Assignment 3

COS 212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Deadline: 26 April 2019 at 23:00

Objectives:

- Implement the complete delete algorithm of a B-Tree.
- Implement a B^+-Tree.

General instructions:

- This assignment should be completed individually, **no group effort** is allowed.
- Only the output of your Java program will be evaluated.
- Your code may be inspected to ensure that you've followed the instructions.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so, you will be awarded a mark of 0. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of 0.
- **All submissions will be checked for plagiarism.**
- You will be afforded three opportunities to upload your submissions for automarking.

Plagiarism:

The Department of Computer Science regards plagiarism as a serious offence. Your code will be subject to plagiarism checks and appropriate action will be taken against offending parties. You may also refer to the the Library's website at `www.library.up.ac.za/plagiarism/index.htm` for more information.

After completing this assignment:

Upon successful completion of this assignment you will have implemented a delete algorithm for a B-Tree that follows the pre-emptive split and merge strategy. You will also have implemented your own B^+-Tree for use with database indexes.

Task 1

In practical 5 you were given a B-Tree and asked to implement various methods. The B-Tree was defined to be always of an even order and thus used the top-down pre-splitting insertion strategy. In this task you will implement the still outstanding delete method of this B-Tree.

Download the archive `assignment3CodeTask1.zip`. This provides a functional B-Tree class and a partially implemented B-Tree node class to use. You will have to complete the `BTreeNode` class by implementing the delete method. A functional insert method that uses the pre-splitting insertion strategy has been provided.

You should use your own helper functions to assist in implementing the delete method as per the specification. However, you may not modify any of the given methods or method signatures.

You are also provided with the file `Main.java`, which will test some code functionality. You are encouraged to write your own test code in this file. Test your code thoroughly using this file before submitting it for marking.

Merging

Merging will take place pre-emptively. If a node with the minimum number of keys is visited and keys cannot be redistributed, it is merged with its sibling (next sibling if possible, otherwise previous sibling) before the deletion key is further considered.

Key Redistribution

Before pre-emptive merging is considered, key redistribution is first attemped. First check if the left sibling has enough keys to share. If it doesn't, check if the right sibling can share keys. If neither of the siblings have enough keys to share, then the node merges.

Deletion

To delete from the tree, delete by copying is always applied. Use the direct predecessor (i.e. the largest key smaller than the key that you would like to delete) when applying delete operations on internal nodes.

Example

You can use the B-Tree visualiser (`https://www.cs.usfca.edu/~galles/visualization/BTree.html`) to get a visual demonstration as to how the pre-emptive merge deletion strategy should work. You need to tick the pre-emptive split/merge option and use even order trees.

Task 2

Consider the STUDENT database table in *Table 1*. The first column is a unique internal identifier given by the database to each record in a table. This column is hidden from all general database activity. All other columns in this table represent user data that is visible. This particular table contains student information consisting of student number, name and surname. The student number has been designated a primary key index since it is unique at a university. To speed up searching for students by surname, another index has been created for the surname column.

Database indexes are generally implemented using B^+-Trees. B^+-Trees represent an efficient way to store indexes in memory and provide a fast and constant way to search for records stored on disk. The

	PrimKey		SecKey
RowID	**StudentID**	**Name**	**Surname**
100	16094340	John	Botha
200	16230943	Lerato	Molefe
300	17012340	Michael	Evans
400	17248830	Isabel	Muller

Table 1: A STUDENT database table

two indexes of *Table 1* can be stored in two B^+-Trees. The first tree would use the student numbers as keys, while the second one would use the surnames. In both trees, the row id would be the value that is stored in the leaf nodes.

Your task is to implement a B^+-Tree that can be used to store both the indexes that are defined in the STUDENT table in *Table 1*. Download the archive `assignment3CodeTask2.zip`. This provides a functional B^+-Tree class and partially implemented B^+-Tree node subclasses to use. You will have to complete the `BPTreeNode` class by implementing the insert, search and delete methods. The classes `BPTreeInnerNode` and `BPTreeLeafNode` inherit from the `BPTreeNode` parent class and will contain type specific functionality.

You should use your own helper functions to assist in implementing the insert, search and delete methods as per the specification documented in the code. However, you may not modify any of the given members, methods or method signatures.

You are also provided with the file `Main.java`, which will test some code functionality. It also provides an example of how the indexes of *Table 1* could be defined and used. You are encouraged to write your own test code in this file. Test your code thoroughly using this file before submitting it for marking.

Search

Searching can be performed either sequentially via the linked leaf nodes or by tree traversal. The value associated with the given key should be returned. If the given key is not found, `null` should be returned.

Deletion

Deletion will always take place at the leaf node level. A possible corresponding separator key in the parent internal node should not be removed during deletion. Only a future node merge can lead to the separator key removal.

Underflow

If a node underflows, first check if the left sibling has enough keys to share. If it doesn't, check if the right sibling can share keys. If neither of the siblings have enough keys to share, then the underflowing node merges with its sibling (left sibling if possible, otherwise right sibling).

Merging

Merging of nodes will require the update of the separator keys in the parent internal node. Firstly, old separator keys will need to be removed. Secondly, a new separator key may need to be added for the newly merged node.

Example

You can use the B^+-Tree visualiser (`https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html`) to get a visual demonstration as to how the insertion and deletion strategies should work. Unfortunately, the visualiser does not leave the separator key in an internal node once the key has been deleted from the leaf node. You should not follow this strategy in your implementation, but stick to the strategy in the textbook.

Submission instructions

You need to create your own makefile and submit it along with your Java code. Your code should be compiled with the following command:

```
javac *.java
```

Your makefile should also include a `run` rule which will execute your code if typed on the command line as in `make run`.

For your submission, you need to place all your source files including your makefile in a zip or tar/gzip archive (you need to compress your tar archive) named uXXXXXXXX.zip or uXXXXXXXX.tar.gz where XXXXXXXX is your student/staff number. There should be no folders/subfolders in your archive.

Submit your code for marking under the appropriate links (*Assignment 3: Task X*) before the deadline. Separate upload links are provided for the two tasks. Each link grants 3 uploads. Since every upload is a marking opportunity, do not use it to test your code.