Emilia Victoria Jácome Iñiguez.

23/10/2020

A00828347

Reflexión de la Actividad 3.4:

Un heap Tree (también denominado como Priority Queue o su traducción al español como Cola Priorizada) es un arreglo de elementos organizados de acuerdo a un criterio de prioridad (ya sea de menor a mayor o viceversa). Esta estructura se caracteriza porque sus operaciones aunque muy parecidas a las de un BST (Binary Search Tree) regular, siempre deben mantener el orden de priorización del árbol. Si el criterio de prioridad es de Mayor a Menor, entonces se lo puede denominar como una Max-Cola mientras que si es de Menor a Mayor su denominación es Min-Cola. Las colas de prioridad se pueden emplear para hallar el k-enésimo elemento de un vector desordenado al colocar todos los elementos del vector en una cola priorizada y siempre reordenando la cola cada vez que uno de los nodos hijos sean mayores o menores que el padre (según corresponda). Algunas características fundamentales de un heap son las siguientes: (Martínez, 2012)

Todos los nodos hojas del árbol solo podrán estar en los dos últimos niveles del árbol y no en las ramificaciones superiores. Es decir, que el llenado del árbol debe realizarse de forma consecutiva, siempre llenando primero los nodos de la izquierda y luego los de la derecha.

Cada elemento padre del árbol deben almacenar un valor que sea ya sea mayor/ igual que o menor que los dos elementos almacenados en sus nodos hijos (tanto izquierdo como derecho).

El menor elemento de un Min-Heap debe estar en su raíz.

El mayor elemento de un Max-Heap debe estar en su raíz.

Un heap con n cantidad de elementos siempre va a tener una altura de $h = \log_2(n+1)$ (Martínez, 2012)

Todos los niveles del árbol deben estar completamente llenos excepto posiblemente el último nivel ya que los elementos de la izquierda deberán llenarse primero antes que los de la derecha.

La aplicación de un Heap Binary Tree es muy común en la creación e implementación de diccionarios. Por otra parte, también permiten representar diferentes secuencias matemáticas tales como la secuencia Fibonacci y al desarrollo de binomios. En adición, permiten resolver eficazmente problemas matemáticos tales como encontrar el N máximo elemento dentro de un arreglo de números, combinar 2 arreglos ya ordenados o ordenar un conjunto de valores casi ordenados con rapidez. Finalmente, otra aplicación significativa corresponde a la graficación de algoritmos tales como el Dijkstra's Shortest Path and Prim's Minimum Spanning Tree. (geeksforgeeks, 2020)

El uso y eficiencia de este tipo de estructuras de datos en una situación problema de esta naturaleza parte del hecho que efectuando una comparación entre todos los algoritmos de ordenamiento, el heap sort ha demostrado ser el más efectivo con una amplio rango de diferencia en el tiempo de ejecución con respecto a los demás algoritmos. Esto se debe a que al crear un heap tree, automáticamente los ordena de manera priorizada en forma de una estructura jerárquica donde únicamente se comparan los padres con los hijos de cada subárbol. Lo cual hace que el proceso de comparación sea mucho más rápido y organizado que al tener una estructura lineal. Y luego, se van añadiendo los elementos del heap tree a una estructura lineal (en el caso del proyecto decidí utilizar una doble linked list por la flexibilidad que brinda el hecho de tener doble apuntadores). Esto se justifica por el hecho de que la complejidad de las operaciones de un binary heap son mucho menores que las de un BST donde tanto inserción, eliminación y ordenamiento tienen una complejidad de O(LogN) y el obtener el máximo o mínimo elemento es de O(1) mientras que un BST sus operaciones tienen una complejidad de O(n). Dado que un heap se implementa con Linked Lists siempre va a existir una ubicación referencial más directa que únicamente recorriendo todos los elementos de un BST hasta encontrar el elemento deseado. Aparte, dentro de un Binary Tree, no se requieren apuntadores extra por lo que no consumen ese espacio de memoria. (GeeksforGeeks, 2018)

¿Cómo podrías determinar si una red está infectada o no?

Para detectar si una red está infectada o no es necesario observar la cantidad de accesos que ha tenido dicha red así como la frecuencia periódica con la que se ha accesado y la cantidad de diferentes puertos dentro de una determinada dirección de IP lo que proveerá información sobre la cantidad de dispositivos que han podido acceder. Para poder observar esta información de manera muy eficaz, se debería realizar un algoritmo similar al que se ha elaborado para esta presente actividad 3.4 con los registros de acceso a la red

con los cuales se deberá utilizar uno de los métodos de ordenamiento más efectivos de heapsort para asegurar la efectividad de este método al momento de ordenar los IP´s y realizar un conteo organizado de la cantidad de repeticiones de cada uno de los registros de acceso. Con ello, se podría almacenar los resultados dentro de una lista que fácilmente pueda desplegar los resultados de los IPS mayormente accesados y que automáticamente los pueda identificar como una posible infección a dicha red. Por IP, se está refiriendo al protocolo de Internet propio de una red, es decir, la dirección única que identifica a una red desde dispositivos externos. La gran ventaja de haber utilizado distintas estructuras dentro de una estructura de Heap Tree fue que, al contar con más de 16000 registros, el heap sort me permitió desarrollar el proceso de ordenamiento mucho más rápido y poder organizar los datos según diferentes criterios, ya sea solamente el puerto, la dirección IP completa o la dirección IP. El uso de 3 estructuras similares con la sobrecarga de operaciones para cada criterio fue clave para organizar los datos y manejarlos de acuerdo a las necesidades sin modificar mi estructura base de Heap, Double Linked List y Node manteniendo el formato de template. Personalmente, encontré al uso de heap muy simple, sencillo y eficaz de implementar lo cual fue una ventaja al momento de programar ya que la mayoría de los algoritmos fueron muy similares a los de un BST regular siempre procedidos por el método de reorganización hacia arriba o hacia abajo lo cual siempre mantenía la base de la estructura con un nivel de ordenamiento alto, lo que permite que los algoritmos de ordenamiento se ejecuten de manera instantánea partiendo de una lista de elementos quasi-ordenados.

Referencias:

GeeksforGeeks. (13 de Enero de 2018). *Why is Binary Heap Preferred over BST for Priority Queue?* Recuperado el 24 de Octubre de 2020, de GeeksforGeeks: https://www.geeksforgeeks.org/why-is-binary-heap-preferred-over-bst-for-priority-queue/

geeksforgeeks. (8 de Septiembre de 2020). *Binary Heap*. Recuperado el 24 de Octubre de 2020, de geeksforgeeks.org: https://www.geeksforgeeks.org/binary-heap/

Martínez, C. (2012). *Algorítmica: Heaps y heapsort*. Recuperado el 24 de Octubre de 2020, de Universidad Politécnica de Catauña: http://algorithmics.lsi.upc.edu/docs/pqueues.pdf