# Reflexión Actividad 3.4 (Evidencia)

Santiago Reyes Moran - A01639030 - 06/11/2021

## Importancia y eficiencia de los BST

Los binary search trees (BST's) son sumamente útiles en situaciones donde los elementos que se encuentran dentro de la estructura de datos se pueden comparar entre sí como mayor o menor, es decir, tiene un valor explícito que hace que un elemento sea mayor o menor a otro. La estructura de un binary search tree implica que cada nodo puede tener un atributo left y uno right, el left será un elemento que es menor al elemento padre y el right será un elemento mayor. Tener una estructura de este tipo nos permite hacer búsquedas de manera eficiente utilizando recorridos inorder, preorder o postorder .

Una de las ventajas más importantes de los BST´s es que la inserción y el borrado de elementos dentro de la estructura es más eficiente que en los arreglos o vectores cotidianos. Si quisiéramos insertar un elemento en la posición número 50 dentro de un arreglo de 100 elementos, tendríamos que actualizar el índice todos los elementos después del que acabamos de insertar, entonces al querer insertar un elemento en la primera posición de un arreglo, la complejidad temporal de la inserción sería de O(n), ya que tenemos que actualizar los índices de todos los elementos del arreglo. Esta complejidad temporal es ineficiente (al compararla con la del BST) y puede ser problemático en estructuras con cientos de miles de elementos, porque la complejidad temporal del borrado de un dato es la misma.

En un BST, la complejidad temporal de un borrado y de una inserción es O(log(n)), lo cual es mucho mejor en situaciones como la que se presenta en esta actividad, en la cual hay aproximadamente 17,000 datos. Debido a que todos los elementos utilizados pueden ser comparados entre sí (con la IP para la primera parte de la actividad), entonces utilizar un BST es mucho más conveniente y eficiente, tanto para el ordenamiento como para el borrado (.pop()) que se tiene que ejecutar en esta actividad.

## Algoritmos utilizados y su complejidad

### Heap Sort:

El Heapsort es el método que se utiliza para ordenar la bitácora completa de registros basado en el valor numérico de la IP. Este método únicamente se utiliza en el MaxHeap de registros para poderla escribir con dichos registros ordenados en base a su IP. El algoritmo hace uso de el método de Heapify, el cual tiene una complejidad temporal O(log(n)), pero al estarlo llamando una cantidad n de veces, la complejidad temporal total del algoritmo de Heapsort se vuelve O(nlog(n)), la cual es una buena complejidad temporal para un algoritmo de ordenamiento.

### Push:

Este método es el que se utiliza para agregar elementos al Max Heap. En un vector normal, la complejidad temporal es constante O(1), sin embargo, como tenemos que asegurar que las propiedades básicas de un Heap no son violadas y corregir el heap si sí lo son, entonces la complejidad temporal se vuelve O(log(n)). Sin embargo, si la inserción inicial se hace sin violar las propiedades del Heap entonces sería el mejor caso O(1). Si todos los push se hacen de forma correcta, la raíz (el primer elemento del vector) será el elemento con mayor prioridad.

### Pop:

El Pop tiene la misma complejidad temporal que el Push O(log(n)), lo cual se debe a que después de borrar el elemento deseado se debe de utilizar el método de heapify; este tiene una complejidad temporal de O(log(n)). Heapify se debe de implementar para asegurar que las propiedades del heap no se violen y por lo tanto el elemento con mayor prioridad quedará como la raíz (el primer elemento del vector) después del pop.

## Cómo determinar si una red está infectada

Se podría determinar si una red está infectada o no al observar los accesos que se han hecho a esa red, poniendo atención a los accesos por IP. Si se registra una cantidad muy grande de accesos a una red por parte de una sola IP en poco tiempo, es posible que se esté haciendo un ataque tipo DDOS desde esa IP. Otra forma de determinar si una red está infectada es observando los errores que están siendo ocasionados por la red. En los registros individuales que se están utilizando para esta actividad existe un mensaje de error; si este error ocurre repetidamente para varios usuarios separados al mismo tiempo, es probable que la red esté infectada (asumiendo que no hay errores de programación). Si “Illegal User” es el error que se le está arrojando a todos los usuarios sin importar la IP o información que estén utilizando, es señal de que la red ha sido infectada y que por ello no le está permitiendo el acceso a los servicios de forma adecuada.

## Conclusión

La complejidad temporal total de esta actividad es de $O(nlog(n))$, lo cual se debe a la implementación del método heap sort, cuya complejidad temporal también es $O(nlog(n))$. Es importante tomar en consideración que ese algoritmo de ordenamiento fue implementado únicamente para el MaxHeap que se ordena mediante IP's. Existe otro MaxHeap que se implementó en el programa, pero el tamaño de este es mucho menor debido a que sólamente existe un elemento por cada IP única (Son 735 IP's distintas dentro del archivo) y este no se tiene que ordenar, sino que sólamente se extraen las 5 IP's con más repeticiones utilizando .pop() y .top(), cuyas complejidades temporales con O(log(n)) y O(1) respectivamente.

Es interesante la aplicación que se tuvo que hacer del heap para esta actividad e hizo que la importancia de este tipo de estructuras me haya quedado mucho más claro. Sobre todo para la extracción de las 5 IP's con más repeticiones me ayudó a comprender la importancia de la complejidad temporal de los borrados y de las inserciones de las estructuras de datos. Finalmente considero importante mencionar que la implementación de estos métodos me ocasionó problemas al inicio porque no comprendía bien su propósito, pero al realizar la actividad me quedó claro la eficiencia que estos métodos conllevan en situaciones como la que se presentó para esta actividad.

## Referencias:


- *Heap implementation: Push: Pop: Code*. TECH DOSE. (2021, January 26). Retrieved November 6, 2021, from https://techdose.co.in/heap-implementation-push-pop-code/.ç
- Ang, N. (2017, December 10). *Why use binary search tree?* Nick Ang. Retrieved November 6, 2021, from https://www.nickang.com/2017-12-10-why-use-binary-search-tree/#:~:text=The%20main%20reason%20to%20use,data%20points%20contiguously%20in%20sequence.
- Biguenet, J. (2021, May 13). *How and when to implement binary search trees*. LookFar. Retrieved November 6, 2021, from https://www.lookfar.com/blog/2016/07/28/why-binary-search-trees-matter/#:~:text=Implementing%20a%20binary%20search%20tree,less%20than%20another%20element%20(eg.
- *Heapsort*. GeeksforGeeks. (2021, September 14). Retrieved November 6, 2021, from https://www.geeksforgeeks.org/heap-sort/