Project Assignment – Implementing B+-Tree

Due: 11:59pm 2nd December (Friday), 2022

- Each group should have 1-2 members. Students are responsible to form groups among themselves.
- Each group should have a group leader. The group leader needs to send an email listing all the members (including the information of student name and student id) in the group to the TA's email address (jbhuang@comp.hkbu.edu.hk), by **4th November (Friday), 2022**. If you do not send the email before 4th November 2022, you will be formed as a single-person group.
- Start early and proceed in steps. Read the project description carefully before you start.
- This assignment is worth 12% of your overall grade.

Project Description

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

1. Assume the whole B+-tree is kept in the main memory.
2. Assume the fanout of each node is 5 (i.e., 4 index/data entries per page).
3. Assume Alternative 2 is used for data entries.
4. You do not need to maintain actual data records (so the rids in data entries can be set to 0).
5. You need to handle search keys with the *string* type, which has at most 10 characters, and adopt the dictionary order to compare different search keys (e.g., "aa" < "aaa", "aaa" < "aab", and "abcd" < "bacd"). Our test cases only contain lowercase English characters ('a' to 'z').
6. You can implement this index by using any programming language (e.g., C, Java, C++, C#, etc.).

Group Members

Each group should have 1-2 members. You need to decide whether you want to work alone (easier to manage the progress but higher workload) or collaborate with another student (lower workload but more difficult to manage the progress). Note that each member of a group will be assigned the same score for this project.

Assignment Requirements

In this assignment, you need 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 need to implement are given below.

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 "closes" the index. This includes deallocating 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 (pid denotes the pointer id.) to be inserted into index nodes. You should always check to see if the current node has enough space before you insert. If the current node does not have enough space, your program has to split the current node by creating a new node, and copy some pairs from the current node to the new node. Splitting will cause a new entry to be added in the parent node.

Splitting of the root node should be considered separately. The main reason is that we need to update the root pointer to reflect its change if we have a new root. 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 whether a node is underflow (i.e., the fill-factor is less than 50%) after each deletion. If a node is underflow, your program needs to handle this issue by adopting the merge and redistribution operations (read and implement the algorithm in the lecture notes). As a remark, 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.

BTree::Search

This method implements range queries. Given a search range (*key1*, *key2*), the method returns all the qualifying key values in the range that are 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 need 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.

BTree::PrintTree, BTree::PrintNode

These operations should show the contents of the B+-tree structure, which can be used to understand whether the tree structure is correct. 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 the B+-tree test program…
Waiting for your commands:

insert abc
The key abc has been inserted in the B+-tree!

delete abc
The key abc has been deleted in the B+-tree. (or: The key abc is not in the B+-tree.)

print
print out your B+-tree here; the format is up to you, but it must be clear.

stats
Statistics of the B+-tree:
Total number of nodes: 9
Total number of data entries: 20
Total number of index entries: 3
Average fill factor: 60%
Height of tree: 3

quit
The program is terminated.

>

The data file (e.g., data.txt) containing the search key values has the format of one key value per line. An example data file is:

crab
shrimp
fish
coffee
tea
water
lemon
orange
apple
watermelon

Below are the commands your program should support:

insert <key>	Insert the key in the B+-tree
delete <key>	Delete the key in the B+-tree. If the key is not in the B+-tree, the program does not need to modify the tree structure.
search <low> <high>	Return the keys that fall in the range [low, high].
Print	Print the whole B+ tree (The format is up to you. However, it must be clear!).
Stats	Show the statistics of the B+-tree.
Quit	Terminate the program.

Submission Procedure

1) Zip the entire set of source code, video, report, and make files (if any), and upload it to BUMoodle. 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 and algorithms used in the implementation
   - 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

- **Video Demonstration (40%)**: Each group needs to submit a **video (at most 6 minutes)** for demonstrating different components of your code (e.g., insertion, deletion, and searching operations). This part is graded based on the following rubrics.
  1: Clearness of the presentation (15%)
  2: Representative of the test cases (10%) [The text cases are more representative if it can show more effects of a B+-tree. For example, can your test cases correctly show the split of internal nodes/leaf nodes/root nodes?]
  3: Language proficiency (10%)
  4: Number of components in the demonstration (5%)
- **Correctness (40%)**: 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 (10%)**: 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 we have detected the plagiarism case, all the students who are involved in this case will be given some penalty in the final grade and the zero mark to the project.**