B-Tree

Lawrence Cabbabe, Aaron Levesque, Ben Uthoff, Sam Zhong

4 December, 2023

Table of Contents:

Introduction to B-Tree:

B-Trees are a widely used data structure that is designed to efficiently organize and store large amounts of data while facilitating quick search, insertion, and deletion operations. B-tree was introduced by Rudolf Bayer and Edward M. McCreight in 1972, and it is well-suited for managing data in storage systems such as databases and file systems.

The main feature of a B-tree is its balance, which ensures that the tree remains symmetrical and even, and prevents nodes going down the same side. As a result, B-tree offers a more reliable and consistent performance since even when the size grows, it stays relatively balanced. The primary advantage of B-tree is its ability to optimize disk I/O operations; it minimizes the number of disk accesses required for search operations which makes it well-suited for instances where data is stored on external storage devices. The efficiency provided by B-tree is a big advantage for accessing a large amount of data while maintaining the performance.

The structure of a B-tree is defined by its order, which determines the maximum number of children a node can have. Each node in a B-tree contains a certain number of keys and pointers to its children. The keys are arranged in ascending order within each node, facilitating efficient search operations. This hierarchical arrangement allows B-trees to achieve logarithmic time complexity for search, insertion, and deletion operations.

To conclude, B-tree is a fundamental data structure that plays a crucial role in the design and implementation of high-performance storage systems, databases, and file systems. The balancing and disk access aspects make B-tree very efficient for managing large datasets and supporting rapid data retrieval and modification.

Introduction to Project:

Methods:

As mentioned in the introduction, a B-tree is commonly used in databases and file systems to store and manage large amounts of data; it is designed to provide efficient searching, insertion, and deletion operations. Its ability to maintain its balance with larger datasets make it efficient. The structure of a B-tree consists of nodes, and each node has multiple keys and pointers to child nodes; the number of keys in each node is between a minimum and a maximum value, which ensures the balance in the tree. The order of a B-tree is the maximum number of children each node can have, and a B-tree of order **t** will have at most **2t - 1** keys and at least **t - 1** keys in each non-root node.

The structure starts with a root node which may have as few as one key or may have multiple keys. To insert a key into the B-tree, it starts at the root and traverses the tree to find the appropriate position for the key, if the node is not full, insert the key into the node in its correct position, and if the node is full, split the node into two, and promote the middle key to the parent node, and repeat the process with parent node if it is needed. When a node is split, it involves redistributing the keys and pointers to form two nodes; the middle key is moved up to the parent node, the keys to the left of the middle key stay in the original node, and the keys to the right of the middle key moves to the new node; the pointers adjusts accordingly. When a key is promoted to the parent, it is inserted into the parent node, and if the parent node is already full, it is split similarly, and the process may propagate up the tree; If the root node is split, a new root is create, and the height of the tree increases by 1. There are a lot of steps when inserting

a value into the B-tree, because there are many factors to be considered to keep the tree balanced.

Another key feature in B-tree is deletion, which is used to remove a key from the tree. To delete a key, it starts at the root and traverses the tree to find the key; when the key is found, if the key is a non-leaf node, it finds its predecessor or successor in the tree, and replaces the key with its predecessor or successor and then deletes the predecessor or successor.

As we see with deletion and insertion, there are a lot of steps that could cause a split or a merge; when a split or a merge happens, the pointers also need to be adjusted accordingly to maintain the tree structure. Maintaining the balance property of the B-tree is crucial because it ensures that the depth of the tree remains relatively controlled.

Implementation:

Insert:

Search:

Contribution:

Group Member	Contribution
Lawrence Cabbabe	
Aaron Levesque	
Ben Uthoff	
Sam Zhong	

Conclusion:

Work Cited:

"B Tree in Data Structure: Learn Working of B Trees in Data Structures." EDUCBA, 14 June 2023, www.educba.com/b-tree-in-data-structure/