Reflexión Actividad 3.4

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

A diferencia de los bloques de actividades anteriores de esta unidad, este bloque se centró en estructuras de datos no lineales. Dentro de la Ingeniería en Tecnologías Computacionales, este tipo de estructura de datos son de gran utilidad ya que, similar a lo que ocurre con las lineales, facilitan la interpretación e interacción con conjuntos de datos. La mayor diferencia entre ambos tipos de estructuras mencionadas es que las estructuras de datos no lineales organizan sus elementos de manera jerárquica utilizando varios niveles. Esto impide el acceso de todos sus elementos en un solo movimiento ya que no están en una secuencia lineal continua. Esta propiedad hace además que la implementación de estructuras no lineales sea más difícil pero use la memoria computacional de manera más eficiente. (MKS075, 2022) Siendo más específicos, este bloque únicamente se centró en un tipo de estructura de datos no lineales: los árboles binarios. Este tipo de estructura está conformada por nodos (elementos con dos apuntadores) los cuales apuntan a otros nodos "hijos". (*Binary Search Tree*, 2022) Existen diferentes clasificaciones para los árboles binarios en base a su estructura, incluyendo:

- **Árbol binario estricto:** Árbol en el cual cada nodo tiene dos o cero hijos.
- **Árbol binario completo:** Árbol en el cual todos sus niveles superiores están llenos y el último (en caso de no estar lleno) tiene sus nodos lo más a la izquierda posible.
- **Árbol binario perfecto:** Árbol en el cual todos sus niveles están llenos.
- **Árbol binario balanceado:** Árbol en el cual la diferencia de altura entre los subárboles de cada uno de sus nodos es menor a uno.

(GeeksforGeeks et al, 2022, *Binary Tree*) Además de los árboles binarios tradicionales, existen distintas variaciones las cuales les dan funcionalidades particulares a sus conjuntos de datos y también utilizan este tipo de estructura de datos no lineal. Cada uno de ellos hace uso de propiedades adicionales a las del árbol binario tradicional se utilizan con un propósito en particular dependiendo de factores como la complejidad computacional de sus operaciones básicas. A continuación se describen las cuatro variaciones de los árboles binarios tradicionales cubiertas en clase seguido por una tabla con las complejidades computacionales de cada uno:

1. **Árboles de búsqueda binaria:** Tipo de árbol binario conocido como ABB (o BST por sus siglas en inglés) en el cual todos sus elementos tienen hijos de menor valor a su izquierda y de mayor valor a su derecha. (*Binary Search Tree*, 2022)
2. **Árboles AVL:** Tipo de árbol de búsqueda binaria el cual se mantiene balanceado al añadir y borrar nodos. (GeeksforGeeks et al, 2022, *AVL Tree*)

3. **Árboles Heap:** Tipo de árbol binario completo en el cual cada nodo padre tiene un valor de mayor prioridad que sus hijos (sea de mayor o menor valor). (*Heap Data Structure*, 2022)
4. **Árboles Desplegados:** Tipo de árbol de búsqueda binaria auto-balanceable también conocido como *Splay Trees* en el cual los nodos más utilizados están más cerca de la raíz. (GeeksforGeeks et al. (2022)

Tipo de árbol	Insertar elemento	Eliminar elemento	Buscar elemento
Árbol binario	O(n)	O(n)	O(n)
Árbol de búsqueda binaria	O(n)	O(n)	O(n)
Árbol AVL	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
Árbol Heap	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
Árbol desplegado	O(log n)	O(log n)	O(log n)

Tabla 1. Complejidad computacional de las operaciones básicas de cada tipo de árbol binario (GeeksforGeeks et al, 2018)

Es importante aclarar que debido a las diferencias presentes en este tipo de estructuras de datos, no todos los algoritmos de ordenamiento y búsqueda mencionados en bloques anteriores pueden utilizarse con árboles binarios. Por ello, y como parte de los temas vistos dentro de los árboles Heap, se introdujo un nuevo algoritmo de ordenamiento particular a este tipo de estructura de datos: el *Heap Sort*. Dentro de este algoritmo, similar a lo que ocurre con el ordenamiento por selección, se intercambia el valor de mayor prioridad dentro del Heap (raíz/primer valor) con el valor de menor prioridad (último valor). Una vez teniendo el valor de mayor prioridad al final del Heap, este se vuelve a organizar excluyendo dicho valor. Dicho proceso se lleva a cabo repetidamente excluyendo siempre los valores que se van colocando al final hasta que se logran ordenar los valores del Heap de menor a mayor prioridad. (GeeksforGeeks et al, 2022, *HeapSort*). La siguiente tabla muestra las complejidades computacionales de todos los algoritmos vistos hasta ahora:

Algoritmo de ordenamiento	Complejidad computacional
Swap sort	$O(n^2)$
Selection sort	$O(n^2)$
Bubble sort	$O(n^2)$
Insertion sort	$O(n^2)$
Merge sort	O(n log n)
Quicksort	O(n log n)* *su peor caso tiende a ser $O(n^2)$

Heap sort	O(n log n)

Tabla 2. Complejidad computacional de algoritmos de ordenamiento
(Instituto Tecnológico y de Estudios Superiores de Monterrey, 2020)

Para la entrega para la Actividad 3.4 (*Actividad Integral de BST*), se solicitó la organización de registros dentro una bitácora con el mismo propósito que las actividades anteriores pero esta vez utilizando una estructura de datos BST. Posteriormente, esto registros se ordenaron en base a la prioridad de sus valores para poder encontrar los más repetidos. Cabe destacar que a diferencia de actividades anteriores, en esta entrega se hizo uso de distintos registros (tanto en cantidad como en información). Dado el enfoque de las instrucciones, se decidió organizar la información dentro de un Árbol Heap ya que a pesar de que el Árbol Heap no tenga la simplicidad computacional que tiene el Árbol desplegado, el algoritmo de ordenamiento específico a él es de los más simples computacionalmente hablando.

Considerando la complejidad computacional de la inserción de elementos en este tipo de estructuras y la cantidad *n* de registros de esta actividad, la creación de un Heap para esta bitácora tuvo una complejidad computacional de $O(n \log_2 n)$. Además, cabe mencionar que el árbol Heap que utilizó en esta actividad dio prioridad a los elementos de menor valor. Esta decisión se tomó considerando que con este tipo de priorización el algoritmo de ordenamiento Heap sort ordenaría los elementos de la estructura de mayor a menor valor. Ello permitió que el algoritmo de ordenamiento se pudiera usar dos veces sin hacer cambios: una para ordenar los registros de la bitácora y otra para que al tener el número de repeticiones para cada valor dentro de otro árbol Heap se pudieran colocar los valores más repetidos al principio de la estructura para su fácil acceso.

Finalmente, igual a lo que se ha hecho en todas las actividades integrales anteriores, es importante mencionar el valor de las estructuras de datos no lineales y árboles binarios para la situación problema. Recordando que esta se enfoca en la detección de ataques cibernéticos de redes de bots, los árboles binarios (especialmente los de búsqueda) nos ayudan a ordenar los conjuntos de información para una búsqueda de elementos más eficiente. Las propiedades de las estructuras no lineales permiten no tener que pasar por cada uno de sus elementos al hacer una búsqueda e incluso estructuran la información para que su ordenamiento sea más eficiente que con estructuras de datos lineales. Al tener una recopilación con toda la información relevante para la identificación de ataques de redes de bots, justo como la que se tuvo para esta actividad, las estructuras de datos permiten la búsqueda y ordenamiento de instancias con ciertas características. Esto demuestra la utilidad que tienen en conjunto las estructuras de datos con los algoritmos de búsqueda y ordenamiento ya que permiten determinar si una red está infectada.

Bibliografía:

Binary Search Tree. (2022). GeeksforGeeks. https://www.geeksforgeeks.org/binary-search-tree-data-structure/

Binary Tree Data Structure. (2022). GeeksforGeeks. https://www.geeksforgeeks.org/binary-tree-data-structure/

GeeksforGeeks et al. (2018). *Complexity of different operations in Binary tree, Binary Search Tree and AVL tree*. GeeksforGeeks. https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/

GeeksforGeeks et al. (2022). *AVL Tree | Set 1 (Insertion)*. GeeksforGeeks. https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

GeeksforGeeks et al. (2022). *Binary Tree | Set 3 (Types of Binary Tree)*. GeeksforGeeks. https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/

GeeksforGeeks et al. (2022). *HeapSort*. GeeksforGeeks. https://www.geeksforgeeks.org/heap-sort/

GeeksforGeeks et al. (2022). *Splay Tree | Set 1 (Search)*. GeeksforGeeks. https://www.geeksforgeeks.org/splay-tree-set-1-insert/

Heap Data Structure. (2022). GeeksforGeeks. https://www.geeksforgeeks.org/heap-data-structure/

Instituto Tecnológico y de Estudios Superiores de Monterrey. (2020). *sesion16-Heaps*[pdf].

MKS075. (2022). *Difference between Linear and Non-linear Data Structures*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-linear-and-non-linear-data-structures/