Programación III

TUDAI 2017

Informe TPE

Etapa 2

Integrantes:

* Zelenka Eugenia
* Mazza eloy

**Introducción**

En base a la primera etapa, se pidió disminuir la complejidad computacional mediante la implementación de algún método que ordene la estructura y otro de búsqueda logarítmica. Eligiendo una de las estructuras utilizadas en la etapa 1.

**Resolución**

Se eligió la estructura del arreglo, por lo tanto se utilizaron las clases de la aplicación número uno de la primer etapa, con las modificaciones necesarias.

Con el arreglo previamente cargado con los datos, decidimos realizar un ordenamiento con MergeSort por su complejidad computacional (O(n.logn)).

Para la búsqueda utilizamos el método de búsqueda binaria Divide y Conquista, su complejidad varía entre O(n) y O(n.logn).

**Análisis de tiempos**

Debido a que el realizar el ordenamiento en tres millones de usuarios demoraba una gran cantidad de tiempo, se decidió evaluar las operaciones de ordenamiento y búsqueda solo con un millón y quinientos mil usuarios.

|  |  |  |
| --- | --- | --- |
| Tiempo Total | 500.000 usuarios | 1.000.000 usuarios |
| Búsqueda Divide y Conquista | 24.737.628 ns | 31.095.085 ns |
| Ordenamiento MergeSort | 161.338.994.100 ns | 626.419.137.832 ns |
| Búsqueda lineal (Sin ordenar) | 71.500.060.142 ns | 131.801.758.098n s |

**Conclusiones:**

Observando este cuadro con los resultados obtenidos, podemos darnos cuenta que la búsqueda utilizando una estrategia de “Divide y Conquista”, en un arreglo previamente ordenado produce unos tiempos de búsqueda enormemente menores a la búsqueda lineal en arreglos sin ordenar. Sin embargo el inconveniente es que

para una gran cantidad de usuarios, realizar el ordenamiento es muy costoso computacionalmente, tardando incluso más que la búsqueda lineal sin ordenar.

Por lo tanto llegamos a la conclusión que ordenar antes de buscar obtiene mucho mejores resultados pero solo si es implementada en cantidades relativamente pequeñas de datos. Para cantidades grandes de datos, es más conveniente no realizar el ordenamiento (Perdiendo obviamente la capacidad de implementar un algoritmo de “Divide Y Conquista”).

Analizamos el comportamiento de la estructura con respecto a los escenarios:

1. Pocas altas de usuarios y muchas verificaciones de existencia.
2. Muchas altas de usuarios y pocas verificaciones de existencia.

La estructura elegida junto con los algoritmos tienen un mejor desempeño, y mejor costo, en el escenario a), debido a que el algoritmo de ordenamiento es muy costoso. Y el algoritmo de búsqueda no tiene inconvenientes.

Se podría haber implementado el MergeSort de manera iterativa, disminuyendo así el tiempo.

Se podría utilizar un árbol binario, el inconveniente de esta estructura es que generalmente están desbalanceados, lo que producen un costo computacional mayor a la búsqueda binaria. Por lo tanto sería ineficiente para ambos escenarios, se podrá evaluar el caso de balancear antes, convirtiéndolo en una estructura más eficiente.

Para balancear un árbol se podría utilizar algunas restricciones como por ejemplo su altura, debido que un árbol perfectamente balanceado tiene como altura lg(n+1), o bien se podría verificar que los nodos tengan dos hijos y los niveles estén en su mayoría completos. Otro balanceo podría ser por peso, si para cada nodo, su subárbol izquierdo y derecho difieren en a lo sumo en uno.