CS2351 Data Structures

Homework 4

Due: 11:59 pm, June 1, 2010

In this assignment, you may choose to work alone or work in pairs, though the minimum requirements will be different. If you work in pairs, please submit your file together as one copy, and mark clearly your names and your university numbers in the submitted file.

Part I. Written (50%)

1. Consider the following sequence of operations on a simple BST (no balancing involved):

   Insert 2; Insert 8; Insert 5; Insert 13; Insert 12; Insert 15; Insert 4; Delete 5.

   (a) Show the structure of the BST after each steps.

   (b) Suppose we use an AVL tree instead of the simple BST. Show the structure of the AVL tree after each steps.

2. In a standard AVL tree, each node maintain a balance factor, which measures the difference in height between its left subtree and its right subtree. In addition, the balance factor in each node is required to be either 0, 1, or -1. Consequently, the height of the AVL tree with n nodes is always bounded by $O(\log n)$.

   Now, suppose we allow the balance factor of each node to be 0, 1, 2, -1, or -2. That is, the maximum height difference between the left and the right subtrees can be 2. Show that after this change, the height of the AVL tree is still bounded by $O(\log n)$.

3. Suppose we want to extend the functionality of a BST to support the following `rank` query. Given a value x, `rank`(x) returns the number of values smaller than or equal to x in BST.

   Show how to store $O(1)$ amount of extra information in each node of the BST such that `rank`(x) can be answered in the same time as we search for x in the BST, while the time for `search`, `insert`, or `delete` times remain unchanged.

   Remark. You will also need to explain how to update this extra information during `insert` or `delete`.

4. Suppose we use a hash function h to map n distinct keys into a table T with m entries. The number of collisions is defined as the size of the set $\{\{k, \ell\} \mid k \neq \ell \text{ and } h(k) = h(\ell)\}$.

   (a) In the worst case, what is the maximum number of collisions?

   (b) Assuming simple uniform hashing, what is the expected number of collisions?
   Hint: Define a random variable X_{ij}, with $i < j$, such that its value is 1 if key i and key j collide, and 0 otherwise.

Part II. Programming (50%)

In this part, you are asked to implement the simple BST discussed in the class, and the extended BST with `rank` query in Question 3 of Part I. If you work alone, you can just implement

the simple BST. If you work in pairs, you will need to implement both data structures.

You can develop your code using either C or C++. Your code should be compilable using Dev-C++. We will assess your code by the correctness (35%) and the readability (15%). Note that you should not use any predefined system calls or standard template library calls to perform the operations.

Specifications:

Your program of both implementations will expect two arguments when it is tested. The first one gives the filename of the input file that contains a sequence of operations and queries, and the second one gives the filename of the output file where the answers of the queries are stored. For example, suppose your program name is `BST`, the input filename is `input.txt`, and the output filename is `output.txt`. Then when we test your implementation, we will type: `./BST input.txt output.txt` so that your program performs the operations specified in `input.txt` and store the results in `output.txt`.

Each line in the input file contains a command of the following form:

`INSERT` x
`DELETE` x
`SEARCH` x
`RANK` x
`PRINT`

where x is an integer. We assume that we start with an empty BST. The command `INSERT` x inserts the integer x into the BST. The command `DELETE` x deletes x from the BST. The command `SEARCH` x prints `YES` to the output file if x is found in the BST, and `NO` otherwise. The command `RANK` x prints to the output file the number of integers in the BST whose value is at most x; this command will only appear in the testing of the extended BST. The command `PRINT` performs a pre-order traversal in the BST, and prints the integers in the nodes, separated by a space, to the output file. For example, the input file may look like:

```
INSERT 3
INSERT 8
INSERT 7
INSERT 9
PRINT
RANK 6
RANK 7
DELETE 3
RANK 6
SEARCH 5
SEARCH 9
PRINT
INSERT 13
```

Note that only `RANK`, `SEARCH`, or `PRINT` require us to print something to the output file. We assume that the result of each of these commands is printed on a separate line. Then the output file of the previous example should look like:

```
3 8 7 9
1
2
0
NO
YES
8 7 9
```

Additional Requirement:

When you implement `delete`, use the following rules discussed in the class: If we are removing a leaf, remove it. Else if we are removing a node which has only one child, we remove the node directly while connect its original parent to the child. Else if we are removing a node which has two children, we replace it by the *successor* and remove the successor instead.