**B+ tree Project Report**

Jiashu Chen, Helen Wu, Hongyi Zhang, Ziyi Zhang

In this project, the design goal is to build a part catalog. Instead of using a database, a B+ tree is implemented to store the part data in it. The part data contains information of part ID and part description. They are stored in a B+ tree that allows 2 to 4 keys per index node and 16 records per leaf node.

The B+ tree is implemented with the following functions, get, insert, update, and remove. With these functions implemented, a B+ tree storage for part catalog is built.

**B+ tree**

The B+ tree data structure is briefly discussed here. A B+ tree is essentially a B tree. The records are only stored at the leaf nodes which are also external nodes. The leaf nodes are required to be at least half full and linked together to form a linked list. The keys are stored at the internal nodes.

Diagram

Description automatically generated

Figure 1. A B+ tree example.

Figure 1 shows a B+ tree of order 3. The root node contains the key 90. It points to two internal nodes contain keys 60 and 78 and keys 108 and 120, respectively. These keys points to the leaf nodes where the records are stored. The records are linked together and form a singly linked list. The singly linked list data structure makes search queries more efficient. As can be seen in the figure, leaf nodes are at least half full. The root and the internal nodes contain the keys.

Diagram

Description automatically generated

Figure 2. The advantages of using a B+ tree.

Figure 2 shows the advantages of using a B+ tree as a storage system. The goal of faster data access is achieved in this way.

**Functions**

In this section, brief descriptions for the functions of the B+ tree are discussed. The operations of getting a record, insertion, and removing a record are discussed with their pseudocodes.

**Get**

Finding a record in a B+ tree is essentially a binary search operation. The algorithm is briefly described here.

A binary search is performed.

If the parameter matches the exact key

the record is returned

Elif the parameter is a leaf node, and the record is not found

report an unsuccessful search

Else

follow the proper branch and repeat the search

**Insert**

B+ tree insertion contains three different cases. Different operations are included to handle these cases. The steps for each case are shown below.

Case 1: no overflow

Insert the record into the leaf node in ascending order.

Case 2: overflow in leaf node

Split the leaf node into two leaf nodes. The first leaf node contains 8 records while the second leaf node contains the remaining records.

Copy the smallest search key value from the second node to the parent node.

Case 3: overflow in internal node

Split the internal node into two nodes. The first node contains 2 records.

Copy the smallest among the remaining to the parent node.

The remaining keys are stored in the second node.

**Remove**

Similar to insertion, removing a record from a B+ tree also requires the proper handle of the leaf node. The leaf node should be half full after the removal.

1. Start at the root and go up to leaf node containing the key K
2. Find the node n on the path from the root to the leaf node containing K
3. If n is root, remove K

* If root has more than two keys, done
* If root has only K
* If any of its child nodes can lend a node

borrow key from the child and adjust child links

* else

merge the child nodes

1. If n is an internal node, remove K

* If n has at least ceil(m/2) keys, done
* If n has less than ceil(m/2) keys
* If a sibling can lend a key,

borrow key from the sibling and adjust keys in n and the parent node

adjust child links

* Else

merge n with its sibling

adjust child links

1. If n is a leaf node, remove K

* If n has at least ceil(M/2) elements, done

In case the smallest key is deleted, push up the next key

* If n has less than ceil(m/2) elements
* If the sibling can lend a key

borrow key from a sibling and adjust keys in n and its parent node

* Else

merge n and its sibling

adjust keys in the parent node

**Conclusion**

In this project, we built a B+ tree. The very first obstacle is to fully understand the concept of B+ tree. After tackle this down, our teammates work on implementing a B+ tree with the knowledge of the nature of B+ tree. This project helps us understand the concept of B+ tree as well as B tree in general better, which I believe it helps build a solid foundation for us to work as software engineers.

After the class presentation, we are reminded that the process we developed the code is incorrect. The proper way to develop a code should be writing a method and immediately using a main function to test it. This is a great lesson for all of us.

In this project, I (Helen Wu) mainly worked on the pseudocode, testing the code my teammates developed, writing the project report, and making the presentation. Jiashu Chen, Hongyi Zhang, and Ziyi Zhang were working together to develop the code which is not able to be complied yet. It is worth noting that Ziyi Zhang and I (Helen Wu) tried our best to work on that code together after to class aiming to make the code work.