CS127 PA3

The due date for this assignment is **December 2nd, 2015 at 3:30 PM**. Submissions that miss this deadline by less than a day will receive a 50% penalty. Solutions sent more than a day late need not be submitted.

Introduction

PA1 and PA2 were designed to teach you about the skills involved in being a database administrator. PA3 will focus on the systems side of databases. You will be asked to implement – in Java – a B+-Tree that conforms to specific guidelines.

Background

A B+-Tree is a balanced tree where every path from the root of the tree to the leaf of the tree is the same length. For a B+-Tree of degree `n`, the following constraints must hold:

- At the root node, the maximum number of pointers in `n` and the minimum number of pointers is `2`.
- At an internal node (not the root and not a leaf), the maximum number of pointers is `n` and the minimum number of pointers is `n/2`, rounded up.
- At leaf nodes, the maximum number of values stored is `n-1`, and the minimum number of values stored is `(n - 1)/2`, rounded down.

The specifics of B+-Trees have been covered in class.

Java Classes

This project requires you to work with several Java classes.

- `Main`: this class provides two important methods: `main`, which you can use to test your B+-Tree at the command line, and `buildTree`, which will be called by the tests. You should not need to change this class.
- `BTree`: this class contains the outwards-facing functionality of the B+-Tree (`insert` and `delete`). It also contains a method called `outputGraphviz`, which you can use for rendering your tree graphically, if you so desire. You should not need to change this class.
- `Node`: this is the superclass of the two types of nodes in the B+-Tree (`InternalNode` and `LeafNode`). Many of the methods of this class are implemented for you and documented. Some of them will likely be very

useful. Unlike in class, our nodes have pointers to both their next sibling (`next`) the their previous sibling (`prev`). You will need to implement the marked methods in this class.

- `InternalNode`: inherits from `Node` and represents the internal nodes of the tree. You will need to implement these methods. You will need to implement the marked methods in this class.
- `LeafNode`: inherits from `Node` and represents the nodes at the bottom of the tree.
- `Reference`: contains information about references to nodes or to values

Task Descriptions

You will need to implement a working version of the B+-Tree algorithm. We will use the static method `buildTree` in `Main.java` to test your code. As you can see, this method will call the `BTree` constructor with a single parameter equal to the desired degree of the tree. Then, this method will call these methods in the `BTree` class:

- `BTree::insert`, which should insert a value into the `BTree`
- `BTree::delete`, which should delete a value in the `BTree`
- `BTree::print`, which should print out the BTree (for your convenience only)
- `BTree::search`, which should check to see if a value is in the tree (not tested by us)

We will test your code in the following way:

1. `buildTree` in `Main.java` will be invoked to get a B+-Tree with values inside of it.
2. We will extract the root node from the returned tree `t`, via `t.root`.
3. We will use the `Node::getPtr` and `Node::getKey` methods to ensure that the returned tree has the correct structure.

We highly recommend taking a look at the provided unit tests to see what we are expecting. *If you'd like*, you may ignore the rest of the provided code and simply make sure the `BTree` class works as specified. You may not modify `Main.java` or any of the tests. *However*, the B+-Tree algorithm is quite complex, and we've written a substantial chunk of the algorithm for you. You'll just have to fill out a few of the more tricky methods…

Methods that you must implement are marked inside of the Java files, and they are described here as well.

- `InternalNode::search(int val)` returns `Reference`: This method will call `findPtrIndex(val)` (specified in class Node), which returns the array index `i`, such that the `i`th pointer in the node points to the subtree containing `val`. It will then call `search()` recursively on the node returned in the previous step.

- `InternalNode::insert(int val, Node p)` returns `void`: This method will call `findKeyIndex(val)` (specified in the class Node), which returns the array index `i` in the key array such that `keys[i]` is the largest key that is less than or equal to `val`. It returns a `Reference` referring to the current node and an array index in the `key` array. If the `i`th key in the `key` array matches `val`, `match` is set to true in the `Reference` returned, otherwise it is set to false.
- `InternalNode::insert(int val, Node p)` returns `void`: This method calls the following routines, which you will implement:

  1. `findKeyIndex(val)`: (in the class `Node`)
  2. `insertSimple()`: (this method is called when the current node is not full)
  3. `redistribute()`: This method is called when the node is full. In class, algorithms presented involved creating a supernode and splitting it into two. In this implementation, you will instead create a new `InternalNode`, and distribute the keys and pointers in the full node (as well as the value and pointer to be inserted) so that the old node and the new node are each half full. `redistribute()` is the method you will call to trigger this movement of data between nodes. This method should return the key value to be inserted into the parent node as a result of the split.

- `LeafNode::insert(int val, Node p)` returns `int`: This method calls the following routines:

  1. `findKeyIndex()`: (specified in the `Node` class)
  2. `insertSimple()`: (called when the current node is not full)
  3. `redistribute()`: This method is as defined in `InternalNode` except that it creates a new `LeafNode` and the key returned is the first key of the new node.

- `Node::delete(int i)` returns `void`: This method calls the following routine which you will need to define as:

  1. `deleteSimple()`: this method removes the `i`th key and pointer from the key array and pointer array of the node. The method is implemented in
  2. `combinable(Node other)`: This method is used to test if this node can be combined with other node without splitting.
  3. `combine()`: This method combines the current underfull node with the sibling to it's right. This method should be called only if this node is `combinable()` with it's right sibling.
  4. `redistribute()`: If current node cannot be combined with one of its siblings, use this method to distribute its keys with it's next sibling.

- `Node::redistribute()` returns `int`: This method moves some of the keys and pointers from this node to its next sibling. There are no parameters here because each node knows its next sibling. If `p` is the current node:

  1. To redistribute keys and pointers of `p` with its next sibling, use `p.redistribute()`.
  2. To redistribute keys and pointers with `p`'s previous sibling, use `(p.prev).redistribute()`.In this case, `p`'s previous sibling moves some of its keys and pointers to `p`.
  3. To split a full node, `p`, into two nodes:
  4. create a new node, `p'` (`p'` is between `p` and `p.next`).
  5. call `p.redistribute()` to move keys and pointers from `p` to `p’`.

- `Node::combine()` returns `void`: given a node, `p`, `p.combine()` moves all keys and pointers of `p`'s next node to `p`. If `p` is the current node to combine with `p`'s next node, use `p.combine()`. If p is the current node to combine with `p`'s previous node, use `(p.prev).combine()`. This call moves all keys and pointers from `p` to the previous sibling of `p`.

As a reminder, the following are properties of a B+-Tree:

- If a node is underfull and can be merged with either of it's siblings, merge it with the sibling to it's right (next sibling).

- When a node is split into two nodes, the first node should have at least as many keys and pointers as the second node.

- Every key in an internal node must also appear in a leaf node. This means that each time you delete a key from a leaf node, you should update the internal nodes if necessary.

- When a node becomes underfull after a deletion, if it can be combined with one sibling and redistributed with another sibling, choose combining with its sibling. In other words, in your `delete()` method, the process that deals with an underfull node should be:

  if (siblings(next) && combinable(next)) { // combine with next sibling } else if (siblings(prev) && combinable(prev)) // combine with previous sibling } else if (siblings(next)) { // redistribute with next sibling } else if (siblings(prev)) { //redistribute with previous sibling }

Getting Started

Eclipse Project

First, get the files from LATTE. You should have a file called `CS127_PA3_Eclipse.zip`, which can be imported as an Eclipse project.

1. Download the ZIP file.
2. Open Eclipse
3. Choose `File -> Import`
4. Select `General -> Existing Projects into Workspace`
5. Pick the “Select archive file” radio button
6. Click `Browse` and locate the ZIP file
7. Press `Finish`

Debugging

The folder `data` contains 5 example command sets that can be feed to your program via standard in. Notice that each contains the `o` and `p` commands, which will cause the tree to be printed out to the terminal and for a GraphViz file called `tree.dot` to be created. You can turn `tree.dot` into a graphical representation of your tree like this:

```
dot -Tpng tree.dot > tree.png
```

You can then compare the tree generated by your code to the correct answers, which are stored in the `results` folder.

Testing

The JUnit tests ran by `AllProvidedTests` will use the `buildTree` method of the `Main` class in order to construct a tree. This tree will be tested against the correct values. A good submission should pass every test.

Submitting

If you would like to leave us any notes, such as reasons a test is failing, for *possible* partial credit, please leave them as comments in `Main.java`.

To submit, simply export your project from Eclipse by right-clicking on your project folder, selecting `Export`, and then choosing the `General -> Archive File` option. Please name your submission with your first initial and your last name followed by `_CS127PA3.zip`. For example, `RMarcus_CS127PA3.zip`. Use LATTE to submit your ZIP archive.

Do not simply send your `src` folder. Do not extract each of your source files individually. Do not use `.rar` or `.7z`. Do not package the Eclipse exported archive inside of another archive. Using the export feature in Eclipse (as described) will ensure that we will be able to grade your submission.