Alexandria University
Faculty of Engineering
Computer and Systems
Engineering Dept.
Second Year

File Structure
Assignment 4
Assigned: Tuesday, April 15th, 2014
Due: Sunday, April 27th, 2014

B+ Trees

Introduction

In this assignment, you will implement a B+ tree in which leaf level pages contain entries of the form **<key, list of rids of data records with search key>** (Alternative 3 for data entries, in terms of the textbook.) You must implement the full **search** and **insert** algorithms as discussed in class. In particular, your insert routine must be capable of dealing with overflows (at any level of the tree) by splitting pages; as per the algorithm discussed in class, you will not consider re-distribution. For this assignment, you can deal with deletes by simply marking the corresponding leaf entry as 'deleted'; you do not have to implement merging. You will be given, among all other necessary packages and classes, HFPage and BTSortedPage. BTSortedPage is derived from HFPage, and it augments the insertRecord method of HFPage by storing records on the HFPage in sorted order by a specified key value. The key value must be included as the initial part of each record, to enable easy comparison of the key value of a new record with the key values of existing records on a page. The documentation available in the java code documentation is sufficient to understand what operation each function performs. You need to implement two page-level classes, BTIndexPage and BTLeafPage, both of which extends BTSortedPage. These page classes are used to build the B+ tree index; you will write code to create, destroy, open and close a B+ tree index, and to open scans that return all data entries (from the leaf pages) which satisfy some range selection on the keys.

Design Overview

You should begin by (re-)reading the chapter *Tree Structured Indexing* of the textbook to get an overview of the B+ tree layer. There is also information about the B+ tree layer in the HTML documentation.

A Note on Keys for this Assignment

You should note that key values are passed to functions using KeyClass objects (an abstract class). The contents of a key should be interpreted using the AttrType variable. The key can be either a string(attrString) or an integer(attrInteger), as per the definition of AttrType in global package. We just implement these two kinds of keys in this assignment. If the key is a string, its value is stored in a StringKey class which extends the KeyClass . Likewise, if the key is an integer, its value is stored in a IntegerKey class that also extends the KeyClass . Although using the above structure, keys can be of (the more general enumerated) type AttrType, you can return an error message if the keys are not of type attrString or attrInteger.
The BTSortedPage class, which augments the insertRecord method of HFPage by storing

records on a page in sorted order according to a specified key value, assumes that the key value is included as the initial part of each record, to enable easy comparison of the key value of a new record with the key values of existing records on a page.

B+ Tree Page-Level Classes

These classes are summarized in Figure 1. Note again that you must NOT add any private data members to BTIndexPage or BTLeafPage.

- HFPage: This is the base class, you can look at heap package to get more details.
- BTSortedPage: This class is derived from the class HFPage. Its only function is to maintain records on a HFPage in a sorted order. Only the slot directory is re-arranged. The data records remain in the same positions on the page. This exploits the fact that the rids of index entries.are not important: index entries (unlike data records) are never 'pointed to' directly, and are only accessed by searching the index page.

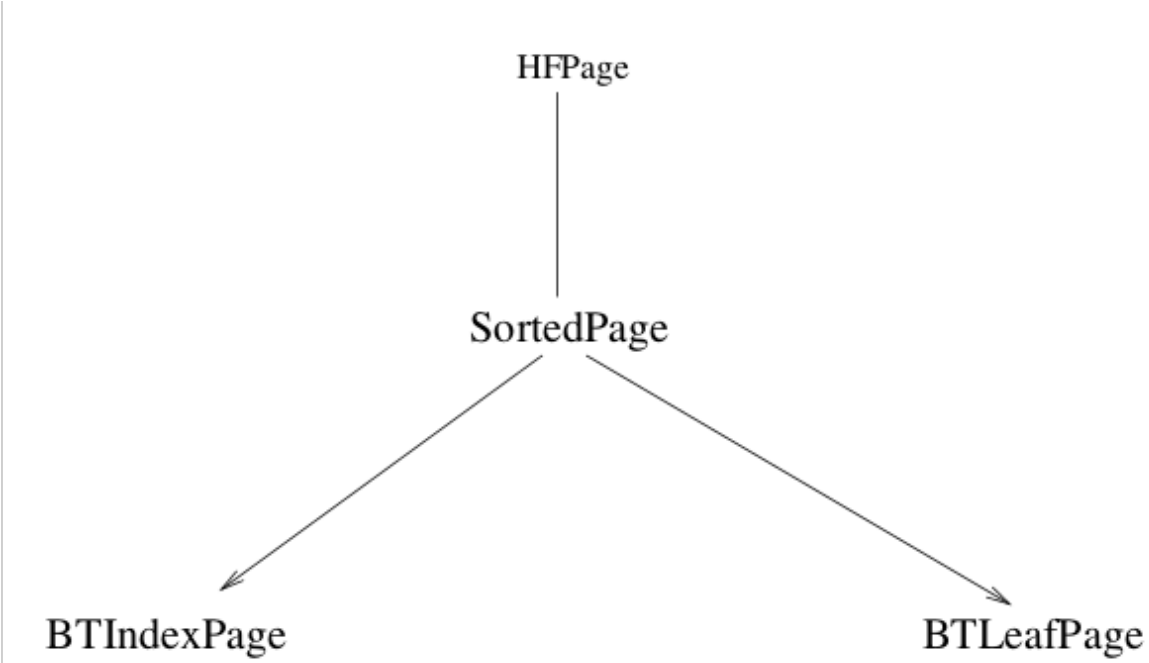

Figure 1: The C++ Classes used for the B+Tree pages

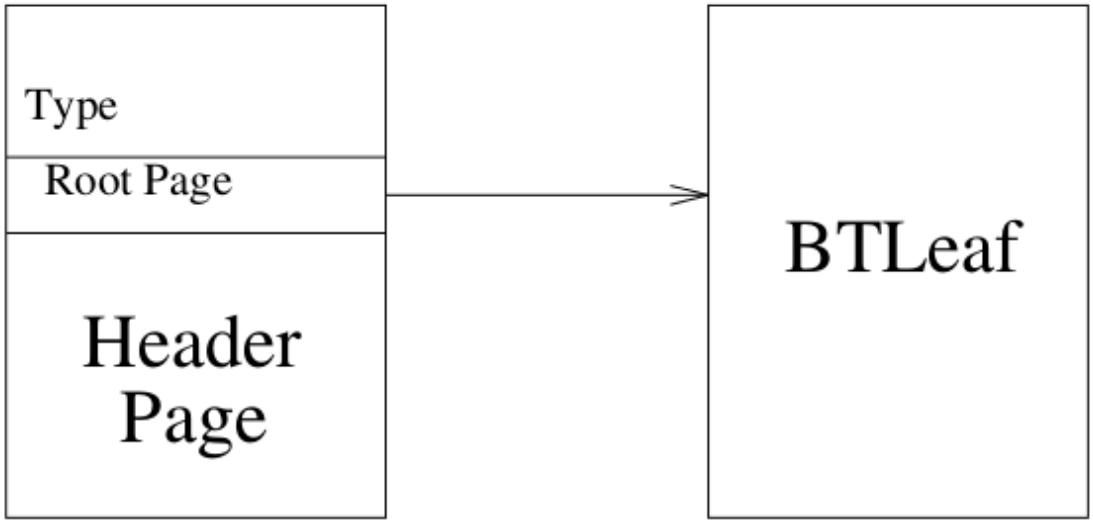

Figure 2: Layout of a BTree with one BTLeafPage

- BTIndexPage: This class is derived from BTSortedPage. It inserts records of the type key, pageNo on the BTSortedPage. The records are sorted by the key.
- BTLeafPage: This class is derived from BTSortedPage. It inserts records of the type key, dataRid on the BTSortedPage. dataRid is the rid of the data record. The records are sorted by the key. Further, leaf pages must be maintained in a doubly-linked list.

Other B+ Tree Classes

We will assume here that everyone understands the concept of B+ trees, and the basic algorithms, and concentrate on explaining the design of the Java classes that you will implement. A BTreeFile will contain a header page and a number of BTIndexPages and BTLeafPages. The header page is used to hold information about the tree as a whole, such as **the page id of the root page**, **the type of the search key**, **the length of the key field(s)** (which has a fixed maximum size in this assignment), etc. When a B+ tree index is opened, you should read the header page first, and keep it pinned until the file is closed. Given the name of the B+ tree index file, how can you locate the header page? The DB class (in diskmgr package) has a method

```
public void add_file_entry(String fname, PageId start_page_num);
```

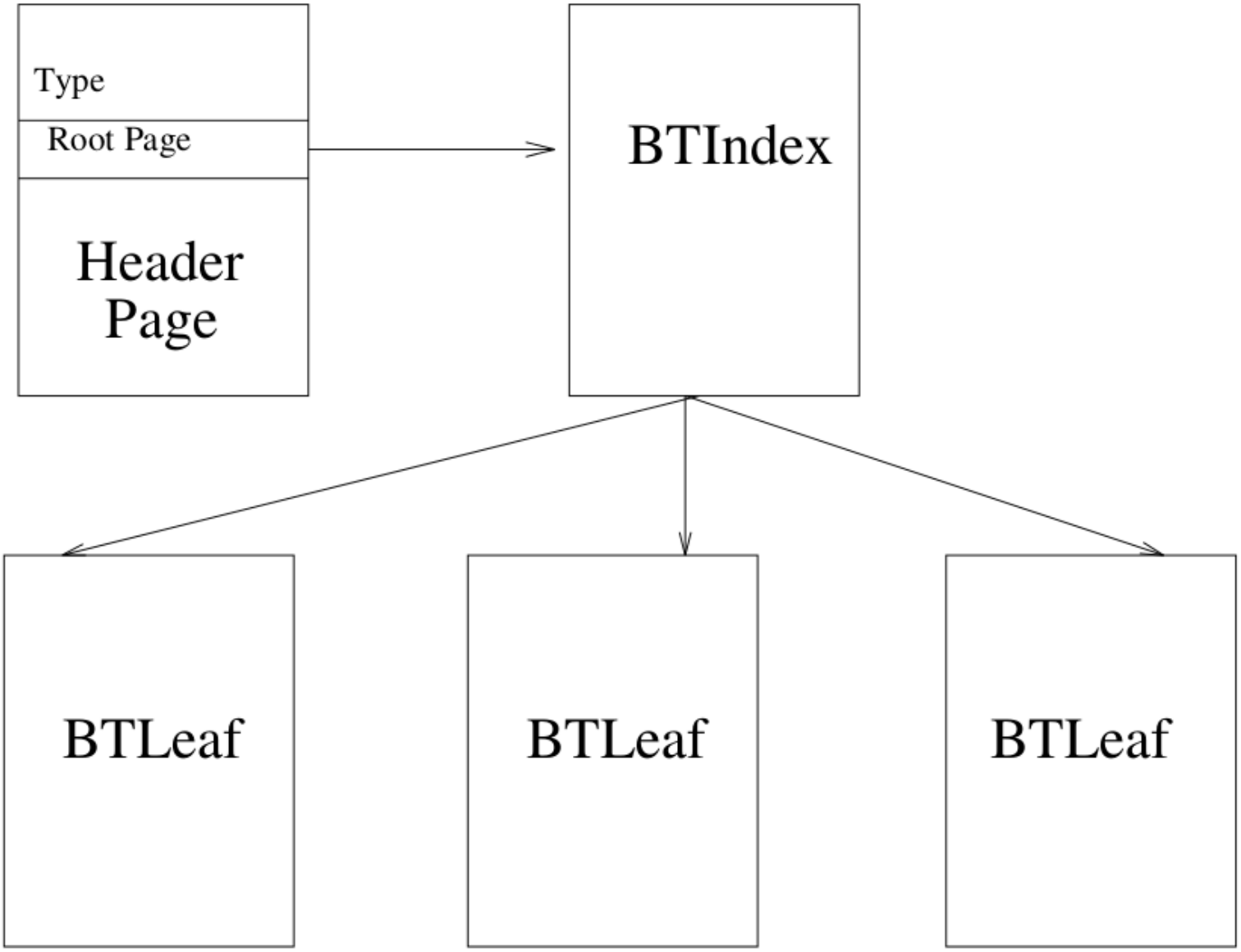

Figure 3: Layout of a BTree with more than one BTLeafPage

that lets you register this information when a file fname is created. There are methods for deleting and reading these 'file entries' (file name, header page pairs) as well, which can be used when the file is destroyed or opened. The header page contains the page id of the root of the tree, and every other page in the tree is accessed through the root page.

Figure 2 shows what a BTreeFile with only one BTLeafPage looks like; the single leaf page is also the root. Note that there is no BTIndexPage in this case. Figure 3 shows a tree with a few BTLeafPages, and this can easily be extended to contain multiple levels of BTIndexPages as well.

IndexFile and IndexFileScan

A BTree is one particular type of index. There are other types, for example a Hash index. However, all index types have some basic functionality in common. We've taken this basic index functionality and created a abstract class called IndexFile. You won't implement any methods for IndexFile. However, any class derived from an IndexFile should support IndexFile(), Delete(), and insert(). And you might have to modify the throws clause for the Delete() and insert() in IndexFile accordingly depending on how you should decide to handle the exceptions. Likewise, an IndexFileScan is a abstract class that contains the basic functionality all index file scans should support.

BTreeFile

The main class to be implemented for this assignment is BTreeFile. BTreeFile is a derived class of the IndexFile class, which means a BTreeFile is a kind of IndexFile. However, since IndexFile is only an abstract class all of the methods associated with IndexFile must be implemented for BTreeFile. The methods to be implemented are described under the btree package of the given java documentation. Specifically, you will be responsible to provide code for the following classes:

- BTreeFile class
  - constructor
    There are two constructors for BTreeFile: one that will only open an index, and the other that will create a new index on disk, with a given type and key size. You can pass 0 for the fourth parameter in the second constructor of BTreeFile because it is not within the scope of this assignment. If a page overflows (i.e., no space for the new entry), you should split the page. You may have to insert additional entries of the form key, id of child page into the higher level index pages as part of a split. Note that this could recursively go all the way up to the root, possibly resulting in a split of the root node of the B+ tree.
  - insert method
    If a page overflows (i.e., no space for the new entry), you should split the page. You may have to insert additional entries of the form key, id of child page into the higher level index pages as part of a split. Note that this could recursively go all the way up to the root, possibly resulting in a split of the root node of the B+ tree.

    - Delete method
      The Delete method simply removes the entry from the appropriate BTLeafPage. You are not required to implement redistribution or page merging when the number of entries falls below threshold.
- BTIndexPage class
  See the explanation in previous sections.
- BTLeafPage class
  See the explanation in previous sections.

Note that the java documentation provided to you for methods in BTreeFile, BTIndexPage, and BTLeafPage, do not specify whether they throw any exceptions; or rather, what exceptions that they throw. You should follow the error protocol defined as before to implement your methods with reasonable exceptions thrown.

BTreeFileScan

Finally, you will implement scans that return data entries from the leaf pages of the tree. You should create the scan through a member of BTreeFile, so that you can report an error if a BTreeFile is closed before a scan is completed.

Note that BTreeFileScans should support several kinds of range selections. These ranges are described in java documentation for BTreeFile class.