Jorge Daniel Cruz Case – A01634536

**Reflexión Actividad integradora 3.4**

Durante la realización de la actividad integradora se nos dio de tarea, similarmente a las actividades integradoras anteriores, ordenar mediante estructuras de datos una bitácora dada con una gran cantidad de datos. Para esto fue necesario la implementación y el uso de algunas de las estructuras y algoritmos vistos en clase.

Para el programa realizado hay que tener un buen entendimiento de los BST (Binary Search Tree) ya que son empleados en el mismo. Es importante que durante la ejecución y uso del programa no ocurran fallas de lógica o desbordamiento de memoria, el uso eficiente de la misma es increíblemente importante en programas como este, dónde se están utilizando una gran cantidad de datos.

Para el correcto entendimiento de los procesos utilizados en el programa, es importante conocer las estructuras del *Binary Search Tree & Max Heaps.*

Según el siempre confiable Geeks for Geeks, el *Binary search tree,* es una estructura de datos que al igual que las *Linked List,* consiste en Nodos organizados de manera eficiente. Pero en este caso, los nodos son organizados en forma de árbol, teniendo raíz, hojas y ramas. Dónde existe un nodo raíz o hoja, dónde aquel nodo o hijo a su izquierda es menor al padre y aquellos hijos a su derecha son mayores al padre. Esta regla se repite a lo largo del mismo.

En cierto punto podemos ver a un BST como una serie de sub-BST con solamente 3 nodos, un padre o una raíz, con sus respectivos hijos a la derecha y a la izquierda, los cuales se repiten con las mismas reglas a través del BST principal.

Los BST se dividen en un conjunto de categorías, los *Heap, AVL*, *Splays*, entre otros. En este caso nos centraremos en el *Heap,* el cual fue el utilizado para la solución de la actividad.

Los *Heaps,* son árboles binarios que un cierto orden dependiendo de cual se está trabajando:

- **Min Heap,** En un min Heap, si vemos al árbol como un conjunto de sub-árboles, el nodo padre siempre es menor a los hijos, siendo el nodo raíz el nodo con el menor valor de todos los nodos en el árbol.
- **Max Heap,** El max Heap, es casi igual a su contraparte, pero inverso, todo nodo padre debe ser mayor a sus hijos, con el nodo raíz siendo el nodo con el mayor valor entre sus hijos.

La eficiencia que se ha estado buscando durante la realización de estas actividades ha avanzado una vez más, esta vez convirtiéndose lo más eficiente que nunca. La ejecución se ha eficientado demasiado desde sus días en un vector, la necesidad de un comando de compilación como “ g++ -std=c++17 -o main -O3 *.cpp” es casi nula, el desbordamiento o escape de memoria se ha erradicado, y el código es más eficiente y limpio que antes.

Pero el propósito de este tipo de actividades no es solamente la búsqueda de eficiencia o el uso de estructuras de datos, si no también de la identificación de ataques cibernéticos hacia una red o una IP.

¿Cómo podemos identificar una red comprometida o infectada por un malware? Si tenemos en cuenta la cantidad de accesos fallidos hacia una IP especifica, podemos asumir que aquellas IPs que cuentan con una cantidad alta de accesos fallidos son aquellas que están potencialmente infectadas o comprometidas, aunque también hay que tener en cuenta errores humanos o potenciales variables ajenas a un malware o a un ataque. Pero por simplicidad, en un caso perfecto, dónde cada humano logra una conexión a la primera, aquellas IPs con mayor cantidad de accesos fallidos son aquellas comprometidas o atacadas.

En un dado caso que se requiera o haya la necesidad de reducir la búsqueda, podemos ver el tipo de error que se presenta en cada acceso fallido, tal vez el malware sabe utilizar diferentes métodos para su conexión, y también comprobar que aquellas conexiones exitosas no hayan sido un malware.

**Referencias**

Geeks for Geeks. (2021). *Binary Search Tree.* Recuperado de:

https://www.geeksforgeeks.org/binary-search-tree-data-structure/#basic

Geeks for Geeks. (2021). *Heap Data Structure.* Recuperado de:

https://www.geeksforgeeks.org/heap-data-structure/

Informatica.uv.es. (2021). *Laboratorio de estructuras de datos*. Recuperado de:

http://informatica.uv.es/iiguia/AED/oldwww/2002_03/Practicas/Practica8/pr_08_2003.html