CSI 2131 File Management — Winter 2004
Computer Science — University of Ottawa

Homework Assignment #3 (100 points, weight 10%)

Due: Thursday, April 1st at 5:00 p.m. (via webCT)

B tree Indexing

1 Problem description

This assignment is an extension of assignment #2. You will be dealing with the same datafile and some of the operations on it, but the index you will build will be organized as a B tree file, rather than a simple index in main memory.

Datafile description:
same as in assignment#2

B tree index file description:
The B tree file contains a header record with the following fields:

- m (1 byte), the order of the B tree
- L (1 byte), the number of levels of the B tree
- *root* (4 byte integer) RRN of the root node in this file

Other records (nodes) in the B tree file are of fixed length and contains the following fields:

- numKeys (1 byte integer): number of keys stored at the node.
- keys[0], key[1], ..., key[m-1] (7 characters each): m fields storing the keys (course codes)
- pt[0], pt[1], ..., pt[m-1] (4 byte integer each): m fields storing the pointers

Some of the fields will be filled with blanks when numKeys $< m$.
The records are placed one in each line. The header record is the first one.
Note that each pointer represents the RRN of the record it points to, for records at all levels except the leaf level. For records at the leaf level, each pointer represents the byte offset of the corresponding record on datafile (same information originally contained in the simple index for assignment #2). You know that a record is at the leaf level if its level is equal to L. Note that the byte offset in this assignment changed from 2 bytes to 4 bytes (long int).

Creating a class `Btree`
Class `Btree` should handle the basic operations on the B tree index file. Class `indexedFile` will simply request changes in the B tree by calling methods of this class.
This class should contain, among other things, the following methods:

- `createFile`: Given m and an opened fstream BtreeIndexFile (empty file), it updates the file to represent an empty B tree, associates this file with this Btree object and initializes any relevant variables in this class. When creating an empty B tree, we would have m as given, $L = 0$ and $root = -1$ stored at the header record. It is also useful to keep these values in variables inside the class.

- `loadFileInfo`: Given an opened fstream BtreeIndexFile, this method associates this file with this Btree object, and loads the necessary information from its header record into member variables in this class.

- `findLeaf` (private): Given a *key* (course code), finds the leaf that could contain the *key*, loading all the nodes in the path from the root to the leaf into an array stored in the class. Also store the RRNs for these nodes in another array.

- `search`: Given a *key* (course code), returns the byte offset of the datafile record containing the *key* or -1 if the search was not successful. This method should call `findLeaf`.

- `add`: Given a pair *(key,byteoffsetInDatafile)*, inserts the pair into the B tree, using the B tree insertion algorithm seen in class (the one that handles all the cases for insertion, including successive node splits). This method should call `findLeaf`.

Note that all the B tree index manipulations are done directly on the file by the class `Btree`; only a few nodes may be stored in this class (the current path from root to leaf, and a couple of auxiliary nodes to manipulate the splits before writing to the file).
Also note that you have been spared the work of deleting nodes from the B tree!

Modifying class `indexedFile`
Instead of an array inside this class for holding the index, you will have an object of the `Btree` class in order to manipulate the B tree index file.
Your class `indexedFile` will provide public methods to handle the following operations:

- Initialization:
  no major modification from assignment#2.

- Open indexed datafile: method `open(m)`
  This should open the datafile and prepare the B tree index file for usage. A new parameter, an integer m, will be provided for this class which will indicate the order of the B tree in case the B tree index file is being created, or it will be ignored if this file already exists.
  The B tree index file will have the same name as the datafile but with the prefix `index` before it.
  If the B tree index file exists, the method opens it and calls the `loadFileInfo` method of the Btree class object. If the B tree index file does not exist, the method creates it by opening a new file, and calling the method `createFile` of the Btree class object;

after that, it builds the B tree corresponding to the datafile by successively performing insertions using the method `add` in class `Btree`.

- Close indexed datafile: method `close`
  This operation will close the datafile and the B tree index file.

- Search for course: method `recordSearch`
  This operation will receive a course code and call the method `search` of the B tree index, and if successful, it will read and return the data record.

- Add record: method `addRecord`
  This operation will receive a new record to be added to the datafile. The B tree index file should be updated accordingly. Note that you must make sure that records for existing courses will not be added again.

- Delete record, Storage compaction, Update record and Simulated power off are removed from this assignment.

2 Implementation details and standards

Don't forget: your program will be compiled and run with the dotnet compiler; so it must successfully compile and run when using this compiler.

Program structure:
Your program would contain the same files as Assignment#2 (with the appropriate modifications to `indexedfile.*`, plus the files for the new class `Btree`, namely `btree.h` and `btree.cpp`. A **new main program** is provided to reflect changes.
Note that if your assignment#2 is not working very well or you are not sure of its correctness, you may use the solution for assignment#2 (provided by us) as the startup code for your assignment#3.

Documentation and Style
Please follow the same guidelines as in assignment#2.

What and how to submit your assignment
Create a directory/folder named `a3` containing the following files: `record.cpp`, `record.h`, `indexedfile.cpp`, `indexedfile.h`, `btree.h`, `btree.cpp`.
EVERY FILE MUST CONTAIN A HEADER WITH Student Name, Student Number, Course and Section. Zip the folder and submit the zipped file as your assignment#3 submission to webCT (only one file is submitted).