

Carnegie Mellon University
15-415 - Database Applications
Fall 2009, Faloutsos
Assignment 5: Indexing (DB-internals)
Due: 10/8, 1:30pm, e-mail only

1 Reminders

- Weight: **20%** of the homework grade.
- Out of **100** points.
- Lead TA: Leman Akoglu
- There will be **two** recitation sessions for this assignment on Wed Sep 30th and Oct 7th, from 2:30pm to 4:20pm in Doherty Hall (DH). We will be in DH 2122 from 2:30pm to 3:20pm and in DH 2105 from 3:30pm to 4:20pm.
- Please post your questions to the blackboard (<http://www.cmu.edu/blackboard/>) or send them to lakoglu 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, ~5 hours to implement ‘range’ functions (R,r) and ~10 hours to implement ‘all pairs’ functions (A,a)). Please start early.

2 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.

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-F09/hws/hw5/db_asn5.zip or through AFS (`/afs/cs.cmu.edu/user/christos/www/courses/dbms-F09/hws/hw5/db_asn5.zip`). Untar the file. Type `make demo` to compile everything and see a demo.

SANITY CHECK: Type `make sanitycheck`, which compiles the source code, inserts words into the B+ tree from the files in `Datafiles` and then runs the search query `S travolta`. The output of the search query is saved in `Tests/sanitycheck.out` and compared with the expected output in `Tests/sanitycheck.sol` using `diff`, which is empty.

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.
- `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.

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

Table 1: Existing interface

2.3 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.

Clarifications/Hints

- There is a subtle difference between `CompareKeys` and `strdist`. The former, `CompareKeys(A, B)` returns (1, 0, -1) and tells which of the given strings is larger (zero, for tie). On the other hand, `strdist` tells us the distance of the two strings, roughly as if they were numbers in base-128 (see code for details). *E.g.*, `strdist('a', 'b') = -1`, `strdist('b', 'a') = 1`, `strdist('cc', 'ca') = (2.2)128 - (2.0)128 = 0.015625`.
- 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 hw5`. If `diff` is empty for both `test_range` and `test_allpairs`, 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.

3 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 avoid sequential scans. In particular, for the 'range' functions the effort should be proportional to the size of the output, *i.e.* $O(k)$ for k keys in the range. Also, for the 'all pairs' functions the effort should be less than $O(N^2)$. *Hint: Try to use recursion.*

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.

4 What to hand-in

1. Create a tar file of your complete source code (including **all and only** the appropriate files as well as the `makefile`) and
2. mail it to `lakoglu AT cs.cmu.edu`. Please make sure that the subject of your email is “**submission homework 5**”.

Reminders:

- You must code the new functions in files (from top to bottom in Table 2) `keysInRange.c`, `countKeysInRange.c`, `allPairsWithin.c`, `countAllPairsWithin.c` in SRC.
- Please make sure that `make` compiles everything.
- The points for each question are as in Table 2, 50% of which goes for correctness, and 50% for efficiency.

ARGUMENT	EFFECT
R <key1> <key2>	<p>[30 pts] Perform a range search and print the keys between two given keys in the B+ tree. Please note that the interval is closed ($\langle \text{key1} \rangle \leq \dots \leq \langle \text{key1} \rangle$), i.e., if any of the given keys $\langle \text{key1} \rangle$ and $\langle \text{key2} \rangle$ exist in the tree, they should also be printed. Print nothing if the tree is empty, or if there exists no keys within a given range.</p> <p>E.g. If the tree contains three keys, “a”, “b”, “c”, the range search for keys “ab” and “bc” should return</p> <p style="padding-left: 40px;">b</p> <p>The range search for “a” and “d” should return</p> <p style="padding-left: 40px;">a b c</p> <p>The range search for “d” and “z” should return</p> <p style="padding-left: 40px;"><code><Empty string></code></p>
r <key1> <key2>	<p>[10 pts] Perform a range search and print the count of the keys between two given keys in the tree. Please note that the interval is closed as discussed in the above function. Print 0 if the tree does not contain any keys or there exists no keys within a given range.</p>
A < ϵ >	<p>[40 pts] Print all the pairs of keys within distance ϵ to each other in the tree for a given ϵ.</p> <p>Things to consider in your implementation are:</p> <ol style="list-style-type: none"> 1. Print nothing if the tree contains zero or one keys, or if it does not contain any pairs within the given ϵ. 2. Make sure that you do not print self-pairs (e.g. {“a”, “a”}). Also, you should print only one of the mirror-pairs (e.g. {“a”, “b”} and {“b”, “a”}). In general, please print only those pairs where the 1st key is strictly greater than the 2nd key ({“a”, “b”} in the above example). 3. A distance metric for the string-based keys is given in the “strdist” function.
a < ϵ >	<p>[20 pts] Print the count of all the pairs of keys within distance ϵ to each other in the tree for a given ϵ. Please make sure to consider the items enumerated in the above function while counting. Print 0 if the cases in item 1 hold.</p>

Table 2: Commands to be implemented and their weights