Branch: master Find file Copy path

fa19-moocbase / hw2-README.md

es1024 Release (hw2) - 09-16-2019
8df4315 6 hours ago

1 contributor

Raw Blame History

187 lines (139 sloc) 7.5 KB

Homework 2: B+ Trees

This homework is due: **Friday, 9/27/2019, 11:59 PM**.

Overview

In this homework, you will be implementing B+ tree indices.

Prerequisites

You should watch the B+ Trees lectures before working on this homework.

Getting Started

The test cases for this homework are all located in `src/test/java/edu/berkeley/cs186/database/index`. To build and test your code in the container, run the following inside `/cs186/fa19-moocbase`:

```
mvn clean test -D HW=2
```

There should be 16 failures, 4 errors, and 20 tests run.

Understanding the Skeleton Code

The `index` directory contains a partial implementation of an Alternative 2 B+ tree (`BPlusTree`), an implementation that you will complete in this assignment. Every B+ tree maps keys of type `DataBox` (a single value or "cell" in a table) to values of type `RecordId`. A B+ tree is composed of inner nodes (`InnerNode`) and leaf nodes (`LeafNode`). Every B+ tree is persisted to disk (through the Buffer Manager and Disk Space Manager), and every inner node and leaf node is stored on its own page. Calls are done through `BPlusTree`, which propagate through zero or more `InnerNode`s, and eventually terminate at a `LeafNode`.

Both `LeafNode` and `InnerNode` inherit from `BPlusNode`, which has some useful methods implemented already.

You should read through all of the code in the `index` directory. Many comments contain critical information on how you must implement certain functions. For example, `BPlusNode::put` specifies how to redistribute entries after a split. You are responsible for reading these comments. If you do not obey the comments, you will lose points. Here are a few of the most notable points:

- Our implementation of B+ trees does not support duplicate keys. You will throw an exception whenever a duplicate key is inserted.
- Our implementation of B+ trees assumes that inner nodes and leaf nodes can be serialized on a single page. You do not have to support nodes that span multiple pages.
- Our implementation of delete does not rebalance the tree. Thus, the invariant that all non-root leaf nodes in a B+ tree of order `d` contain between `d` and `2d` entries is broken. Note that actual B+ trees **do rebalance** after deletion, but we will **not** be implementing rebalancing trees in this project for the sake of simplicity.

LockContext objects

There are a couple of points in this project, where a method will take in objects of the type `LockContext`. You do not need to worry too much about these objects right now; they will become more relevant in HW4.

If there are any methods you wish to call that require these objects, use the ones passed in to the method you are implementing, or defined in the class of the method you are implementing (`this.lockContext` for `BPlusTree` and `this.treeContext` for `InnerNode` and `LeafNode`).

B+ Tree

You should first implement the `fromBytes` in `LeafNode`. This method reads a `LeafNode` from a page. For information on how a leaf node is serialized, see `LeafNode::toBytes`. For an example on how to read a node from disk, see `InnerNode::fromBytes`. Note that `testToAndFromBytes` will not pass just from completing this method, you need to complete `LeafNode#put`.

After implementing `fromBytes`, you will need to implement the following methods in `LeafNode`, `InnerNode`, and `BPlusTree`:

- `get`
- `getLeftmostLeaf` (`LeafNode` and `InnerNode` only)
- `put`
- `remove`
- `bulkLoad`

For information on what these methods should do, refer to the comments in `BPlusTree` and `BPlusNode`. Do not forget to call `sync` when implementing the three mutating methods (`put`, `remove`, and `bulkLoad`); it's easy to forget.

Each of these methods, although split into three different classes, can be viewed as one recursive action each - the `BPlusTree` method starts the call, the `InnerNode` method is the recursive case, and the `LeafNode` method is the base case. It's suggested that you work on one method at a time (over all three classes).

You will also need to implement the following methods in `BPlusTree`:

- `scanAll`
- `scanGreaterEqual`

In order to implement these, you will have to complete the `BPlusTreeIterator` inner class in `BPlusTree.java`.

After this, you should pass all the HW2 tests we have provided to you (and any you add yourselves). These are all the provided tests in `database.index.*`.

Debugging

To help you debug, we have implemented the `toDotPDFFile` method of `BPlusTree`. You can add a call to this method in a test to generate a PDF file of your B+ tree.

For example,

```
BPlusTree tree = ...
tree.toDotPDFFile("tree.pdf");
```

If you get `"Cannot run program "dot"`, and are running your code inside the container, run the following (inside your container) and then try again:

```
sudo apt-get update
sudo apt-get install -y graphviz
```

If you are running this outside your container (inside your IDE, for example), you need to install GraphViz. Alternatively, you can run the following inside your container after the exception is thrown to manually generate a PDF file from the `tree.dot` file:

```
dot -T pdf tree.dot -o out.pdf
```

Submitting the Assignment

See the main readme for submission instructions. The homework number for this homework is hw2.

You may **not** modify the signature of any methods or classes that we provide to you, but you're free to add helper methods.

You should make sure that all code you modify belongs to files with HW2 todo comments in them (e.g. don't add helper methods to DataBox). A full list of files that you may modify follows:

- index/BPlusTree.java
- index/InnerNode.java
- index/LeafNode.java

Make sure that your code does *not* use any static (non-final) variables - this may cause odd behavior when running with maven vs. in your IDE (tests run through the IDE often run with a new instance of Java for each test, so the static variables get reset, but multiple tests per Java instance may be run when using maven, where static variables *do not* get reset).

Testing

We strongly encourage testing your code yourself. The given tests for this project are not comprehensive tests: it is possible to write incorrect code that passes them all (but not get full score).

Things that you might consider testing for include: anything that we specify in the comments or in this document that a method should do that you don't see a test already testing for, and any edge cases that you can think of. Think of what valid inputs might break your code and cause it not to perform as intended, and add a test to make sure things are working.

To help you get started, here is one case that is *not* in the given tests (and will be included in the hidden tests): everything should work properly even if we delete everything from the BPlusTree (iterator should immediately return false for `hasNext`).

To add a unit test, open up the appropriate test file (in `src/test/java/edu/berkeley/cs186/database/index`) and simply add a new method to the file with a `@Test` annotation, for example:

```
@Test
public void testEverythingDeleted() {
    // your test code here
}
```

Many test classes have some setup code done for you already: take a look at other tests in the file for an idea of how to write the test code.

Grading

- 60% of your grade will be made up of tests released to you (the tests that we provided in `database.index.*`).
- 40% of your grade will be made up of hidden, unreleased tests that we will run on your submission after the deadline.