Act-Integradora-3

Conceptos básicos y algoritmos fundamentales

Programación de estructuras de datos y algoritmos fundamentales
(Gpo 850)

Profesor
Eduardo Arturo Rodríguez Tello

Nombres:

Federico Melo B. A00833536

13/Mayo/2025

A. **Analiza la importancia y eficiencia del uso de estructuras de datos jerárquicas en una situación problema de esta naturaleza.**

En la situación problema, las estructuras de datos jerárquicas como los Binary Heaps, tienen un papel importante gracias a su capacidad para gestionar las prioridades de manera eficiente. En este caso usamos un MaxHeap para identificar rápidamente las IPs con mayor cantidad de accesos y un MinHeap para encontrar la IP con menos número de accesos pero que cumpla algunos criterios. La ventaja de las estructuras jerárquicas es que permiten acceder al elemento máximo o mínimo de manera constante (O(1)) y mantener el orden con inserciones y eliminaciones en tiempo O(log n). Esto las hace que sean ideales para situaciones donde se necesita procesar o priorizar elementos de acuerdo a su frecuencia.

B. **Reflexiona por qué en la solución de este reto es preferible emplear un Binary Heap y no un BST, basa tu reflexión en la complejidad temporal de las operaciones de ambas estructuras de datos.**

Utilizar un Binary Heap en lugar de un Binary Search Tree (BST) es mejor por las características de eficiencia y simplicidad que tiene el Binary Heap. Pero los dos son estructuras jerárquicas que nos permiten gestionar elementos de manera ordenada, el Binary Heap tiene la capacidad de realizar las operaciones críticas de inserción y eliminación en un tiempo de O(log n). Esto es porque el Binary Heap garantiza que el elemento máximo o el elemento mínimo siempre va a estar en la raíz, permitiendo acceso constante O(1) a estos elementos. Y del otro lado, un BST puede tener operaciones en O(log n) pero si el árbol no está balanceado, el tiempo puede llegar a ser O(n). Esto hace su rendimiento impredecible. También, el Binary Heap se puede implementar usando un arreglo, mientras que el BST necesita nodos enlazados. Entonces, el Binary Heap ofrece una solución más rápida.

C. **Analiza la complejidad computacional de las operaciones básicas de la estructura de datos empleada en tu implementación (push, pop, getTop, etc) y cómo esto impacta en el desempeño de tu solución**

El Binary Heap es muy eficiente gracias a que sus operaciones principales están diseñadas para ser rápidas y predecibles. La inserción de un nuevo elemento (push) se realiza en O(log n) ya que el elemento se inserta al final del arreglo y luego se ajusta hacia arriba (heapifyUp) para mantener la propiedad del heap. De la misma manera, la eliminación del elemento máximo o mínimo (pop) también opera en O(log n), porque el elemento en la raíz se reemplaza y se ajusta hacia abajo. El acceso al elemento de mayor o menor prioridad (top) es todavía más eficiente, operando en O(1), este siempre está en la raíz del heap. Pero sin embargo, una limitación del Binary Heap es qu el buscar un elemento específico requiere recorrer todo el arreglo, resultando en un tiempo de O(n). Estas características hacen que el Binary Heap sea ideal para el problema actual, donde se necesita manejar grandes cantidades de IPs y priorizar aquellas con más accesos o identificar las menos frecuentes de manera eficiente.

D. **Presenta el pseudo-código del algoritmo empleado para buscar la IP con menor número accesos en un Binary Heap (del punto 6), así como el análisis de su complejidad temporal.**

```
1. Si longitud(A) = 0 Entonces
2.    retornar ("", 0)
3. FinSi
4. inicio ← floor(longitud(A) / 2) // Índice de la primera hoja
5. bestCount ← longitud(A) + 1 // Valor sentinela alto
6. bestIP ← ""
7. Para i desde inicio Hasta longitud(A) – 1 Hacer
8.    count ← A[i].first
```

```
9.    ip ← A[i].second
10.   Si count ≥ minCount Entonces
11.      Si count < bestCount Entonces
12.         bestCount ← count
13.         bestIP ← ip
14.      FinSi
15.   FinSi
16. FinPara
17. Si bestCount = longitud(A) + 1 Entonces
18.    retornar ("", 0)
19. FinSi
20. retornar (bestIP, bestCount)
```

El algoritmo busca la IP con el menor número de accesos que cumpla con el mínimo de 3, recorriendo solo las hojas del MaxHeap. Esto garantiza eficiencia porque las hojas contienen los valores más bajos. Su complejidad es O(u), donde u es el número de hojas del heap. Esto evita la necesidad de un segundo heap y hace que haya un mejor rendimiento del programa.

Referencias

Heap Data Structure. (n.d.). Programiz.

https://www.programiz.com/dsa/heap-data-structure

Heap Data Structure Overview. (n.d.). Tutorialspoint. Retrieved May 13, 2025, from

https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

Jain, S. (2025, April 26). *Heap Data Structure*. GeeksforGeeks. https://www.geeksforgeeks.org/heap-data-structure/

std::priority_queue - Priority queue. (n.d.). CPlusPlus.com. https://cplusplus.com/reference/queue/priority_queue/