Discussion Board

Carlesha Hayes

[Institutional Affiliation(s)]

Author Note

B-tree: Implementation and Examples

 Discussion Board

B-Tree is a type of self-balancing search tree like AVL and Red-Black Trees. It is used when we need to manage a huge amount of data and there is a lack of main memory. The idea behind the use of B-Tree is to reduce the incidents of disk accesses. The “height” of the B-Tree is kept low by putting maximum keys, or “branches”. This way the disk access is reduced compared to the use of other methods like AVL tree or Red-Black Tree.

 A B-Tree can be implemented using a variety of methods. The most favored among the programmers and the computer pundits is the method of using C++ programming language (D Malhotra, 2019). The reason many scholars prefer this language due to its range and versatility. Many programmers even go as far as to declare it a "middle-level programming language." In the B-Tree, several internal non-leaf nodes have a varying number of child nodes attached to them with some pre-defined range. As the range of child nodes permitted have a predefined range, internal nodes can be split or joined. The upper and the lower bounds on the number of child nodes are fixed, but for a particular implementation. Take the example for a 2-3 B-Tree. As the name specifies, each internal node may have 2 or 3 child nodes.

 B-trees have several advantages over different implementations when the time required to access the data often surpasses that of actually accessing the data (Popov, 2017). For one, each node can be assigned to multiple operations that usually helps multiple users to access the required information simultaneously. However, it should be noted that the number of child nodes strictly depends upon the amount of information that must be stored in these nodes. While a 2-3 B-Tree may be easier to explain, practical B-Tree using secondary storage always needs a large number of nodes to improve their performance.

 Heap data structure is also a type of data tree (Jinn-Pean Lin, 2019), but there are still some noteworthy differences between a B-Tree and a Heap. Heap is a special case of a balanced binary structure where the root-node key is often compared with the child, or subsequent structure and arranged consequently. They are usually implemented with an implicit heap data structure. This structure consists of an array where a specific parent and children relationship is defined indirectly. One disadvantage of this system is that heap gets imbalanced by frequent deletions and additions and therefore they have to balance accordingly.

 Different types of heaps implement their operations differently. But, the process of insertion of data is almost the same. It is done by the addition of new elements at the end of any first available free space. This process often violates the structure of the heap that is often needed to be reestablished. Also, the root can be deleted by deleting the target root and selecting the child node below it to rebalance the whole structure. The programming language C++ is the preferred language in the case of Heap Structures as well (D Malhotra, 2019).

# References

D Malhotra, N. M. (2019). *Data Structures and Program Design Using C++.* Stylus Publishing.

Jinn-Pean Lin, J. L. (2019). Efficient Heap Data Management on Software Managed Manycore Architectures. *32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems.* IEEE.

Popov, A. (2017). An Introduction to MISD Technology. DOI:10.24251/HICSS.2017.119