CS 4320/5320 Homework 2

Spring 2016

Due: 11 March, 2016

This assignment is due on Friday, 2016-03-11 at 23:59. It is out of 95 points and counts for 10% of your overall grade. We assume you are familiar with the course policies so we do not repeat information about group formation, academic integrity, etc.

1 Programming Part: B+ Tree (50 points)

In this section, you are asked to implement a basic BPlusTree structure as described in the textbook (pages 344-356). For simplicity, you can assume for this homework that no duplicate keys will be inserted, and that keys will implement interface *Comparable*.

1.1 Description

A B+ tree is a balanced tree structure in which the **index nodes** (internal nodes) direct the search and the **leaf nodes** contain the data entries.

1. Constraints and properties of B+ trees:
   - Keys are sorted in the nodes.
   - Nodes are sorted by their keys. For Index Nodes, a B+ tree asserts that for all keys k and adjacent pointers, $before(k)$ and $after(k)$, it must be that:

     $$max(keys(before(k))) < k \leq min(keys(after(k))) \tag{1}$$

     In other words, k is the key that divides the keys in pages $before(k)$ and $after(k)$.
   - The tree is balanced: all paths from the root to the leaf nodes must be of equal length.
   - The nodes are sufficiently filled: For any non-root node, given Order D, D $\leq$ node.keys.size() $\leq$ 2*D. The node is *underflowed* if $node.keys.size() < D$, and *overflowed* if $node.keys.size() > 2 * D$.
2. In this assignment, you are asked to write a B+ tree that has comparable keys and stores values (of any class). It has search, insert and delete functionality.
   (a) **Search:** Given a key, return the associated value.
   (b) **Insert:** Given a key/value pair, insert it into the correct leaf node, without violating the B+ tree constraints noted above. In insertion, there are three distinct cases to think about:
      - the target node has available space for one more key.

- the target node is full, but its parent has space for one more key. (leaf overflow)
- the target node and its parent are both full. (leaf overflow and index node overflow)

When splitting a node during insertion, the first D keys stay, while the rest move to a new node.

(c) **Delete:** Given a key, delete the corresponding key/value pair without violating the B+ tree constraints noted above.

There are also three main cases to think about (after deletion). Here is how your tree should handle them:

- the target node is not underflowed (at least half full), there is nothing left to do.
- the target node is underflowed. Identify a *target sibling* (see below). If the sibling has extra entries, then redistribute *evenly* between the node and sibling. If there is an odd number - say, $2n + 1$ - of total entries, redistribute so that n entries remain in the left node, while $n + 1$ entries are stored in the right node.
- the target node is underflowed and the target sibling doesn't have extra entries (redistribution couldn't be done), then merge the node with the target sibling.

The sibling of a node N is a node that is immediately to the left or right of N and has the same parent as N. In this assignment you should handle underflow as follows. Identify a *target sibling* for the underflowed node. By default, this is the left sibling of the underflowed node. If the underflowed node does not have a left sibling, then use its right sibling. Once the target sibling is identified, try to redistribute with it; if this is impossible, merge the node with the sibling.

Additionally, as per your textbook's description of B+ trees, each Leaf Node should keep track of the leaves to its left (lesser key values) and right (greater key values). Note that a leaf node directly to the left (or right) of another need not be its sibling - it may be its cousin, for example.

1.2 Skeleton Code

We provide you with the following skeleton code:

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

(2) **Node.java**: general node class for the B+ tree.

(3) **IndexNode.java**: subclass of Node.java, represents the index (i.e. interior) nodes of the B+tree.

(4) **LeafNode.java**: subclass of Node.java, represents the leaf nodes of the B+tree.

(5) **Utils.java**: contains methods aiding testing and debugging your code

(6) **Tests.java**: JUnit test class containing some test examples. **You should write more test cases** to test your tree, although you will not be graded on the additional test cases you write.

1.3 To Do

Implement all the methods in BPlusTree.java. Add any helper methods you need in any of the Java classes.

(1) `search()`, `insert()` and `delete()` are the main functions that will be called when we test your tree.

(2) `splitLeafNode()`, `splitIndexNode()` are helper functions that may be called by insert. They split the provided leaf/index node, and return the splitkey/newnode pair to be inserted in the node's parent. For example, say node n contains 1/3/4/5/7. After splitting by splitkey 4, n contains 1/3 and a new node m contains 4/5/7. Entry(4, m) is returned.

(3) `handleLeafNodeUnderflow()`, `handleIndexNodeUnderflow()` are helper functions that may be called by `delete()`. While not required, we advise that you might find writing `delete` easier if you structure your code to use such functions. They take in two adjacent nodes from the same parent, left (contains smaller keys) and right (contains larger keys), and their parent node. They return -1 if underflow is handled by redistribution indicating no further deletion needed. Otherwise, they return the original splitkey position in their parent for the two merged nodes, so that the parent node knows which key/value pair to delete.

(4) Helper methods' signatures can be modified if needed. You can add/remove/modify any of the helper methods as you wish. But the signatures of the main methods (`search`, `insert`, `delete`) must **not** be changed.

1.4 Grading

We will be running unit tests like the one provided in the skeleton code to test your B+ tree's `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) 5 points on Search

(2) 15 points on Insert

(3) 25 points on Delete

(4) 5 points on logic and style (will be used mostly to allow us to penalize excessively inefficient and/or unreadable code)

2 Written Part (45 points)

For all questions in this part, please write down your reasoning as well as your answer. **Just giving a final answer won't give you full points even if correct, and is guaranteed to give you zero points if incorrect.**

2.1 Hashing Indices (10 points)

Assume:

- An Extendible Hashing Index's directory occupies a single page.
- 6 data entries fit on a page.
- In our Linear Hashing Index, a split is triggered whenever adding a new entry creates an overflow page.

- The data we're storing are natural numbers, and the family of hash functions h_i we're using are simply the last $i + 2$ binary digits of the hashed integer. Therefore, hash indices start with 4 buckets.
- Hashing Indices should be diagrammed as in your textbook, with $Next$ values, primary and overflow pages, and hash values for pages listed.

2.1.1 Linear Hashing is bigger (5 points)

Give an example of a Linear Hashing Index and an Extendible Hashing Index with the same data entries, such that the Linear Hashing Index has more pages. Please diagram your answer in the style used by the textbook, chapter 11.

2.1.2 Extendible Hashing is bigger (5 points)

Give an example of a Linear Hashing Index and an Extendible Hashing Index with the same data entries, such that the Extendible Hashing Index has more pages. Please diagram your answer in the style used by the textbook, chapter 11.

2.2 The Cost of Joins (12 points)

Assume a system has the following costs (measured in IOs):

- 1.5 per hash-based lookup
- 6 per lookup with B+ index

Assume we want to join relations R and S on a particular attribute A. S has A as primary key, and R has A as a foreign key (so when computing the join, each element of R matches exactly one element of S). Assume a uniform distribution on the foreign keys, i.e. each tuple in S matches the same number of tuples in R.

Both relations can be sorted in 2 passes.

In terms of the number of tuples and pages of R and S, what is the cost of joining the relations using:

(a) Index Nested Loop Join (Clustered B+ Tree Index on $S.A$) (3 points)

(b) Index Nested Loop Join (Unclustered B+ Tree Index on $R.A$) (3 points)

(c) Index Nested Loop Join (Hash Index on S) (3 points)

(d) Sort-Merge Join (3 points)

2.3 Least Expensive Evaluation (23 points)

Consider a relation with this schema:

$$Cats(\underline{cid : integer}, cname : string, age : integer, location : string, toy\_preference : string)$$

Suppose that the following indexes, all using Alternative (2) from your textbook for data entries, exist: a hash index on cid, an unclustered B+ tree index on age, an (unclustered) hash index on location and a clustered B+ tree index on $\langle location, age\rangle$. Each Cat record is 120 bytes long. For simplicity you can assume that each index data entry is 20 bytes long regardless of whether the index is on one or two attributes. The Cats relation contains 56,000 pages. There are 32 tuples per page.

(a) For each of the following conditions, find the most selective path for retrieving all Cat tuples that satisfy that condition (only). Explain why this path is the most selective and state its cost. You may assume the cost of navigating a tree index root-to-leaf is 2 I/Os and that the reduction factor (RF) for each term that matches an index is 0.1, regardless of whether the condition has = or some other operator such as <.

(i) $age > 10$ (2 points)

(ii) $location =$"CA" (2 points)

(iii) $age > 5$ AND toypreference="Shoelace" (3 points)

(iv) $location$="NY" AND $age > 5$ AND $toy\_preference$="Yarn Ball" (3 points)

(b) For each of the above, suppose you want to retrieve the average age of qualifying tuples. For each selection condition, describe the least expensive evaluation method and give its cost.

(c) Suppose that, for the above selection condition (ii), you want to compute the average age of cats at each location (i.e., group by location). For selection condition (ii), describe the least expensive evaluation method and give its cost. (3 points)

3 Submission Instructions

Submit a .zip archive containing:

1. a .pdf file containing all your answers to the written questions.
2. a `btree` directory containing:
   - all your **java** files.
   - A README file that includes your names and and explains your coding logic, any known bugs and anything else you would like the reader to know.
3. if applicable, an acknowledgments file as required under the academic integrity policy.