Tecnológico de Monterrey

Campus Monterrey

Materia

Programación de estructuras de datos y algoritmos fundamentales (Gpo 608)

Tarea

“Evidencia 3- Actividad Integral de Estructuras de Datos Jerárquicas”

Estudiante

Oscar Cardenas Valdez

Profesor

David Cantu

Fecha

Nov 9

Reflexión sobre el uso de estructuras de datos jerárquicas en la actividad

Introducción

En esta actividad, se hace uso de varias estructuras de datos jerárquicas, específicamente **Heap** y **Árbol Binario de Búsqueda (BST)**, para procesar y manipular registros de un archivo de log. La tarea consiste en ordenar datos, encontrar las 10 IPs con más registros, realizar búsquedas eficientes de IPs y generar salidas en archivos. Este ejercicio demuestra cómo el uso de las estructuras adecuadas puede mejorar tanto la eficiencia como la claridad en el manejo de grandes volúmenes de datos. En esta reflexión, se analizará la importancia de estas estructuras y su eficiencia, comparando diferentes algoritmos de ordenamiento y búsqueda que pueden aplicarse a este tipo de problemas.

Estructuras de Datos Jerárquicas

Heap

El **Heap** es una estructura de datos jerárquica, que generalmente se utiliza para implementar **colas de prioridad**. El Heap se organiza como un árbol binario completo en el que cada nodo tiene un valor mayor (MaxHeap) o menor (MinHeap) que sus hijos. Esta propiedad permite realizar operaciones eficientes como inserciones y extracciones en tiempo logarítmico.

En la actividad, se utilizó el **MaxHeap** para ordenar los registros de mayor a menor y el **MinHeap** para ordenar los registros de menor a mayor. Esto es particularmente útil porque la propiedad de los heaps garantiza que siempre podemos obtener el valor máximo o mínimo en tiempo constante, y la ordenación se realiza en tiempo **O(n log n)**, lo cual es más eficiente que el uso de algoritmos de ordenación más simples, como el burbujeo o la inserción.

Aplicación en la actividad:

- **HeapSort** fue utilizado para ordenar los datos por IP, fecha y hora. HeapSort tiene la ventaja de ser estable en términos de complejidad de tiempo **O(n log n)**, y es eficiente al tratarse de grandes volúmenes de datos.
- La utilización de **MaxHeap** y **MinHeap** permite ordenar los datos en un formato eficiente, garantizando que el proceso de ordenamiento y generación de archivos de salida se realice sin necesidad de iterar manualmente sobre los datos, lo que optimiza el rendimiento.

Árbol Binario de Búsqueda (BST)

El **Árbol Binario de Búsqueda** es una estructura de datos jerárquica que organiza los datos de manera que para cada nodo, todos los valores de su subárbol izquierdo son menores que el nodo y todos los valores del subárbol derecho son mayores. Esto permite realizar búsquedas, inserciones y eliminaciones de manera eficiente en tiempoReflexión sobre el uso de estructuras de datos jerárquicas en la actividad

En este caso, el BST se utiliza para almacenar las IPs del archivo de log, lo que facilita la búsqueda rápida de una IP específica y contar cuántas veces aparece en el archivo. Esto es mucho más eficiente que realizar una búsqueda lineal en un vector o lista.

Aplicación en la actividad:

- La búsqueda de IPs dentro del árbol binario de búsqueda es mucho más eficiente que una búsqueda lineal. Esto se debe a que en un árbol balanceado, la búsqueda se realiza en un tiempo **O(log n)** en lugar de **O(n)**. Esto optimiza el rendimiento de la aplicación cuando el archivo de log contiene un número elevado de registros.
- Adicionalmente, la estructura BST permite realizar una búsqueda rápida para verificar si una IP está presente en el archivo, y al estar balanceada, no existe la necesidad de recorrer todos los elementos como en un arreglo o lista.

Comparación de Algoritmos de Ordenamiento y Búsqueda

Algoritmos de Ordenamiento

En esta actividad, se han utilizado dos tipos de **HeapSort**, uno con **MaxHeap** para ordenar de mayor a menor y otro con **MinHeap** para ordenar de menor a mayor. Tambien se uso un tercer tipo llamado **CountHeap**, que ordena mediante frecuencias, no ips

HeapSort: Con una complejidad **O(n log n)**, HeapSort es más eficiente que algoritmos de ordenamiento de complejidad **O(n²)**, como el algoritmo de **BubbleSort** o **InsertionSort**. Además, no requiere espacio adicional como **MergeSort**, que tiene una complejidad espacial **O(n)**.

Algoritmos de Búsqueda

1. **Búsqueda Binaria**: Utilizando un **BST**, la búsqueda de una IP específica se realiza en **O(log n)**, lo que es considerablemente más rápido que una búsqueda secuencial (que tiene una complejidad de **O(n)**).
2. **Búsqueda Secuencial**: Si se tratara de una lista o un arreglo sin estructura jerárquica, tendríamos que buscar la IP línea por línea, lo que sería mucho más lento y menos eficiente que el uso de un BST.

Complejidad Computacional

La eficiencia de cada estructura de datos depende del tipo de operación que estamos realizando. A continuación se presentan las complejidades de tiempo de las principales operaciones en las estructuras utilizadas:

- **Heap** (MaxHeap y MinHeap):
    - Inserción: **O(log n)**
    - Extracción de la raíz (máximo o mínimo): **O(log n)**
    - Ordenamiento (HeapSort): **O(n log n)**
- **Árbol Binario de Búsqueda (BST)**:
    - Búsqueda: **O(log n)** (en un árbol balanceado)
    - Inserción: **O(log n)** (en un árbol balanceado)
    - Eliminación: **O(log n)** (en un árbol balanceado)

Conclusión

En conclusión, el uso de estructuras de datos jerárquicas como **Heap** y **Árbol Binario de Búsqueda (BST)** en esta actividad demuestra ser muy eficiente para manejar, ordenar y buscar datos de manera rápida y efectiva. El **HeapSort** y el **BST** no solo mejoran el rendimiento de la aplicación, sino que también permiten optimizar la gestión de grandes cantidades de datos como la gran cantidad de logs en el archivo TXT. Además, la comprensión y uso adecuado de estas estructuras es clave para resolver problemas de programación que requieren eficiencia tanto en el tiempo como en el espacio.