Héctor Gutierrez

A01253031

ReflexAct3.4

Los árboles binarios de búsqueda (BST) son estructuras de datos fundamentales en la ciencia de la computación, particularmente útiles para garantizar la eficiencia algorítmica debido a la manera en que están estructurados y operan. (Rastogi, 2023)

Un punto clave sobre la importancia y eficiencia de los BST es que permiten un acceso rápido a los elementos, especialmente si el árbol está equilibrado. La búsqueda, inserción, y eliminación tienen una complejidad de tiempo logarítmica en el caso promedio, lo que es bastante eficiente. (Exposito Lopez et al., s. f.)

Otro punto importante es que los BST mantienen los datos en un orden específico, lo que facilita las operaciones de búsqueda y ordenación. La implementación de tablas de símbolos usando BST combina la flexibilidad de inserción en listas enlazadas con la eficiencia de búsqueda en un array ordenado. (Wayne, n.d.)

Debido a su estructura de nodos enlazados e implícita ordenación de llaves, los BST permiten operaciones de actualización simples y efectivas, como inserciones y eliminaciones. (Mustapha Meguellati & Eddine Zegour, 2021)

Los BST alcanzan su máxima eficiencia cuando están equilibrados. Un BST no equilibrado no es más eficiente que una lista enlazada regular. Los árboles autoequilibrados ajustan su estructura para mantener una altura logarítmica y así garantizar la eficiencia de las operaciones. (Mallawaarachchi, 2021)

En el código se está utilizando un BST para almacenar y procesar datos de un archivo. A través del recorrido en orden del árbol, se pueden realizar operaciones específicas en cada nodo, lo que puede ser útil para analizar o procesar los datos de una manera particular. El código no se está utilizando las capacidades de búsqueda eficiente del BST, lo que podría ser una oportunidad para mejorar la eficiencia del código.

La función insertNode es responsable de insertar un nuevo nodo en el BST. La complejidad de tiempo de esta operación depende de la altura del árbol. En el peor caso, cuando el árbol está completamente desequilibrado (es decir, cada nodo sólo tiene un hijo), la complejidad de tiempo es O(n), donde n es el número de nodos en el árbol. En el mejor caso, cuando el árbol está perfectamente equilibrado, la complejidad de tiempo es O (log n).

La función inOrderTraversal recorre todos los nodos del árbol en orden inorden. La complejidad de tiempo de esta operación es O(n), ya que visita cada nodo exactamente una vez.

En resumen, los BST proporcionan una plataforma robusta y eficiente para manipular y acceder a datos de manera estructurada y ordenada, lo que es crucial en muchas situaciones de resolución de problemas. Esta estructura de datos es especialmente útil en escenarios donde el rendimiento y la eficiencia en el acceso y manipulación de datos son críticos para la solución efectiva de problemas.

Referencias:

- Rastogi, A. (2023, August 21). Exploring the benefits of using a binary search tree for algorithmic efficiency. *Medium*. https://medium.com/@akshat28vivek/unlocking-the-benefits-of-binary-search-trees-for-algorithmic-efficiency-a19815fc0c21
- Wayne, R. S. a. K. (n.d.). *Binary search trees*. https://algs4.cs.princeton.edu/32bst/
- Mustapha Meguellati, F., & Eddine Zegour, D. (2021, 27 diciembre). A survey on balanced binary search trees methods. IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9678439
- Mallawaarachchi, V. (2021, December 13). Self-Balancing Binary Search Trees 101 - towards Data science. Medium. https://towardsdatascience.com/self-balancing-binary-search-trees-101-fc4f51199e1d
- Exposito Lopez, D., García Soto, A., Gomez, M., & Jose, A. (s. f.). ARBOLES-B. https://ccia.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B.htm