Luis Cano Irigoyen – A00827178

Act3.4 Reflexión

En situaciones problema de esta naturaleza el utilizar árboles de búsqueda binarios (BST) podría ser la diferencia entre tener una función con complejidad de O (16807) a una con O (282475249). Clasificar cantidades masivas de data resulta ser un trabajo muy lento al utilizar métodos tradicionales de ordenamiento y búsqueda, por lo cual llega a ser necesario el encontrar alternativas que permitan hacer estos procesos en situaciones problemas de esta naturaleza de una manera más eficiente.

Los BST ofrecen eficiencia en almacenamiento de data de una forma ordenada, de manera que se puedan acceder o buscar los elementos guardados de manera rápida. Si nos ponemos a analizar la complejidad de los diferentes algoritmos de ordenamiento y búsqueda que se utilizaron, para ordenar la lista con las 16807 líneas de datos de la bitácora se utilizó un Heap Sort con complejidad de O (n log n), lo cual equivale a O (71017.8); luego, para agrupar los registros por IP con su cantidad de accesos en una nueva lista solo se requirió de una complejidad de O (n), mientras que algún otro proceso que no hubiera utilizado el Heap Sort podría haber requerido una complejidad de hasta O (n^2), igual a O (282475249), lo cual hubiera vuelto nuestro programa casi inservible; y por último, una vez con la lista de accesos convertida a un Heap, para encontrar los 5 elementos con mayor “cantidad” sólo se requirió de correr una instrucción de remove() 5 veces, con una complejidad de O (n) = O (5).

El utilizar un BST definitivamente es de gran importancia en situaciones problemas de esta naturaleza, en donde la diferencia de O(n) y O(n log n) contra O(n^2) es masiva. De igual manera cabe mencionar que es importante el considerar que tipo de BST se debería utilizar: en esta situación problema el uso de un Heap permitió que se realizara un programa eficiente que corriera en alrededor de 10 segundos, siendo los elementos buscados los 5 IPs que más se repetían; pero en situaciones diferentes, dependiendo de lo que se busque, podrían llegar a ser más útiles otros BST, cómo el Splay Tree en situaciones donde se quieran encontrar los nodos accedidos recientemente, o el Red-Black Tree en situaciones donde se inserten y eliminen datos con frecuencia.

En un último punto, en este reto nos preguntamos cómo podríamos determinar si una red está infectada o no, y la elaboración de este programa nos ayuda a ver esto. Para determinar que una red está infectada habría varios parámetros por analizar. El conocer cuantas veces cierto IP ha intentado acceder a la red de manera fallida es definitivamente un buen comienzo par contestar esta pregunta. Analizando estos datos podríamos ver cuáles IPs son los que más veces han fallado en sus registros, y podríamos tomar estos como enfoque de nuestro análisis para descubrir redes infectadas, ya que un IP con pocos errores en sus registros sería más probable por causas de error humano al intentar acceder y no valdría tanto la pena considerarlo. Una vez con estos IPs con más accesos fallidos de los 16807 registros que tenemos, podríamos observar las horas en que intentaron su acceso, en donde muchos intentos en un solo instante podrían ser un síntoma de una red infectada, así como podríamos observar la razón por la cual falló, en dónde muchos intentos fallidos a acceder admin o root podría decirnos más que la red está infectada que el haber tenido muchos intentos fallidos a acceder un guest user.