

## **Actividad Integral estructura de datos linealesAct.2.3**

Autor: David Román Velasco

Fecha: 22/10/2021

Con esta actividad me di cuenta de los algoritmos de ordenamiento y búsqueda son bastante esenciales para el desarrollo de programas que deben de manejar bases de datos o archivos con grandes cantidades de datos, por lo que al contar con estas herramientas facilitamos la modificación y acceso a información específica entre todos los datos que se le ingresan al programa, porque si no resultaría imposible ir buscando los datos que necesitas de uno por uno en archivos de 16,000 líneas, y menos ordenarlos, de ahí a que estos algoritmos sean tan importantes, ya que nos permiten hacer esto de una forma rápida y eficiente, un claro ejemplo es con el desarrollo de la “Act 1.3 - Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales (Evidencia Competencia)”, sin los algoritmos de búsqueda y ordenamiento que vimos en clase, no se hubiera llegado a una solución factible y es muy probable que no se hubiera podido realizar correctamente, ya que se debe manejar una gran cantidad de datos.

### **Complejidad computacional de los diferentes algoritmos:**

#### Búsqueda Secuencial

Según RuneStone (s.f.) esta búsqueda consiste en recorrer el array desde el primer elemento hasta el ultimo de manera lineal hasta encontrar el valor buscado.

Complejidad si el item no está presente:

- Mejor:  $O(1)$
- Promedio:  $O(n)$
- Peor:  $O(n)$

Complejidad si el item está presente:

- Mejor:  $O(1)$
- Promedio:  $O(n/2)$
- Peor:  $O(n)$

Complejidad de búsqueda secuencial de arreglos (es similar, pero se detiene la búsqueda cuando se encuentra un valor mayor al buscado) ordenados tanto si el item está o no presente:

- Mejor:  $O(1)$
- Promedio:  $O(n/2)$
- Peor:  $O(n)$

### Búsqueda Binaria

Según DelftStack (2021), consiste en analizar el valor central del vector ordenado, si el elemento buscado es mayor o menor se va recorriendo por uno de los dos tramos del vector.

Complejidad:

- Mejor:  $O(1)$
- Promedio:  $O(\log_2 n)$
- Peor:  $O(\log n)$

### Orden Burbuja

Según Hidalgo (2001), consiste en ciclar la lista comparando elementos de dos en dos, si el que le sigue es menor al anterior se intercambian

Complejidad:

- Mejor:  $O(n)$
- Promedio:  $O(n^2)$
- Peor:  $O(n^2)$

### Ordena Intercambio

Según Hidalgo (2001), consiste en agarrar un valor inicial e ir comparándolo con el resto del arreglo hasta encontrar su posición final. Eso se hace con cada valor del arreglo.

Complejidad:

- Mejor:  $O(n)$
- Promedio:  $O(n^2)$
- Peor:  $O(n^2)$

### Ordena Merge-Sort

Según CodeMyN (s.f.), divide el arreglo en subarreglos que se subdividirán para poder ordenar el arreglo.

Complejidad:

- Mejor:  $O(n \log n)$
- Promedio:  $O(n \log n)$
- Peor:  $O(n \log n)$

### **Lista doblemente ligada:**

Según la Universidad de Granada (s.f.), estas listas están compuestas de nodos que se apuntan entre sí de la siguiente manera: cada nodo de esta lista apunta a su nodo anterior y elemento siguiente, los que apuntan a los extremos apuntaran a un NULL. Y su complejidad para insertar un nodo en determinada posición o rescatar un elemento de una posición específica es de  $O(1)$ , mientras que para eliminar o encontrar una posición es de  $O(n)$ .

### **Binary Search Tree:**

Es una estructura de datos ordenada con nodos que apuntan a otros dos nodos máximo (derecho e izquierdo), los cuales tienen un dato y llave de búsqueda, estos nodos se ordenan en el árbol en base a su llave de búsqueda, donde los que son menores se ponen a la izquierda de su nodo padre, y lo que son mayores a la derecha de su nodo padre. Dichas estructuras tienen diferentes métodos y

funciones con diferentes valores de complejidad, para las usadas en la evidencia tenemos:

- Searching: Para buscar un nodo, tenemos que pasar por todos los anteriores, por lo que al buscar en el árbol de búsqueda binario el peor caso de complejidad es  $O(n)$  y el tiempo de complejidad es  $O(h)$  donde  $h$  es la altura del árbol. (GeeksforGeeks, 2018)
- Insertion: Para insertar un nodo, tenemos que pasar por los nodos del árbol, por lo que al insertar en el árbol de búsqueda binario el peor caso de complejidad es  $O(n)$  y el tiempo de complejidad es  $O(h)$  donde  $h$  es la altura del árbol. (GeeksforGeeks, 2018)

## Conclusión

El algoritmo de ordenamiento usado fue el de burbuja, sin embargo, nos dimos cuenta de que usar el de merge-sort hubiera sido más factible debido a su menor grado de complejidad, ya que como son 16,000 líneas que ordenar se tarda bastante en ordenar las líneas con el de burbuja.

Si se hubiera realizado con listas doblemente ligadas, debido a su bajo costo de complejidad, el acceso a dicha lista para su ordenamiento hubiera sido más eficiente, por lo que ofrece una mayor ventaja respecto a la complejidad del proceso, sin embargo, una desventaja sería que para convertir la estructura que ya teníamos a una lista doblemente ligada, nos hubiera requerido un poco más de trabajo y tiempo, por lo que se optó por mantener la estructura original de vectores.

El Binary Search Tree, es bastante útil para este tipo de problemas, agiliza los procesos gracias a que cuenta con un nivel menor de complejidad, gracias al uso de apuntadores, a comparación de la manera en que realizamos el programa en las evidencias anteriores, este compiló todo rápida y eficientemente, asimismo nos permite buscar nodos y extraer sus datos, sin perder tiempo. Respecto a cómo podemos determinar si una red está infectada, podemos revisar la cantidad de accesos de las IPs y en qué periodos de tiempo se intentaron los accesos, asimismo si el acceso fue negado, revisar la razón del bloqueo del acceso, de esta manera podemos determinar si un bot está intentando acceder o realizar un ataque, ya que si se intenta conectar una IP muchas veces un periodo de tiempo demasiado corto, es un indicio de ataques de bots, ya que son usados para realizar spam, phishing y **ataque DDoS** con intención de desbordar la página (Kaspersky, s.f.).

## Referencias

CodeMyN. (s.f.). Métodos de Ordenamiento MergeSort en C++. Recuperado de: <https://codemyn.blogspot.com/2019/07/metodos-de-ordenamiento-mergesort-en-c.html>

DelftStack. (2021). Búsqueda binaria. Recuperado de: <https://www.delftstack.com/es/tutorial/algorithm/binary-search/>

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

Hidalgo, J. (2001). Algoritmos de ordenamiento. Recuperado de: <http://conclase.net/c/orden/introduccion>

Kaspersky. (2017). What is a Botnet?. Recuperado de: <https://youtu.be/3BbxUCOFX8g>

Kaspersky. (s.f.). ¿Qué son los ataques DDoS?. Recuperado de: <https://latam.kaspersky.com/resource-center/threats/ddos-attacks>

RuneStone Academy. (s.f.). La búsqueda secuencial. Recuperado de: <https://runestone.academy/runestone/static/pythoned/SortSearch/LaBusquedaSecuencial.html>

Universidad de Granada. (s.f.). “Capítulo 5: Arrays y Cadenas.” Proyecto: Sistema De Ayuda Al C. Recuperado de: [https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap5/f\\_cap56.htm](https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap5/f_cap56.htm)

Universidad de Granada. (s.f.). Listas Doblemente Enlazadas. Recuperado de: <https://ccia.ugr.es/~jfv/ed1/ledi/cdrom/docs/ldoble.html>