Tecnológico de Monterrey

Actividad 2: Investigación de Mejores Prácticas Organizacionales.

Yael García Morelos | A01352461 | Campus Guadalajara

Programación de Estructura de Datos y Algoritmos Fundamentales. (Gpo 850)

Profesor. Eduardo Arturo Rodríguez Tello

14 de mayo del 2025

En esta Actividad Integradora se desarrolló un algoritmo donde se trabajó con una bitácora la cual contenía una gran cantidad de datos. Durante esta actividad se emplearon los conocimientos adquiridos relacionados con estructura de datos, específicamente un árbol *heap*. El objetivo principal de esta actividad fue desarrollar un algoritmo el cual no comprometa su eficiencia en tiempo y recursos.

Durante la Actividad Integradora se fueron presentando distintos desafíos. Analizando desde las primeras indicaciones, pasar los datos de la "*bitácora*" a un vector no presentó ningún problema. De igual manera, la implementación de *Heap Sort* se pudo realizar con facilidad, pues ya se conocía su estructura y funcionamiento. Para poder implementar correctamente este método de ordenamiento, únicamente fue necesario *parsear* las direcciones IP con la finalidad de únicamente tomar en consideración la IP sin puntos ni el puerto.

El único problema que se llegó a presentar fue uno relacionado a la lógica, pues durante la planeación, en el pseudocódigo se tuvo un error en los signos para la comparación de datos, sin embargo, en poco tiempo se detectó el error y fue posible seguir adelante sin problema alguno.

Se considera, que la complejidad aumentó a partir del cuarto punto, contabilizar las repeticiones de cada IP de manera eficiente, en un principio se consideró realizar un método iterativo, sin embargo, por la gran cantidad de datos que se tiene, no era una alternativa eficiente, por otro lado, al realizar una breve investigación, se identificó que la implementación de un *Unordered_map* era una alternativa que no compromete el funcionamiento del algoritmo por su uso de *Hash Tables,* por esta razón, se decidió su implementación para el conteo, además de no tener una implementación complicada.

Un error que se detectó fue que el *heap* utilizado no era un *binary heap,* gracias a este error en algún momento se consideró rehacer la clase o implementar una nueva, sin embargo, se descubrió una alternativa que ahorro tiempo y recursos, la implementación de "*priority_queue*", gracias a esto, no fue necesario crear una clase nueva, además de tener una usabilidad sencilla.

Con el uso de "*priority_queue*", se pudo realizar un *max_heap,* una gran alternativa para identificar cuales fueron las IP con más accesos pues al tener un *max_heap* en automático el

dato con mayor valor se posicionará en la cabeza del árbol, e implementando *top()* y *pop()* sustraer los 10 con una mayor iteración fue sencillo.

Utilizando esta misma lógica se implementó la función para encontrar la dirección IP con una cantidad de accesos mayor o igual a tres. Como se debía utilizar la estructura del *max_heap* y por lógica se conocía que una IP con las características anteriores estaría al final del árbol y recorrerlo de manera iterativa podría resultar en *O(n^2)*, pues realmente los datos no se encontraban ordenados como en el archivo "*bitácora_ordenada.txt*" y recorrer todo el árbol y además realizar las comparaciones para saber si es el menor dentro de los parámetros establecidos comprometería la eficiencia del programa.

Con esto en mente, se optó por realizar un *min_heap,* de esta manera se invierte el árbol, el de menor cantidad se encuentra en la raíz y es más fácil de sustraerlo para poder identificar si cumple con las características, en caso de no hacerlo, implementando *pop()* se puede eliminar fácilmente y seguir con el segundo menor, de esta manera, seguimos implementando un *binary heap,* y el algoritmo puede ser fácilmente manipulado si se desea conocer más IP en vez de una. Realizar el árbol tuvo una complejidad computacional de O(n log n), mientras que sustraer el dato deseado, fue de *O(1)*, pues como ya se explicó anteriormente, este se encontraba en la raíz del árbol.

Pseudocódigo:

```
FindIPMin (ipCount)
Creación de Min_Heap con (int, string)
Para cada (ip, count) en ipCount
      Insertar (ip, count) en Min_Heap
Si Min_Heap no esta vacío
      Sustraer elemento en cabeza
      Print "IP " + ip + " – Accesos " + count
      Eliminar cabeza
Fin
```

En conclusión, esta actividad fue desafiante por la gran cantidad de datos utilizados y por las adaptaciones que se realizaron a las funciones para que se puedan utilizar con la clase de

Registro, por otro lado, se pudo aprovechar para reforzar conocimientos y aplicarlos en un contexto distinto y práctico.