Project Assignment – Implementing B+-Tree

Due: 11:00am April 12, 2017

- Students are required to work in groups of 2 - 3 members. Students are responsible to form groups among themselves.
- Register groups online (http://www.comp.hkbu.edu.hk/~comp4035/proj/reg.html), by **March 12, 2017**. Successful registration will earn each member 5 marks (out of 100). Those who fail to register their groups by the deadline will be grouped *by random* by the instructor.
- Start early and proceed in steps. Read the project description carefully before you start.
- This assignment is worth 10% of your overall grade.

Project Description

In this assignment, you will implement a B+-tree index:

- Assume the whole B+-tree is kept in the main memory.
- Assume the fan-out of each node is 7 (i.e., 6 index/data entries per page).
- Assume Alternative 2 is used for data entries.
- For simplicity, we only handle integer search keys; we don't maintain actual data records (so the *rids* in data entries can be set to *null*).
- Your implementation should also deal with duplicate key values (refer to Page 357 in the (3rd edition) textbook).
- You can use any language (e.g., C, Java, C++, C#, etc.) for the implementation.

Group Members

- You are required to work in groups of 2-3 members.
- A 3-member group will be graded in the same way as a 2-member group. Under special circumstances, students may form groups of a single student, but ***prior approval*** from the instructor is required. 4-member groups are not allowed.
- Each member of the same group will receive the *same* mark, so you need to distribute the work among yourselves *evenly*. Under exceptional cases, if the members of the same group wish to receive different grades, you should attach one separate page in the documentation, justifying the reason and identifying individual contributions (in percentage).

Assignment Requirements

In this assignment, you are required to implement a class, *BTree*, link it with the main test program, and make sure that all test programs run successfully. Note that a successful run does not mean that your program is correct. You should also ensure that your program will work for all possible test cases.

The details of the functions that you have to implement are given below. We have also provided some sample code that illustrates the use of the class *BTree*. However, these samples are extremely simple and do not reflect the actual coding that we expect from you. For example, we do not show any error checking in these samples. You are expected to write robust programs by signaling errors when necessary.

BTree::BTree

The constructor for the BTree takes in a filename, and checks if a file with that name already exists. If the file exists, we "open" the file and build an initial B+-tree based on the key values in the file. Otherwise, we return an error message and the program terminates.

BTree::~BTree

The destructor of BTree just "closes" the index. This includes de-allocating the memory space for the index. Note that it does not delete the file.

BTree::Insert

This method inserts a pair (key, rid) into the B+-Tree Index (rid can always be assumed to be 0 in your implementation). The actual pair (key, rid) is inserted into a leaf node. But this insertion may cause one or more (key, pid) pair to be inserted into index nodes. You should always check to see if the current node has enough space before you insert. If you don't have enough space, you have to split the current node by creating a new node, and copy some of the data over from the current node to the new node.[1] Splitting will cause a new entry to be added in the parent node.

Splitting of the root node should be considered separately, since if we have a new root, we need to update the root pointer to reflect the changes. Splitting of a leaf node should also be considered separately since the leaf nodes are linked as a link list.

Due to the complexity of this function, we recommend that you write separate functions for different cases. For example, it is a good idea to write a function to insert into a leaf node, and a function to insert into an index node.

BTree::Delete

This method deletes an entry (key, rid) from a leaf node. Deletion from a leaf node may cause one or more entries in the index node to be deleted. You should always check if a node underflows (less than 50% full) after deletion. If a node becomes underflows, merging or redistribution will occur (read and implement the algorithm in the notes).

You should consider different scenarios separately (maybe write separate functions for them). You should consider deletion from a leaf node and index node separately. Deletion from the root should also be considered separately.

The following code fragment may be helpful:

[1] Assume the re-distribution option is NOT used.

Checking if a node is half full:

```
if (node->AvailableEntries() < (Fanout-1)/2)
{
    // Try to re-distribute, borrowing from sibling
    //   (adjacent node with same parent as this node).
    // If re-distribution fails, merge this node and sibling.
}
```

BTree::Search

This method implements range queries. Given a search range (*key1*, *key2*), the method returns all the qualifying key values in the range of between *key1* and *key2* in the B+-tree. If such keys are not found, it returns "none". *Be careful with the duplicate keys* that span over multiple pages.

BTree::DumpStatistics

In this method, you are required to collect statistics to reflect the performance of your B+-Tree implementation. This method should print out the following statistics of your B+-Tree.

1. Total number of nodes in the tree
2. Total number of data entries in the tree
3. Total number of index entries in the tree
4. Average fill factor (used space/total space) of the nodes.
5. Height of tree

These statistics should serve you in making sure that your code executes correctly. For example, the fill factor of each node should be greater than 0.5. You should make sure that DumpStatistics performs this operation.

BTree::PrintTree, BTree::PrintNode

These are helper functions that should help you debug, by showing the tree contents. PrintTree must be implemented and PrintNode is optional.

User Interface

You should run the program interactively. The program should take one argument, which specifies the data file storing the search key values on which the initial B+-tree is built. After you launch the program, the program will wait for commands on stdin. An example interface is given below (the ***bold*** lines are your input, while the others are the output of your program):

> btree -help
Usage: btree [fname]
fname: the name of the data file storing the search key values

> btree data.txt
Building an initial B+-Tree...
Launching B+-Tree test program…

Waiting for your commands:

insert 10 20 2
2 data entries with keys randomly chosen between [10, 20] are inserted!

delete 15 16
The data entries for values in [15, 16] are deleted.

print
print out your B+-tree here; the format is up to you as long as it's clear …

stats
Statistics of the B+-tree:
Total number of nodes: 10
Total number of data entries: 30
Total number of index entries: 3
Average fill factor of leaf nodes:
60%
Height of tree: 3

quit
Thanks! Byebye ☺

>

The data file containing the search key values has the format of one value per line. An example data file is:

13
2
16
14
10
13
16
7

Below are the commands your program should support:

insert <low> <high> <num>	Insert *num* records randomly chosen in the range [low, high]
delete <low> <high>	Delete records with key values in the range [low, high]
search <low> <high>	Return the keys that fall in the range [low, high]
print	Print the whole B+ tree (format up to you, but be clear!)
stats	Show stats
quit	Terminate the program

Submission Procedure

1) Zip the entire set of source code, together with a report, and make files if any, and email the zip file to the TA (Mr. Xu Cheng: chengxu@comp.hkbu.edu.hk). Further details will be provided later.

2) The report should describe (expected to be 2-5 pages long):
   - Brief description of this project and the B+-tree
   - The data structures used in the implementation
   - Algorithms used
   - The platform (Windows or Unix), the usage of your program, and the installation procedure if needed
   - Highlight of features, if any, beyond the required specification

Grading Criteria

- **Group Registration (5%)**: You will get 5 marks upon successful registration of your group before the deadline (March 2, 2016).
- **Correctness (65%)**: You will get full marks if your implementation is correct. Partial credit will be given to a partially correct submission.
- **Coding Style (10%)**: We expect you to write neat code. Code should be nicely indented and commented. We also expect you to follow the coding conventions.
- **Documentation (20%)**: The report should be short, clear, concise, and informative (see the guidelines above).

Coding Conventions

You need to follow certain coding conventions for all your assignments.

- Name for all classes/methods/types should start with a capital letter. Break multiple words with caps. For example InsertLeafEntry.
- Name for all members/variables should start with small letters. Break multiple words with caps. For example numOfNodes;
- Name for all enum/macro should be all caps. Break multiple words with underscore. For example NODE_FANOUT.

VERY IMPORTANT Advice

- Start early.
- Do this assignment in increasingly more difficult steps. For example, you might want to implement BTree::Insert and for tree with only one node (a root) and assume no overflow. Then implement BTree::Insert that handles overflows in leaf nodes, and then implement BTree::Insert to handle overflows in index nodes.
- The project must be done by the individual group. *No sharing of code and copying of code from others are allowed. Once detected, all the involved will be given some penalty in the final grade in addition to zero mark to the project.*