INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS

REFLEXIÓN

TC1031.501: PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS (GPO 501)

Michelle Andrea Arceo Solano.
Tecnológico de Monterrey, Campus AGS
a01625268@itesm.mx

En las estructuras de datos jerárquicos se encuentran los árboles binario, estos árboles pueden tener más de dos sucesores y de hecho se les llama árboles porque si los vemos gráficamente tiene la forma de un árbol pero al revés (como si fuera árbol genealógico) donde la raíz es la parte superior y se van extendiendo los datos hacia abajo. Aquí se encuentra un tipo de árbol llamado *Heap* y cumple dos propiedades básicas:

- El valor de cada nodo es mayor o igual al valor de cada uno de sus hijos.
- El árbol se encuentra perfectamente balanceado, en el último nivel los nodos se encuentran más a la izquierda.

Los árboles *Heap* tienen una complejidad del número de nivel es de ***O(log n)*** pero si el árbol binario se encuentra completo, quiere decir que tiene la altura más pequeña por lo que su complejidad es de $\log_2 n$. [1]

En la siguiente tabla podemos observar la complejidad de las 4 operaciones del Heap:

Operaciones	Eficiencia (Peor de los casos)
IsEmpty() [2]	Su complejidad es $O(1)$, ya que solo verifica si hay datos o no.
Insert() [2]	Su complejidad es $O(log\ n)$, ya que se tiene que mover el elemento insertado hasta arriba para encontrar el lugar correcto donde debe estar según el valor del nodo.
GetMax() [2]	Su complejidad es $O(1)$, ya que solo despliega el valor que se encuentra en la raíz.
ExtractMax() [2]	Su complejidad es $O(log\ n)$, ya que al eliminar el valor máximo, se reemplaza por el valor mínimo en la raíz por lo que se vuelve a acomodar todo el árbol.

Entre las varias ventajas, este tipo de árbol es posible de agregar datos de forma eficaz y eficiente después de construirlo inicialmente, al no tener que hacerse un doble trabajo es un buen método para cuando se tienen numerosos datos.

Para la solución de la situación problema se utilizó este método ya que las operaciones, principalmente la inserción es de tiempo *O(log n)*, es decir, son muy rápidas, también la implementación del algoritmo a comparación de otros árboles es más eficiente y concreto por lo que es más funcional para alinear miles de registros y encontrar IP's con mayores accesos, este es un método que se amolda de acuerdo a nuestras necesidades, por lo que si una red se encuentra infectada es cuestión de detectar si existe alguna falla en las transformaciones de las órdenes del arreglo unidimensional o algún comportamiento inusual en el tráfico de redes.

REFERENCIAS

[1] Drozdek, A. (2012). *Data Structures and algorithms in C++*. Cengage Learning.

[2] Anggoro, W. (2018). *C++ Data Structures and Algorithms: Learn how to write efficient code to build scalable and robust applications in C++*. Packt Publishing Ltd.