B+ tree

implementation

B. Ram Kartikeya

3340-2126

boyini.r@gmail.com

“Java implementation of a B+ Tree, an implementation/representation of a data structure that efficiently retrieves data in a block-oriented storage context”

Contents of the Directory:

- Bplustree.java
- Container.java
- InternalNode.java
- LeafNode.java
- input.txt (sample)
- output_file.txt (sample)

Make File Sample (Doesn't contain Run Check README) :

Courtesy: [Java and Makefiles (swarthmore.edu)]

```
JFLAGS = -g
JC = javac
.SUFFIXES: .java .class
.java.class:
        $(JC) $(JFLAGS) $*.java

CLASSES = \
        Bplustree.java \
        Container.java \
        InternalNode.java \
        LeafNode.java

default: classes

classes: $(CLASSES:.java=.class)

clean:
        $(RM) *.class
```

B^+-Trees

Courtesy: Sartaj Sahni BPlusTree

- Same structure as B-trees.
- Dictionary pairs are in leaves only. Leaves form a doubly-linked list.
- Remaining nodes have following structure:

$$j \ a_0 \ k_1 \ a_1 \ k_2 \ a_2 \ \ldots \ k_j \ a_j$$

j = number of keys in node.

a_i is a pointer to a subtree.

k_i <= smallest key in subtree a_i and > largest in a_{i-1}.

Some thoughts on B+ trees:

Because B+ trees don't have data associated with interior nodes, more keys can fit on a page of memory. Therefore, it will require fewer cache misses in order to access data that is on a leaf node.

B+Trees are much easier and higher performing to do a full scan, as in look at every piece of data that the tree indexes, since the terminal nodes form a linked list. To do a full scan with a B-Tree you need to do a full tree traversal to find all the data.

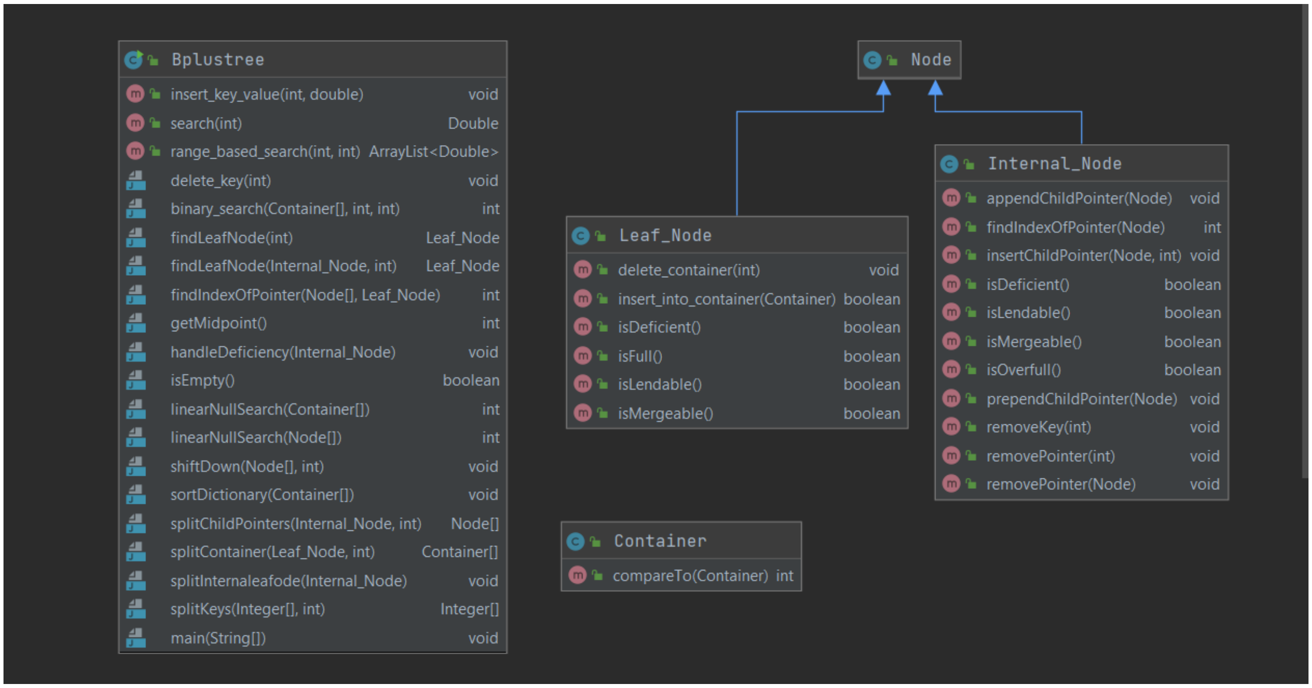
Usages of Bplus Tree:

Courtesy: B+ tree - Wikipedia

"The ReiserFS, NSS, XFS, JFS, ReFS, and BFS filesystems all use this type of tree for metadata indexing; BFS also uses B+ trees for storing directories. NTFS uses B+ trees for directory and security-related metadata indexing. EXT4 uses extent trees (a modified B+ tree data structure) for file extent indexing.[3] APFS uses B+ trees to store mappings from filesystem object IDs to their locations on disk, and to store filesystem records (including directories), though these trees' leaf nodes lack sibling pointers.[4] Relational database management systems such as IBM DB2,[5] Informix,[5] Microsoft SQL Server,[5] Oracle 8,[5] Sybase ASE,[5] and SQLite[6] support this type of tree for table indices. Key-value database management systems such as CouchDB[7] and Tokyo Cabinet[8] support this type of tree for data access. "

The structure of the project:

The project tree below shows 5 different Classes.



Container: It's a Dictionary Pair to hold key-values.

Bplustree: This contains the main driver of the whole project. It

- Inserts (using help of Leaf_Node.class and Internal_Node.class)respectively
- Deletes
- Searches within range
- Searches for a particular key

- Handles deficienys by Merging and combing by using other helper methods like splitChildPointers, splitContainer etc.

Leaf_Node: Extends Node, and has Internal Node specific functionalities like

- checking for deficieny
- checking if a node is ledable(if it can be helpful to a borrower).
- Checking if a node is mergeable
- Or if a node is overfull

With a basic a single container specific option like insert and delete(set the value to null) and value retrieval as key-value pairs are only present in the Leaf Nodes.

Iternal_Node: Extends or has Node as the parent class, has same functionality of a leaf Node with additional functionality like child pointer based operations of finding index of node pointer, removing pointer and the prepending or appending child pointers

Insert and Delete follows the cases when we have an overfull node, or a deficient node to borrow, split and or combine to maintain degree also including the base simple insert case where we just insert into a container.

Insert :

Courtesy: [Sartaj Sahni BPlusTree]

Delete :

Courtesy: [Sartaj Sahni_BPlusTree]

Search has a time complexity of O(h) but we can prove for large m values height is very less. We'll also have to choose an optimum m value to make the B plus tree efficient.

Remaining method short descriptions given in the code as comments

----