad de formación se centra en la detección de ataques cibernéticos de redes de bots, los grafos son de gran valor ya que nos ayudan a ordenar conjuntos de información cuyos datos se ven interrelacionados. Dentro de un ataque de red de bots real y considerando el conocimiento adquirido a través de esta actividad, al identificar una dirección IP con la mayor cantidad de conexiones a otras direcciones identificar la IP desde donde se está llevando a cabo la mayor cantidad de daño del ataque. Posteriormente, encontrando los diferentes valores para cada una de las conexiones entre el *boot master* y las demás direcciones IP permitiría encontrar la seguridad que existe dentro de cada dirección siendo que en este caso a mayor valor mayor esfuerzo requiere la red de bots para llevar a cabo su ciberataque.

Bibliografía:

GeeksforGeeks et al. (2022). *Breadth First Search or BFS for a Graph*. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/?ref=lbp

GeeksforGeeks et al. (2022). *Depth First Search or DFS for a Graph*. GeeksforGeeks. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/?ref=lbp

GeeksforGeeks et al. (2022). *Graph and its representations*. GeeksforGeeks. https://www.geeksforgeeks.org/graph-and-its-representations/

Graph Data Structure And Algorithms. (2022). GeeksforGeeks. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Instituto Tecnológico y de Estudios Superiores de Monterrey. (2020). *sesion18-Grafos*[pdf].