ASHESI UNIVERSITY

JOSEPH PRINCE-AGBODJAN

99612021

A1: Managing Data Documentation

ABDUL WASEY

BENEDICT KWARTEY

Sunday 20th October 2019

**B+ Tree Design - Nodes**

The B+ Tree struct makes use of pointers to store data pointers at the leaf nodes of the tree. The structure of the leaf nodes of the B+ Tree is quite different from the structure of the internal nodes of the B+ Tree. Since data pointers are present only at leaf nodes, the leaf nodes ideally are storing all the key values along with their corresponding data pointers to the disk file block, in order to access them. The leaf nodes are linked to provide ordered access to the records, therefore form the first level of indexing, with the internal nodes forming the other levels of the multilevel index. Some of the key values of the leaf nodes appear in the internal nodes, to simply as a medium to control the searching of records.

In the program we set the MAX\_KEYS to 1024, which is to allow for the nodes to have a capacity of 1024 keys. It is important to have a reasonably large fanout for the B+ Tree. This will allow for the height of the tree to be relatively small. This will make the searching for a key value shorter because there are lesser nodes to touch. For the leaf nodes, they are stored in the form <K, P>, where K is the key value, and P is the data pointer. The data pointer points to the actual data on the disk file block. Each leaf node has a sorted array of keys and are at the same level.

The MAX\_KEYS component for my program is set to 1024, which means that each node has the capacity to store up – to 1024 keys. Therefore, the tree may only need rebalancing if the node is 50% full, completely full, of the new value being inserted causes for some form of rebalancing. However, in this program rebalancing occurs when the leaf node reaches its maximum capacity of 1024 keys.

**FIND ALGORITHM**

This find algorithm quickly find all records which meet some condition on the search key attribute. The idea of this algorithm is to start from the root. Then examine index entries in non – leaf nodes to find the correct child / leaf node. The algorithm is written to traverse the tree until it reaches the leaf node. The non – leaf nodes are traversed using a binary / linear search algorithm.

**INSERTION ALGORITHM**

The insert algorithm functions as the insert method for the B+ Tree. It is intuitive to have some addition functionality that just inserting keys in to the B+ Tree. The B+ Tree has many insertion qualities. The most effective algorithm is the insertion that balances the tree (if there is the need to) parallel to insertion of a new key. It is relatively fast for insertion, because the internal nodes store 1 key and act as pointers to navigate the tree. Given that the fanout is larger (1024), this allows for values to be inserted faster. The insert algorithm primarily, find the correct leaf in which the key being inserted belongs by traversing the entire tree with a binary search. It then checks is the node is full or has space for insertion. If there is space in the leaf node, the key value is inserted without an issue. The insert algorithm has two components, which is the insert leaf node and the insert internal node.

**EXPERIMENTAL SETUP**

To empirically measure the efficiency of the FIND and the INSERT algorithm, I had to write a new C script (test.C). This script test how quickly inserting key values into the B+ tree would be, as well as finding these keys from the B+ tree.

To be able to have a fair comparison of both algorithms, it is important to use the same conditions for each algorithm. For the insert algorithm, insert all the keys in bulk, and for the find algorithm, find a key within the inserted numbers.

The table below show the results of times for the FIND and the INSERT algorithm at different level counts of integers. For testing purposes I used an even count of integers, which mean 2 million integers, 4 million integers and so on, up to 10 million integers. There is also a plot to show the trend of the times for the different set of values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INTEGERS** | **INSERT** | | **FIND** | |
| **Expected** | **Actual** | **Expected** | **Actual** |
| **2 million** |  |  |  |  |
| **4 million** |  |  |  |  |
| **6 million** |  |  |  |  |
| **8 million** |  |  |  |  |
| **10 million** |  |  |  |  |

1. Do your observation match my conceptual understanding of the asymptotic efficiency of both operations (big O analysis)

At the end of the execution of the program, my expectations were not exactly matching the conceptual understanding I had wished for from this exercise.