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

Programación de estructuras de datos y algoritmos fundamentales

*Módulo:* Estructura de datos jerárquicas (árboles)

Actividad 3.4: Actividad integral de BST (Evidencia de competencia)

Investigación y reflexión individual

*Fecha de entrega:* 25 de octubre de 2020

**Objetivo**

Realizar una investigación y reflexión en forma individual de la importancia y eficiencia del uso de BST en una situación problema de esta naturaleza. ¿Cómo podrías determinar si una red está infectada o no?

**Parte 1: Investigación**

***Binary Search Tree***

Un árbol de búsqueda binaria (figura 1.0) es una estructura de datos de árbol binario basada en nodos que cumplen las siguientes propiedades:

- 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.
- El subárbol izquierdo y derecho también deben de ser un árbol de búsqueda binaria.

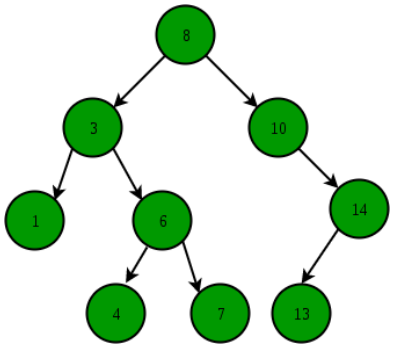


Figura 1.0: representación gráfica de una lista doblemente ligada.

Las propiedades anteriores de Binary Search Tree proporcionan un orden entre las claves para que las operaciones como la búsqueda, el mínimo y el máximo se puedan realizar rápidamente.

[1]

***Buscando una clave***

Digamos que queremos buscar un número, lo que haremos es comenzar en la raíz, y luego compararemos el valor a buscar con el valor de la raíz, si es igual, terminamos con la búsqueda, si es menos, sabemos que tenemos que ir al subárbol izquierdo porque en un árbol de búsqueda binario, todos los elementos del subárbol izquierdo son menores y todos los elementos del subárbol derecho son mayores. La búsqueda de un elemento en el árbol de búsqueda binaria es básicamente este recorrido en el que en cada paso iremos hacia la izquierda o hacia la derecha y, por lo tanto, en cada paso descartamos uno de los subárboles. Si el árbol está equilibrado (llamamos a un árbol equilibrado si para todos los nodos la diferencia entre las alturas de los subárboles izquierdo y derecho no es mayor que uno) comenzaremos con un espacio de búsqueda de 'n' nodos y cuando descartemos uno de los subárboles descartaremos 'n/2' nodos para que nuestro espacio de búsqueda se reduzca a 'n/2' y luego, en el siguiente paso, reduciremos el espacio de búsqueda a 'n/4' y seguiremos reduciendo así hasta que encontremos el elemento o hasta que nuestro espacio de búsqueda se reduzca a un solo nodo. La búsqueda aquí también es una búsqueda binaria y por eso el nombre árbol de búsqueda binaria.

***Inserción de una llave***

Siempre que se inserta una nueva llave en la hoja comenzamos a buscar una clave desde la raíz hasta que llegamos a un nodo hoja. Una vez que se encuentra un nodo hoja, el nuevo nodo se agrega como hijo del nodo hoja (figura 2.0).

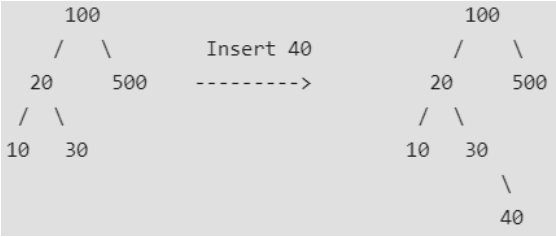


Figura 2.0: representación de inserción de una llave

***Complejidad temporal: buscar e insertar***

La complejidad temporal en el peor de los casos de las operaciones de búsqueda e inserción es O (h) donde h es la altura del árbol de búsqueda binaria. En el peor de los casos, es posible que tengamos que viajar desde la raíz hasta el nodo de hoja más profundo. La altura de un árbol sesgado puede convertirse en n y la complejidad temporal de la operación de búsqueda e inserción puede convertirse en O (n).

[2]

***Eliminar***

Cuando eliminamos un nodo, surgen tres posibilidades.

1. *El nodo que se eliminará es hoja:* simplemente elimínelo del árbol (figura 3.0).


Figura 3.0: eliminación cuando el nodo es hoja

2. *El nodo que se eliminará tiene un solo hijo:* copie el hijo en el nodo y elimine el hijo (figura 4.0).


Figura 4.0: eliminación cuando el nodo tiene un solo hijo

3. *El nodo que se eliminará tiene dos hijos:* Buscar en orden sucesor del nodo. Copie el contenido del sucesor del orden en el nodo y elimine el sucesor del orden. Tenga en cuenta que también se puede utilizar el predecesor en orden (figura 5.0).


Figura 5.0: eliminación cuando el nodo tiene dos hijos

Lo importante para tener en cuenta es que el sucesor de orden solo se necesita cuando el hijo correcto no está vacío. En este caso particular, el sucesor en orden se puede obtener encontrando el valor mínimo en el hijo derecho del nodo.

***Complejidad de tiempo: eliminar***

La complejidad de tiempo del peor caso de la operación de eliminación es O (h) donde h es la altura del árbol de búsqueda binaria. En el peor de los casos, es posible que tengamos que viajar desde la raíz hasta el nodo de la hoja más profundo. La altura de un árbol sesgado puede convertirse en n y la complejidad temporal de la operación de eliminación puede convertirse en O (n).

[3]

***Importancia y eficiencia***

La razón por la que los árboles de búsqueda binaria son importantes es que las siguientes operaciones se pueden implementar de manera eficiente usando una BST:

- Insertar un valor clave
- Determinar si un valor clave está en el árbol
- Eliminar un valor clave del árbol
- Imprimir todos los valores clave en orden ordenado

[4]

Los BST se utilizan para muchas aplicaciones debido a su estructura ordenada.

- Los BST se utilizan para la indexación y la indexación de varios niveles.
- También son útiles para implementar varios algoritmos de búsqueda.
- Es útil para mantener un flujo de datos ordenado.

[5]

***Complejidad computacional***

| ***Operación*** | ***Caso promedio*** | ***Peor caso*** |
|---|---|---|
| *Acceder* | O(log(n)) | O(n) |
| *Buscar* | O(log(n)) | O(n) |
| *Insertar* | O(log(n)) | O(n) |
| *Eliminar* | O(log(n)) | O(n) |

**Parte 2: Reflexión**

En esta actividad integradora se llevó a cabo la implementación de un BST para la estructuración de los datos con el fin de dar solución a la pregunta planteada. En la situación problema se pretende determinar si una red ha sido infectada o no, misma cuestión que es posible de determinar al llevar un análisis de los registros que se tienen acordes a los accesos que se han tenido a la red.

El hecho de que una red sea infectada sucede 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 Binary Search Tree, estructura que, como se puede argumentar en la investigación previa, posibilita funciones de inserción, búsqueda, eliminación y acceso a coste de una complejidad computacional temporal eficiente. La estrategia planteada para determinar si la red había sido o no infectada consistió en crear un BST con las direcciones IP de cada registro contenido en el fichero de dato, el cual, conforme fuese iterado permitiría determinar si la siguiente IP resultaba ser única o repetida, donde de cumplirse el segundo caso un atributo de cada nodo controlaría el número de veces que cierta IP aparece dentro del conjunto de datos del fichero; al final, todo esto para posibilitar acceder a la respuesta plateada de una forma rápida y eficiente: ¿la red ha sido infectada?, de tener una IP repetida muchas veces la respuesta sería sí, sin embargo, para esta situación se dio el caso contrario: todas las IP que se encuentran en el registro resultan ser únicas, es decir, no existe evidencia para demostrar que la red ha sido atacada. Esta cuestión es posible de demostrar, podemos explicar matemática y computacionalmente por qué no pueden existir dos IP repetidas:

$$P(a) \cap P(b) = P(a) * P(b)$$

$$\frac{1}{1000} * \frac{1}{1000} * \frac{1}{1000} * \frac{1}{1000} = \left(\frac{1}{1000}\right)^4 = \frac{1}{1000000000000}$$

Tendríamos que tener mil millones de líneas para que estadísticamente una IP se repitiera

$$\frac{17000}{1 * 10^{12}} = 1.7 * 10^{-8}$$

Probabilidad de tener una IP repetida en el fichero

Como es posible darse cuenta, el número de datos que se deberían de tener como mínimos para estadísticamente decir que hay una IP repetida es de 1 000 000 000 000 registros, para este caso sólo tenemos 16,807 registros, por tanto, la probabilidad de ter una IP repetida es de 1.7*10^-8, se podría decir que nula, y efectivamente el programa desarrollado lo demuestra:

```
PS C:\Users\Josué\Documents\ProgramacionDeEstructuras\programas>
Act3.4\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Direcciones ip con mayor cantidad de accesos (Puesto | No. de ac
1. 1 974.11.969.97
2. 1 480.98.506.70
3. 1 240.63.109.57
4. 1 121.8.964.13
5. 1 62.97.550.14
PS C:\Users\Josué\Documents\ProgramacionDeEstructuras\programas\
```

Analizado lo anterior es prudente plantearse la pregunta: ¿realmente lo más conveniente para esta situación específica es implementar un BST? La respuesta es no, debido a que tenemos datos únicos para las IP el uso de un BST realmente no ofrece una ventaja para la obtención de la respuesta que en un principio se pretende encontrar.

**Referencias**

[1] Geeks for Geeks, «GeeksforGeeks,» 2016. [En línea]. Available: https://www.geeksforgeeks.org/binary-search-tree-data-structure/. [Último acceso: 25 10 2020].

[2] Geeks for Geeks, «GeeksforGeeks,» 22 10 2020. [En línea]. Available: https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/. [Último acceso: 25 10 2020].

[3] Geeks for Geeks, «GeeksforGeeks,» 03 04 2020. [En línea]. Available: https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/. [Último acceso: 25 10 2020].

[4] University of Wisconsin-Madison, «Computer Sciences User Pages,» [En línea]. Available: http://pages.cs.wisc.edu/~vernon/cs367/notes/9.BST.html#:~:text=The%20reason%20binary%2Dsearch%20trees,key%20value%20from%20the%20tree. [Último acceso: 25 10 2020].

[5] After Academy, «AfterAcademy,» 11 02 2020. [En línea]. Available: https://afteracademy.com/blog/binary-search-tree-introduction-operations-and-applications. [Último acceso: 25 10 2020].