Carnegie Mellon University
15-415 - Database Applications
Spring 2012, Faloutsos
Assignment 3: Indexing (DB-internals)

Due: 2/23, 1:30pm, e-mail only

1 Reminders

- Weight: **20%** of the homework grade.
- Lead TA: U Kang
- There will be **two** recitation sessions for this assignment on Wed Feb. 15th and Feb. 22th, from 2:30pm to 3:20pm in Scaife Hall 219.
- Please post your questions to the blackboard (`http://www.cmu.edu/blackboard/`) or send them to `ukang AT cs.cmu.edu`. Please **do not** send any emails to the class mailing list.
- This is a large assignment. Estimated time: **~20 hours** (~5 hours to download, compile and understand the source code, ~15 hours to implement functions).

2 Preliminaries - Implementing a B+ Tree

This assignment is designed to make you more familiar with the B+ Tree data structure. You are given a basic B+ Tree implementation and you are asked to extend it by implementing some new operation/functions, that we list in subsection 3

The specifications of the basic implementation are:

1. It creates an "inverted index" in alphabetical order in the form of a B+ tree over a given corpus of text documents (explained in detail later).
2. It supports the following operations: insert, scan, search and print.
3. No duplicate keys are allowed in the tree. FYI: It uses a variation of "Alternative 3" and stores a postings list for each word that appears many times.
4. It **does not** support deletions.
5. The tree is stored on disk.

2.1 Where to Find Makefiles, Code, etc.

The file is available at `http://www.cs.cmu.edu/~christos/courses/dbms.S12/hws/HW3/db_asn3.zip` or through AFS (`/afs/cs.cmu.edu/user/christos/www/courses/dbms.S12/hws/HW3/db_asn3.zip`).

- Unzip the file.
- Type `make demo` to compile everything and see a demo.

The demo searches the key 'american' and shows the contents of the documents containing the key.

2.2 Description of the provided B+ tree package

The directory structure and contents are as follows:

- `DOC`: contains a very useful documentation of the code.
- `SRC`: the source code.
- `Datafiles`: sample data to add to the tree.
- `Tests`: some sample tests and their solutions.
- Some other useful files, *e.g.*, `README, makefile` etc.
- **IMPORTANT:** Files `B-TREE_FILE, POSTINGSFILE, TEXTFILE, parms` are created by our B+ tree implementation, when a tree is created (recall that the implementation is disk-resident). To allow `main` to access this tree (across multiple executions), make sure that these files are not deleted and are present in the same directory as `main`. Conversely, delete these files if you want to create a new tree.

In more detail, the main program file is called "`main.c`." It waits for the user to enter commands and responds to them as shown in Table 1.

3 Tasks

Our implementation provides each of the described functions in Table 1. In this assignment, you are asked to extend it to support the operations listed in Table 2.

ARGUMENT	EFFECT
`C`	Prints all the keys that are present in the tree, in ascending lexicographical order.
`i arg`	The program parses the text in `arg` which is a text file, and inserts the uncommon words (*i.e.*, words not present in "comwords.h") into the B+ tree. More specifically, the uncommon words of `arg` make the "keys" of the B+ tree, and the value for all these keys is set to `arg`. Since this tree enables us to find which words are present in which documents, it is known as the *inverted index*.
`p arg`	prints the keys in a particular page of the B+ tree where `arg` is the page number. It also prints some statistics about the page such as the number of bytes occcupied, the number of keys in the page, etc.
`s <key>`	searches the tree for <key> (which is a single word). If the key is found, the program prints "Found the key!". If not, it prints "Key not found!".
`S <key>`	Searches the tree for <key>. If the key is found, the program prints the documents in which the key is present, also known as the *posting list* of <key>. If not, it prints "Key not found!".
`T`	preTty-prints the tree. If the tree is empty, it prints "Tree empty!" instead.

Table 1: Existing interface

ARGUMENT	EFFECT
`m`	[**20 pts**] Print the minimum of all the keys. The output should contain a key.
`M`	[**20 pts**] Print the maximum of all the keys. The output should contain a key.
`n`	[**20 pts**] Print the number of all the keys. The output should contain a number.
`R`	[**20 pts**] Perform a reverse scan of a tree. The output is the list of the key in the reverse order, and each line of the output contains a key.
`k <ranking>`	[**20 pts**] Print the kth largest key. The output should contain a key. Assume the `ranking` is between 1 and the number of keys.

Table 2: Commands to be implemented and their weights

Clarifications/Hints

- For your convenience, we have provided you with sample tests and their corresponding outputs in `Tests`. To see if your implementation runs correctly on the test files, type
  - `make test_number`
  - `make test_minimum`

  `test_number` and `test_minimum` test your implementation for the 'n' and 'm' command, respectively. If `diff` is empty for both `test_number` and `test_minimum`, then your implementation passes the provided tests! However, we will use several other

(unpublished) test files during grading, so please also make sure to test your implementation on other test files of your own.

- See def.h for important data structure information.

4 Testing Mechanism

We will test your submission primarily for **correctness**, and secondarily for **efficiency**.

Correctness For correctness, an easy test is to run your code against the sample test files provided with the assignment. For each test file the output from your program should be exactly the same as the solution output (*i.e.*, `diff` is empty). Make sure you test your code on additional datasets of your own. **Note:** We will use extra (unpublished) test cases to grade.

Efficiency For efficiency, the goal is to have linear algorithms for the R, n and k operations, and logarithmic algorithms for m and M.

Format We will use scripts to test the output of your code. Therefore, please make sure your output follows the same format as the sample test file solutions. That is, the result of `diff` between your output and the provided outputs, should be empty.

5 What to hand-in

1. Create a tar file of your complete source code including **all and only** the necessary files as well as the `makefile` (i.e., exclude *.o *.out etc files) and
2. mail it to `ukang AT cs.cmu.edu`. with subject "**submission homework 3**".

Reminders:

- Please make sure that `make` compiles everything.
- The points for each question are as in Table 2, 80% of which goes for correctness, and 20% for efficiency.