CAMPUS QUERÉTARO

TC1031. PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS FUNDAMENTALES

(GRUPO 826)

"Actividad Integradora 3. Estructura de datos jerárquica"

Evidencia presentada por la estudiante:

A01706095 - Naomi Estefanía Nieto Vega

Profesor:

Dr. Eduardo Arturo Rodríguez Tello

Fecha de entrega:

Lunes 10 de mayo de 2021

Actividad Integradora 3. Estructura de datos jerárquica

Naomi Estefanía Nieto Vega

***Index Terms*—data structure, binary search tree, BST, complexities**

***Resumen*—En esta actividad integradora, se dará solución a la problemática del manejo de grandes cantidades de datos y sobre la importancia y eficiencia del uso de Binary Search Tree (BST) en una situación problema de esta naturaleza.**

I. INTRODUCCIÓN

Hoy en día los algoritmos de ordenamiento y las estructuras de datos son muy importantes en el área computacional y en muchas más áreas en los que son utilizados ya que de acuerdo con la literatura un algoritmo es una serie o secuencia ordenada de pasos que dan solución a un determinado problema. No obstante, pueden existir distintos algoritmos que den solución a un mismo problema y es aquí donde entra la importancia de implementar el más eficiente en cuanto a tiempo y recursos ya que estos deben ser utilizados correctamente y de esta forma la computadora podrá darnos resultados muy buenos.

Asimismo, al conocer distintos tipos de estructuras de datos y su implementación podemos hacer uso de ellos para mejorar el manejo que tenemos sobre los algoritmos de acuerdo con las necesidades que tenga el problema, y así proveer una solución adecuada, eficiente y rápida.

Para el desarrollo de esta situación problema, principalmente nos enfrentamos al problema del manejo de una bitácora con gran volumen de datos, aunado a esto debemos realizar búsquedas y conteos dentro de la bitácora para conocer las direcciones IP con más accesos, asimismo conocer las consideraciones en seguridad que esto representa, esto es un factor muy importante a considerar para plantear una solución adecuada y eficiente, que se explicará a continuación.

II. DESARROLLO

Las estructuras de datos son formas de organización y representación de datos o información, en el caso de la programación, son formas de organización que nos permiten utilizar estos datos de una forma efectiva. Existen dos tipos de estructuras de datos:

1. Estructuras de datos lineales
2. Estructuras de datos no lineales

Su principal diferencia es que la estructura de datos lineal tiene a sus elementos de datos dispuestos de manera secuencial y cada uno está conectado con su elemento anterior y siguiente. Por otra parte, las estructuras de datos no lineales no tienen una secuencia establecida para conectar a sus elementos, por lo que un elemento puede tomar varias rutas para conectarse con otros elementos. Este tipo de estructuras son un poco más complejas de implementar pero son más eficientes en cuanto a uso de memoria en la computadora. [2] A continuación, en la figura 1 podemos observar un gráfico con las complejidades (temporal y espacial) de las distintas estructuras de datos que existen.[1]

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Figura 1. Complejidades de las estructuras de datos más comúnes. Obtenido de: https://www.bigocheatsheet.com/

Los Árboles Binarios de Búsqueda (o Binary Search Tree (BST), por sus siglas en inglés) son una estructura de datos jerárquica basada en nodos donde cada nodo tiene una clave (key) y un valor asociado. [3] Permite operaciones como búsqueda, añadir, y eliminar elementos de forma muy rápida y eficaz. Los nodos se organizan de acuerdo con las siguientes características:

- El subárbol izquierdo de un nodo contiene sólo nodos con claves menores que la clave del nodo.
- El subárbol derecho de un nodo contiene sólo nodos con claves mayores que la clave del nodo.
- Los subárboles izquierdo y derecho deben ser un BST.

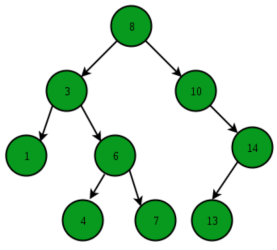

Figura 2. Representación de un árbol binario de búsqueda. Obtenido de: https://www.geeksforgeeks.org/binary-search-tree-data-structure/

Esta estructura de datos es capaz de representar a los datos en una estructura jerárquica y es comúnmente utilizada en muchas aplicaciones de búsqueda en las que los datos

entran y salen constantemente del mapa. Los BST tienen una complejidad temporal promedio de O(log n) y sus principales funciones como búsqueda, inserción y borrado tienen en promedio una complejidad de O(log n) y en su peor caso tienen una complejidad de O(n). En la figura 3, podemos observar un resumen de las complejidades de las operaciones mencionadas anteriormente.

Operación	Promedio	Peor
Buscar	O(log n)	O(n)
Insertar	O(log n)	O(n)
Borrar	O(log n)	O(n)

Figura 3. Complejidades temporales de las principales operaciones de los BST.

Los BST son importantes en la situación problema porque gracias a ellos podemos buscar de una forma muy eficiente en la bitácora con las direcciones IP por sus accesos dependiendo y dependiendo de cuántas veces aparece la misma IP será la cantidad de accesos. En este caso para determinar si una red está infectada por la cantidad de accesos en cada IP tendríamos que analizar la respuesta del servidor, tomando como "infectada" a aquellas direcciones IP que han regresado algún error al momento del acceso, de esta forma gracias a las operaciones que nos es posible realizar con los BST podemos buscar eficientemente las IP que posiblemente han sido infectadas debido a la cantidad de accesos concurrentes y que además tienen algún mensaje de error como respuesta. Es por ellos que los BST son bastante eficaces y útiles en situaciones de esta naturaleza.

III. CONCLUSIÓN

En conclusión, gracias a la actividad pude comprender mejor el funcionamiento de los árboles binarios de búsqueda, así como la importancia de su uso en distintas aplicaciones para la industria como lo es el manejo eficiente de grandes volúmenes de datos y administración de los recursos de la computadora, ya que para realizar software que sea escalable y pueda funcionar de una forma correcta y eficiente en todos los dispositivos es necesario aprovechar bien los recursos y tener en cuenta que se usarán en distintos aparatos con distintas características. Además logré comprender mejor el análisis de complejidad de las principales operaciones de los BST.

REFERENCIAS

[1] Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell. (2020). Big O Algorithm Complexities. https://www.bigocheatsheet.com/

[2] Difference between Linear and Non-linear Data Structures. (2020). Tutorialspoint. https://www.tutorialspoint.com/difference-between-linear-and-non-linear-data-structures

[3] GeeksforGeeks. (2020). Binary Search Tree. https://www.geeksforgeeks.org/binary-search-tree-data-structure/