CS 4320/5320 Homework 2

Fall 2014

Due October 8, 2014

This assignment is due on Wednesday, 10/8/2014 at 11:59pm. It is out of 100 points and counts for 10% of your overall grade.
You must work on this homework in **groups of 2 or 3 students**. You should make one CMS submission for your entire group, and everyone in the group will receive the same grade.
There is both a coding part (in Java) and a written part to this assignment.
Hint: You might feel more confident with the coding part after you have solved question 2 of the written part.

1 Written Part (50 points)

1.1 Question 1: Indexing Basics (9 points)

Consider a relation with this schema:

$$Movies(\underline{name : string}, budget : integer, length : integer, genre : string)$$

For each of the following indexes, list whether the index matches the given selection conditions. (each 1 points; 8 points in total)
Note: Here, a simple yes/no-answer is sufficient.

(a) A B+-tree index on the search key (Movies.budget, Movies.length)

(1) $\sigma_{Movies.budget<5.000.000 \wedge Movies.length=180}(Movies)$
(2) $\sigma_{Movies.budget=6.000.000 \wedge Movies.length>120}(Movies)$
(3) $\sigma_{Movies.length=60}(Movies)$
(4) $\sigma_{Movies.budget=5.000.000}(Movies)$

(b) A hash index on the search key (Movies.budget, Movies.length)

(1) $\sigma_{Movies.budget=8.000.000 \wedge Movies.length>}(Movies)$
(2) $\sigma_{Movies.budget=12.000.000 \wedge Movies.length=80}(Movies)$
(3) $\sigma_{Movies.budget=7.500.000}(Movies)$
(4) $\sigma_{Movies.length=60}(Movies)$

From these observations, explain the advantage of tree-based indexes over hash-based indexes. (1 point)

1.2 Question 2: Tree-based Indexing (11 points)

Assume that you have just built a dense B+ tree index using Alternative (2), on a heap file containing 10,000 records. The key field for this B+ tree index is a 20-byte string, and it is a candidate key. Pointers (i.e., record ids and page ids) are (at most) 8-byte values. The size of one disk page is 560 bytes. The index was built in a bottom-up fashion using the bulk-loading algorithm, and the nodes at each level were filled up as much as possible. For all cases, please show your work.
Note: Alternative (2) means that every entry contains a key and a single reference to a data entry
In all your answers, be sure to show your work and explain your reasoning to get full credit.

(a) How many levels does the resulting tree have? (4 points)

(b) For each level of the tree, how many nodes are at that level? (2 points)

(c) How many levels would the resulting tree have if key compression is used and it reduces the average size of each key in an entry to 10 bytes? (2 points)

(d) How many levels would the resulting tree have without key compression but with all pages 70 percent full? (3 points)

1.3 Question 3: Hash-based Indexing (14 points)

h(1)	**h(0)**	**Primary Pages**				**Overflow Pages**
000	00	32	8	24		
001	01	9	25	41	17	
010	10	14	18	10	30	
011	11	31	35	7	11	
100	00	44	36			

Figure 1: Hash-Index for question 3

Consider the Linear Hashing index shown in Figure 1. The round number counter *Level* is 0 and the split counter *Next* is 1. Assume that we split whenever an overflow page is created. Answer the following questions about this index:

(a) What can you say about the last entry that was inserted into the index? (1 point)

(b) What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far? (2 points)

(c) Suppose you know that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split? (1 point)

(d) Show the index after inserting an entry with hash value 4. Indicate any change in round number or split counter. (2 points)

(e) Show the original index after inserting and entry with hash value 15. Indicate any change in round number or split counter. (2 points)

(f) Show the original index after deleting entries with hash values 36 and 44. Assume that the full deletion algorithm is used. Indicate any change in round number or split counter. (2 points)

(g) Find a list of entries whose insertion into the original index would lead to a bucket with two overflow pages. Use as few entries as possible to accomplish this. What is the maximum number of entries that can be inserted into this bucket before a split occurs that reduces the length of this overflow chain? (4 points)

1.4 Question 4: Indexing Costs (16 points)

You are given the following information:

- Executives has attributes `ename`, `title`, `dname`, and `address`; all are string fields of the same length.
- The `ename` attribute is a candidate key.
- The relation contains 8,000 pages.
- There are 5 buffer pages.

Suppose that the query is as follows:

```
SELECT E.title, COUNT(*) FROM Executive E
WHERE E.dname > 'W%' GROUP BY E.title
```

Assume that only 10% of Executive's tuples meet the selection condition. Please show your work for all questions.
In all your answers, be sure to show your work and explain your reasoning to get full credit.

(a) Suppose that a clustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan? If an additional index (on any search key you want) is available, would it help to produce a better plan? (4 points)

(b) Suppose that an unclustered B+ tree index on title is (the only index) available. What is the cost of the best plan? (3 points)

(c) Suppose that a clustered B+ tree index on `dname` is (the only index) available. What is the cost of the best plan? If an additional index (on any search key you want) is available, would it help to produce a better plan? (4 points)

(d) Suppose that a clustered B+ tree index on `<dname, title>` is (the only index) available. What is the cost of the best plan? (3 points)

(e) Suppose that a clustered B+ tree index on `<title, dname>` is (the only index) available. What is the cost of the best plan? (2 points)

2 Coding Part: ISAM Tree (50 points)

An ISAM tree is a static tree structure in which the **index nodes** direct the search and the **leaf nodes** contain the data entry.

(1) Constraints and Properties of ISAM trees:

  - Keys are sorted in the nodes.
  - Nodes are sorted by their keys

- The structure of the tree does not change after creation; except for overflow pages.

In this assignment, you are asked to write an ISAM tree that has Integer keys and stores String values. It has create, search, insert and delete functionality.

(2) (a) **Create:** Given a set of key-value pairs, a ISAM tree is constructed.
   (b) **Search:** Given a key, return the associated String value (or null if there is none)
   (c) **Insert:** Given a key/value pair, insert it into the correct leaf node. You might need to create an overflow node.
   (d) **Delete:** Given a key, delete the corresponding key/value pair. You have to delete unneeded overflow nodes as soon as possible. This means you might have to merge overflow nodes.

2.1 Skeleton Code

We provide you with the following skeleton code:

(1) **IsamTree.java** : main class to implement, see the specification as described in the comments in the skeleton code.

(2) **IsamNode.java**: general page/node class for the ISAM tree.

(3) **IsamIndexNode.java**: subclass of IsamNode, represents the index (i.e. interior) nodes of the ISAM-tree.

(4) **IsamDataNode.java**: subclass of IsamNode, represents the leaf nodes of the ISAM tree.

(5) **IsamTreeTest.java**: JUnit test class containing one test example. Although not required for grading purposes, you should write more test cases to test your tree. *Note: the textBook-testcase can be seen in Figure 10.5 on page 343 of the textbook.*

2.2 Hints

- You can assume that `create()` will only be called once per instance/execution.
- `create()`, `insert()`, `get()`, and `delete()` are the main functions that will be called when we test your tree.
- You don't need to implement logic that sorts content of leaves.
- The `toString()`-function is already done. Use it to debug your code!
- You are free to add any new classes and methods to the project as long as you don't change the arguments/names of existing functions.
- You are also free to use any classes from the Java SDK.
- You are expected to use the JUnit-tests and write additional ones to test your code. They will not affect grading but help you a lot solving the assignment.

2.3 Grading

We will be running unit tests like the one provided in the skeleton code to test your ISAM tree's create, search, insert and delete. If your code does not run or fails all our test cases, it will receive no more than 5 points. The points breakdown is approximately as follows:

(1) 20 points on Create

(2) 5 points on Search

(3) 10 points on Insert

(4) 10 points on Delete

(5) 5 points on logic and style (will be used mostly to allow us to penalize excessively inefficient and/or unreadable code)

3 Submission Instructions

Submit the two following files:

- a .pdf file containing all your answers to the written questions. As before, all your answers must be typeset; scans of handwritten answers are not acceptable and will receive zero points.
- a .zip file containing: all your **java** files. Also include a README file that includes the names and netIDs of everyone in your group and explains your coding logic and any known bugs.

4 Late Submissions:

See the standard course late policy as posted in CMS.

5 Academic Integrity:

See the standard course academic integrity policy as posted in CMS.