Algorithm Design, Anal. & Imp., Programming Assignment

Out: April 05, 2021, Due: May 03, 2021

Note:

- This assignment will carry 15 points towards your final score.
- You can use any of the following languages (C++, Java).
- You can complete the assignment in a group of two students. If you want to work individually, you are welcome to do so.
- You need to submit source code, which should run on a standard Linux machine. Your output should match exactly as outlined in this document.

Data Structure for Dynamic Set

Background

Various data structures exist for storing dynamic set. In class, we covered binary search tree (BST), and red-black (R-B) tree. If the tree has a height of h, the BST takes $O(h)$ to perform each of the dictionary operations, such as search, insertion, and deletion. On the other hand, by balancing the tree, an R-B tree guaranty $O(\lg n)$ time to perform these operations, where n is the total number of elements in the tree. There are other balanced search trees, such as AVL tree and 2-3 Tree, which also guaranty a $O(\lg n)$ worst case performance for each of the dictionary operations. However, AVL tree and 2-3 tree typically perform worse than an R-B tree, because every insertion and deletion in these trees leads to a balancing scheme; whereas for many insertion or deletion, an R-B tree may not need to execute the balancing sub-routine.

Besides the above, there exist two additional data structures that also provide $O(\lg n)$ time bound for dictionary operations, but not for an worst-case analysis. The first among these are skip-list, a randomized data structure which provides an expected running time of $O(\lg n)$ to perform the dictionary operations. The second is splay tree which provides an amortized cost of $O(\lg n)$ time [1].

In this assignment, we will do a comparative analysis among these data structures. You need to compare between BST, R-B tree, splay tree, and skip-list.

[1] I will uploaded two pdf files in Canvas that will explain these two data structures.

Input File

The input file will list one operation on each line. It will have two tab-separated fields. If the first field has a value of 1, the desired operation is search or insert depending on whether the object is already in the data structure or not. If the value is 0, the desired operation is delete; if the object is not available in the tree, raise an exception and exit with a message that "delete object in Line XX not found in the tree." The second field is the key value that you need to search/insert/dete in the data structure. The key value can be real, integer, or string, for which $<$ is defined by the programming language (C++ or Java). Here is an example input file with string key value

```
1 apple ipod
1 t60 laptop adaptor
1 verizon phones
1 cordless phones
0 t60 laptop adaptor
```

Output

There should be two executables for your submission. The first of the program should output the elements in the tree in the sorted order and also print the total time (wall clock time in miliseconds) that your program takes to execute the operations listed in the input file. The second of the program (does not apply for skipList) should output (on the computer screen) the height of the tree after each operation. Each height should be printed to standard output in separate lines. We will write scripts that will match output from your program with the correct output to verify the correctness of your implementation. Note that, for a given sequence of operations and a given data structure, the generated tree should be identical.

Implementation

- You should implement a generic class with template parameter for each of the above data structures, the class should provide search, insert, and delete operation. You need to implement the data structures from the scratch.
- You should write two driver programs for the two executables. Each driver program should have a command-line parameter to take two inputs: first an integer, and second an input filename. The integer defines which data structure we want to use for a runtime session; the mapping is: 0 for HashTable, 1 for BST, 2 for splay tree, 3 for skip list, and 4 for R-B tree. The string is the name of data input file.

  Example invocations

  ```
  C++: ./MyMainV1.out 0 data_input_file
  Java: sh run.sh 0 data_input_file
  ```

The input file stores the operations (one operation in a line) to be performed on an initially empty data structure. You can assume that the data type is one of the following three: double, integer, or string. To allow us selecting a particular data type write three separate lines of code in your main file. We will select the data type(double, integer, string) by uncommenting a particular line. In submitted version, these three lines should be commented out. You should provide the line number corresponding to each type in files named comment_lines_v1.txt and comment_lines_v2.txt for corresponding versions. The format of the comment_lines_v1.txt and comment_lines_v2.txt should be like below.

```
line number to uncomment to test the double data type
line number to uncomment to test the int data type
line number to uncomment to test the string data type
```

The name of the main program file should be "MyMainV1.cpp" for c++ and "MyMainV1.java" for java.

- For C++, you should have a makeflie named "makefile.txt" which produces two executable files named "MyMainV1.out" and "MyMainV2.out" for the two versions.

  For Java, you should provide shell scripts to compile your program for the two versions. The shell script file should have the names "buildV1.sh" and "bulidV2.sh". To run the java program, you should provide two more shell scripts named "runV1.sh" and "runV2.sh".

- For BST, splay tree, and R-B tree, you will write a second version of the data structures, in which you will augment the data structure so that we can compute a tree height in O(1) time. We will use this feature to verify the correctness of your implementation.

Deliverables

Please submit the source files in a tarzipped folder through Canvas. The folder should be named as *lastname1_lastname2* and the submission file should be named as *lastname1_lastname2.tgz*. The folder should have a README with instructions to compile and run the program. Also, for the case of group submission, please highlight the contribution of each member of the team in the README file. Program will be graded by automated scripts, so it is important that you follow the submission guideline carefully. You may lose substantial part of your score if you do not follow the formatting instruction.