COP 5536 Fall 2012

Programming Project

Due Date: October 26, 2012, 11:55 PM Eastern Time
Submission Website: Sakai

1. General

1. Problem description
The purpose of this project is to compare the relative performance of AVL, red-black, and B-trees as well as their counterparts with a hash table front end. We shall focus only on the search and insert operations and require all keys to be distinct.

2. Programming Environment
You may implement this assignment in Java or C++. Your program must be compilable and runable on the Thunder CISE server using gcc/g++ or standard JDK. You may access the server using Telnet or SSH client on thunder.cise.ufl.edu.

3. Steps to Follow
a. Code and debug the class AVL.
b. Code and debug the class AVLHash. This class uses a hash table with each slot containing an AVL tree (or a pointer to an AVL tree). The hash table, itself, is a one-dimensional array whose size, s, is specified by the user. Use the hash function key mod s. Note that the search method for AVLHash, for example, simply does a search in the AVL tree stored in the slot “key mod s”.
c. Code and debug the class RedBlackHash, which is similar to AVLHash except that each slot is a red-black tree (or a pointer to a red-black tree). Do not write your own red-black code. Instead use either TreeMap(Java) or map(C++). Documentaries of these classes can be reachable from the links below.
http://www.cplusplus.com/reference/stl/map/
http://docs.oracle.com/javase/6/docs/api/java/util/TreeMap.html
d. Code and debug the class BTree. The BTree order should be a parameter to the class constructor.
e. Code and debug the class BTreeHash, which is similar to AVLHash except that Btrees are used in place of AVL trees.

2. Input/Output Requirements

1. Running modes:
The name of your program should be **dictionary.cpp** for C++ and **dictionary.java** for Java. Your program **MUST** support all of the following modes.
(i) random mode:
Your program should input the number n, generate a random permutation of the keys 1 through n, insert these n keys and associated values in this order into each of AVL, AVLHash, RedBlack, RedBlackHash, BTree and BTreeHash (assume the value associated with key k is 2k) then search for each of the n keys in the inserted order. You should repeat this random permutation/insert/search experiment 10 times and report the average time taken for the inserts as well as for the searches by each of the six data structures. You may use any random permutation code you have access to. For example, the STL function random-shuffle may be used. The command line for this mode is:

$ dictionary –r s b-tree-order

(ii) user input mode:
Get the (key, value) pairs from the input file assuming keys and values are non-negative integers. Input file starts with an integer, n, followed by n lines. Each line includes a key and a value separated by a space. Insert the records into each of the structures AVL, AVLHash, BTree and BTreeHash one by one. Run with the operation sequence from input file. For the structures that employ a hash table, assume that s=3 and for those that employ a B-tree, assume that the B-tree order is 3. Create the files "AVL_inorder.out", "AVL_postorder.out", "AVLHash_inorder.out", "BTree_sorted.out", "BTree_level.out" and "BTreeHash_level.out". Then output all values of the records to the corresponding output files. Values should be separated by a space character and trees should be separated by a new line character. Within a tree, the values should be listed in both inorder and postorder, for AVL and red-black trees and in sorted and level orders for B-trees in the non-hashed structures, inorder and level order for hashed structures.

The command line for this mode is:

$dictionary -u file-name // read the input from a file 'file-name'

3. Submission

The following contents are required for submission:

1. Makefile: If you do not use this file, provide detail instructions on how compile and run in your REPORT file.
2. Source Program: Provide comments.
3. REPORT:
    - The report should be in PDF format.
    - The report should contain your basic info: Name, UFID and UF Email account (Not CISE account).
    - State what compiler you use, how to compile, and etc.
    - Function prototypes showing the **structure** of your programs.
    - A summary of result comparison: You should put first your expectation of the comparison before running your program: i.e. what you think about the relative performance of each scheme, and why.
    - **Please include the structure of your program. List of function prototypes is not enough.**
4. Experiment with your BTree codes (d and e) in random mode with n = 1000000 and determine the optimal BTree order. Your report should include the insert and search times for a few of the Btree orders experimented with.
5. Experiment with your hashed structures with s values 3, 11 and 101. Your report should include the insert and search times for these s values experimented with.
6. Run your program (all codes) in random mode for n = 1000000 and tabulate the reported average insert and search times for each of the six structures (one table for inserts, another for searches). Show these average times also using bar charts. For the BTree codes, use the optimal BTree order determined in 4. and the s value determined in 5.
7. Based on your experiments with n=1000000, what recommendations can you make about a good way to implement a dictionary whose expected size is 1000000? Consider environments where

worst-case performance is important as well as those in which only expected performance is important. How about environments where in addition to insert and search we also want to support nearest match searches?

To submit, Please compress all your files together using a **zip** or the **tar** utility and submit to the **Sakai** system. You should look for Assignment->ADS Project for the submission. Your submission should be named ***LastName_FirstName.*zip(tar)**. Please make sure the name you provided is the same as the same that appears on the Sakai system.
Please DO NOT submit directly to a TA**. All email submission will be ignored without further notification.**
Please note that the due day is a hard deadline. No late submission will be allowed. Any submission after the deadline will not be accepted.

4. Grading Policy

Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.
Correct implementation and execution: 60%
Comments and readability: 15%
REPORT: 25%

5. Miscellanies

*Y*our implementations should be your own. **You have to work by yourself for this project (discussion is allowed).**

Measuring execution time (C++)
You can measure execution time like below:

```
#include <time.h>
clock_t Start, Time;
Start = clock();
.......... (your algorithm)
Time = clock() - Start; // time in micro seconds

Measuring execution time (Java)
long start = 0;
start = System.currentTimeMillis();
....................(your algorthim)
stop = System.currentTimeMillis();
// execution time for your algorithm.
time = stop - start;
```