COMP4035

Implementing B+-Tree

Project Assignment Report

Group 5

WANG Kai 18250297

SONG Yijin 19251599

XU Yizhou 18250807

Content

1. **Description**
   1.1. B+-tree Description
   1.2. Project Description
2. **Data Structures**
3. **Algorithms**
4. **Usage(platform, usage, procedure)**

1. Description

1.1. B+-tree Description

1.1.1. Example of B+-Tree

Data Set : 2, 7, 10, 13, 13, 14, 16, 16

1.1.2. Components of B+-Tree

- Root
  - Which can contain number in range of [1,m-1]
  - One pointer: Link point to index node(when exist)
- Index node
  - Which should contain number in range of [m/2-1,m-1]
  - Two pointer: Being pointed by the root and point to leaf node
- Leaf node
  - Which should contain number in range of [m/2-1,m-1]
  - Storage the data record
  - Being pointed by index/root, There is a circle link list structure between all leaf nodes.

1.1.3. Basic Rules of B+-Tree

- Non-leaf nodes store only key value information.
- There is a chain pointer between all leaf nodes.
- Data records are stored in leaf nodes.
- Clustered index and secondary index

1.1.4. Search rules of B+-Tree

1. Use the search value compared with the index value until the following situation appears:
   1.the search index smaller than the smallest index value;
   2. the search index bigger than the previous index value but smaller than the next index value;
   3.the search a index bigger than the largest index value
2. After entering the leaf node, the search value compares with the value stored in leaf node and.

1.1.5. Delete rules of B+- Tree

1. Find the position of the value
2. Delete the value
3. check the space of the whole leaf node
4. If the space is not lower than 50% the delete process is completed. If the space is lower than 50% then merge/ redistribute the leaf node.

1.1.6. Inserts rules of B+-Tree

1. Check the status of the tree, if the tree is empty build the value as a root node then the insert end, if the tree is not empty then run the following step.b
2. Use search to find the corresponding value or the value most close to the insert value (range)
3. Check the space of the corresponding node to do the directly insert
4. If there is not enough space, split the left node into two left nodes.

1.2. Project Description

The project is an implementation of B+-Tree, and consists mainly of the UI and BTree two parts.

1.2.1. The **UI part** mainly used to provide response to the user.

btree data.txt:

- scanner in the file
- build in the initial B+-tree: based on the insert function

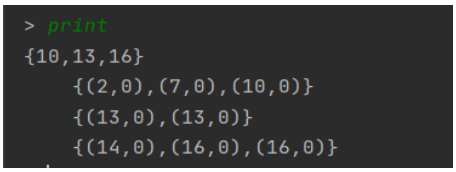
insert 10 20 2

- Randomly choose 2 num in range of [10,20]
- Based on the search function to find the existing value/insert pos

delete 15 16

- Based on the delete function delete all the key exist in the given range
- The delete rule is same as B+-tree delete showing above

print

- Print out the B+-Tree and the picture show in below is the showing format
- 

stats

1.2.2. **BTree part** contains all the class and function and the detail will show in below.

1. **BTree:**
   - Compose: order(the max key count), root, first child
   - Data Structure: B+-Tree (Data structure of BTree)
   - Function:
     - BTree (String Path): Constructor, initial the tree and set the order.
     - insert(int key, int value)/.../insert(int low, int high, int num): through the Stack structure to help to

insert.(Data Structure of BTree insert) and through the recursion to complete. (Recursion of insert)

- delete (long key)/delete (int low, int high): (Algorithms of Delete), and after the delete, entering the check.
- check(Stack<BPlusTreeNode> nodeStack, Stack<Integer> indexStack): check after the delete through the stack to help to check.whether the tree needs to merge or not.(Data Structure of BTree check)
- merge: merge the unqualified space node together t
- search: though the loop to traverse all the number in the range
- print: depends on the toString to complete.
- getNodeTotalCount, getKeyTotalCount, getNodeCount, getAveFillFactor, getHeight

2. Node:

- Compose: index, order, number of keys, the corresponding value of the key, children node, sibling node.
- Data structure: array (Data structure of Node)
- Function:
  - BPlusTreeNode(int m): Constructor, construct the node and set the order.
  - show (int depth): Print the information tree under this node.(Recursion of the show)
  - toString(), toString(int level): return the information tree under this node in string type.(Recursion of the toString)

2. Data Structures

2.1. Array

2.1.1. **Data structure of Node:** key, value, children node

Through the fixed length array to store these variables at the same time control the length(volume) of these variables.

2.2. Stack

2.2.1. **Data Structure of BTree insert:**

Through the stack we can easily pop or push the node and find the first node.

2.2.2. Data Structure of BTree check:

Through the stack we can easily pop or push the node and check whether the space is enough.

2.3. Tree

2.3.1. **Data Structure of BTree:** BTree

The related rules and details is show in the description of the B+-tree, which is the first part of the report

3. Algorithms

3.1. Recursion

3.1.1. **Recursion of the show**

Through the Recursive call in the tail to call the continue show function applied in the children node until all the nodes are shown.

3.1.2. **Recursion of the toString**

Through the Recursive call in the tail to call the continue toString function to transfer all the children nodes into string format.

3.1.3. **Recursion of the Insert**

For the insertion part, we use two separate functions to implement it. The first function of insertion was indicated to find the corresponding locations of one key, and the second one would insert it and deal with different situations, like how to copy it up and push it up when the nodes are full.

3.2. Others

3.2.1. **Algorithms of Delete**

We only use one function to deal with the deletion. And to simplify, we directly delete one key from the tree. The check function is used to make detection after this operation.

3.2.2. **Algorithms of Check**

The check function is to accomplish the work after deletion. If a node is checked as less than half-full, then the merge and check function would be applied to it. One node would be merged to its sibling, parent node based on the different situation.

4. Usage(platform, usage, procedure)

This application is fully built in the language of Java, and it can be applied into any computer environment with JDK or other version of java pre-built.