CS 450 - Design & Analysis of Algorithms

Date November 18 11:59 pm

Submit to Blackboard

Programming Project:

200 points (+ 50 Extra Credit)

Note and Extra Credit - The FLATTEN routine is complicated. Programs that run successfully and implement a reasonably efficient routine (not FLATTEN) to rebuild the tree will receive 200 points. Students that successfully implement *flatten* will get an extra 50 points. Please include in your notes to the grader if you implement FLATTEN.

Scapegoat tree http://en.wikipedia.org/wiki/Scapegoat_tree

"In computer science, a scapegoat tree is a self-balancing binary search tree … It provides worst-case **O(log n)** lookup time, and **O(log n)** amortized insertion and deletion time. Unlike other self-balancing binary search trees that provide worst case **O(log n)** lookup time, scapegoat trees have no additional per-node overhead compared to a regular binary search tree. This makes scapegoat trees easier to implement and, due to data structure alignment, can reduce node overhead by up to one-third."

In the programming language of your choice between Java and Python (given the grader can easily run your code on the CS lab machines), write an executable version of Scapegoat. Use the explanation of Scapegoat, implementation suggestions and pseudo code that are in Scapegoat Trees: the original publication describing scapegoat trees also see Igal Galperin, Thesis: On Consulting a Set of Experts and Searching (full version paper) Scapegoat stars at page 78.

Code Conventions

- Use the *highest* possible ancestor in the tree as the scapegoat.
- See class slides for details

Program Details

- **Input:** A file containing a list of commands and inputs for those commands. The input file will have a new command on each new line. You can assume all tree values are positive integers and that there are no duplicates. Call this file **tree.txt**
- **Output:** Print the output as specified below.

Input Commands are:

- **BuildTree** *alpha*, *key* - Assume that the input file contains one call to **BuildTree** on the first line. This call creates a new tree with *alpha* weight and a first node containing *key* as a value. All operations for that input file are on this tree.
- **Insert** *key* – given an integer *key*, create a new node with *key* value; and insert it into the tree.
- **Search** *key* – find a specified *key* in the tree if it exists, otherwise give an error message.
- **Delete** *key* – delete the specified *key* from the tree.

- **Print** – prints the tree structure. Your output does not need to be fancy but should be fairly easy for the grader to interpret as a tree structure.
- **Done** – exit the program.

Note:

The Scapegoat spec says: **Use the *highest* possible ancestor in the tree as the scapegoat.**

It seems that the definition of the *highest* ancestor for the scapegoat is unclear. It can be interpreted as being the first unbalanced node closest to the inserted node that unbalanced the tree or as the node closest to the root that was unbalanced.

To clarify this, the *highest* means the closest scapegoat to the root of the whole tree. The root should also be fine. But looking in http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-700.pdf it seems that they did NOT use the root in their example when picking the highest node. The idea of using a node high in the tree (nearer the root) is that it may trigger rebuilds/re-balancing of larger sections of the tree, but then overall, rebuilds should happen less frequently.

You may use any open source code for a pretty printer that you find, just be sure to cite the source in your comments and give clear instructions for assembling and running your code. An inelegant routine is given below. Given the root of a binary search tree and level = 0, it prints out a tree "on its side". All code aside from a tree printer must be your own.

```
INORDER-TREE-WALK(x, level)
 if x ≠ nil
        INORDER-TREE-WALK(x.left, level+1)
        print on a new line, level many indentations then
        print x.key
        INORDER-TREE-WALK(x.right, level+1)
```

So the tree

```
     4
    / \
   2   5
  / \
 1   3
```

Should print on its side as

```
                1
        2
                3
4
        5
```

Submitting your solution

Turn in electronically to Blackboard: All of your source code, documentation, etc. archived in a single compressed file (.zip or .tar). Include a text file named **README** that includes

1. Your name and an email address you can be contacted at.
2. A brief description of what you are submitting.
3. A description of how to build and use your program on the WSU system.
4. A list of files that should be in the archive, and a one line description of each file.

Make sure you thoroughly test your code.