Actividad integradora 3

Arturo Gómez Gómez – A07106692

Reflexión sobre el uso de estructuras de datos jerárquicas en la gestión de logs

En el procesamiento de grandes volúmenes de datos, la elección de una estructura de datos adecuada es clave para la eficiencia y escalabilidad. En el contexto de la ordenación y análisis de registros en una bitácora de logs, el **Binary Heap** sobresale como una opción óptima frente a otras estructuras como el **Binary Search Tree (BST)**. Su eficiencia en la gestión de prioridades y el acceso rápido a los elementos más relevantes lo convierten en una herramienta fundamental para la detección y análisis de patrones de acceso a direcciones IP.

Importancia del Binary Heap en el análisis de registros

El principal objetivo de esta implementación es mantener los registros organizados según la frecuencia de aparición de direcciones IP, permitiendo una identificación rápida de las que tienen mayor o menor recurrencia. Para lograrlo, el Binary Heap proporciona beneficios notables:

- **Priorización eficiente**: La IP con más apariciones se encuentra siempre en la raíz del heap, facilitando su acceso en (O(1)).
- **Inserción y eliminación rápidas**: Con una complejidad de (O(\log n)), permite la actualización dinámica de datos sin comprometer el rendimiento.
- **Adaptabilidad a grandes volúmenes de información**: A medida que se registran nuevas IPs, el heap sigue operando eficientemente sin necesidad de balances costosos.

En contraste, el **BST**, aunque útil para búsquedas rápidas dentro de conjuntos ordenados, no resulta óptimo para este caso. Su inserción y eliminación pueden alcanzar (O(n)) en el peor caso si el árbol se desbalancea, lo que afecta el desempeño general del sistema cuando se manejan grandes cantidades de registros.

Análisis de la complejidad computacional de las operaciones básicas

Cada operación realizada en la estructura del Binary Heap tiene un impacto directo en la eficiencia de la solución:

1. **insert (Inserción de un elemento) – (O(\log n))**
    - Se agrega un nuevo nodo y se reorganiza la estructura mediante *heapify*, lo que requiere hasta (\log n) pasos según la altura del heap.
2. **heapSort (Ordenamiento del heap)O(n log n)**
    - Ordena los elementos del heap utilizando el algoritmo heapsort
3. **heapify (Reordenamiento del heap desde un índice dado) – (O(\log n))**
    - Esencial para mantener el heap estructurado, se ejecuta en tiempo logarítmico.
4. **findLeastRecurringIpAboveThreshold (Búsqueda de la IP con menor recurrencia sobre un umbral definido) – (O(m))**
    - Se realiza una única iteración sobre las direcciones IP únicas, asegurando eficiencia lineal.

El uso de estas operaciones permite que la solución mantenga un rendimiento estable incluso en escenarios con grandes cantidades de registros. Esto es crucial para el análisis de tráfico en sistemas de monitoreo y seguridad informática.

Pseudocódigo del algoritmo para encontrar la IP con menor número de accesos

El algoritmo recorre las direcciones IP únicas almacenadas, compara su número de accesos y devuelve la que tiene la menor recurrencia por encima del umbral especificado. Su **complejidad es (O(m))** donde m es el número de IPs únicas, lo que garantiza una ejecución rápida y eficiente.

```
pair<string, int> BinaryHeap::findLeastRecurringIpAboveThreshold(int threshold) {
  if (ipAccessCount.empty()) {
    countIpAccesses();
  }

  string leastRecurringIp = "";
```

```
    int minCount = INT_MAX;

    for (const auto& entry : ipAccessCount) {
        if (entry.second >= threshold && entry.second < minCount) {
            minCount = entry.second;
            leastRecurringIp = entry.first;
        }
    }

    if (leastRecurringIp.empty()) {
        cout << "No IP found with access count >= " << threshold << endl;
        return make_pair("", 0);
    }

    cout << "\nIP with least recurrence >= " << threshold << " accesses: "
         << leastRecurringIp << " - " << minCount << " accesses" << endl;

    return make_pair(leastRecurringIp, minCount);
}
```

Conclusión

El uso del **Binary Heap** en la gestión de registros de una bitácora de logs proporciona una solución eficiente y escalable. Su capacidad para manejar grandes volúmenes de datos con operaciones de inserción, eliminación y acceso de manera logarítmica garantiza un procesamiento ágil. En comparación con otras estructuras como el **BST**, el Binary Heap destaca en la gestión de prioridades y organización dinámica de los registros, lo que lo convierte en la opción ideal para este tipo de problemas.