implement search trees from scratch (i.e. no use of predefined coding routines to perform tree actions).

Create and compare operations on BST, AVL and Splay trees.

There will be a separate output file for each type of tree and a separate file for summary results. The summary results file should be opened in append mode so all your results are in a single file.

The user will enter the title of the test (describes test scenario e.g. “Random Searches” “80% Searches on 50% of numbers” “70% Actions Inserts” etc.)

The user will enter the names of the two input files and the four output files.

Reading from a file (file doesn’t exist?, empty? – print message and continue), insert a series of numbers into each of the trees, printing the tree to a file and to the screen each time along with the tree height after each insert. Test case 1 is below, a set of numbers you need to insert into the three trees, in that order. The beginning of your output should look similar to the output below for the BST.

NOTE: this is only ONE test scenario ….. you will be part of a team to create additional test cases to cover all the test scenarios. You utimately will run YOUR teams test cases AND the required test cases.

The printTree() function can display your tree in whatever way you choose, the tree needs to be easy to read. Show the tree, and tree height, after every insert.

The trees should show one after the other on the screen and in the output file.

Example output after first 6 inserts for BST ...

Test 1: Random Searches [display test number and title of test]

Insert 9

9

Tree height is: 0

Insert 7

9

7

Tree height is: 1

Insert 14

9

7 14

Tree height is: 1

Insert 25

9

7 14

25

Tree height is: 2

Insert 2

9

7 14

2 25

Tree height is: 2

Insert 23

9

7 14

2 25

23

Tree height is: 3

The AVL Tree will need to show, after every insert, that it rebalances correctly and maintains the structure of an AVL tree. You will need to implement functions for the rotations: LL, RR, LR, RL.

The Splay Tree will need to show after every insert that the new node rotates to the root. Implement functions for the rotations: Zig, Zag, Zig-Zig, Zag-Zag, Zig-Zag, Zag-Zig.

For the output of the AVL and Splay Trees you also need to list the number and types of rotations performed to create the new tree.

NOTE: it is expected to see each rotation combination shown in test cases – i.e. what does tree look like, what value is inserted, what does tree look like after insert and rotation(s) used.

Your program will then process a file of searches, inserts and deletes from the trees.

The file will contain an action code: S – search, I – insert, and D- delete (what if it isn’t one of those?????) and the number for that action

As above – if rotations occur because of the action you need to display them.

Splay example: Search 27 – found at level 5 with parent 25 – rotations zig-zig, zig-zag, zig

NOTE: Account for twin chain ……. just add duplicate node to twin chain (i.e. no additional data) ---- display of node 9 with no twin chain is: 9; with twin chain of 2 is: 9 (2)

Display the action and value and what tree looks like after the action takes place. You are also to print the number of operations (and rotations if AVL or splay tree) to perform the action.

Keep track of the actions and the number of operations in a 2-dimensional array (row = action, columns = BST, AVL, Splay). When no more actions print the table and sum the columns by search, insert, delete and overall.

Title of Test

Operation Counts

BST AVL Splay

Search aa bb cc

Insert dd ee ff

Delete gg hh ii

--------------------------

Total jjj kkk lll

Create a summary table of all the test cases runs (test case, search, insert, delete, overall) and write a report summarizing your results.

Summary

Operation Counts

Test Title BST AVL Splay

1 Title of Test 1 lll mmm nnn

2 Title of Test 2 ooo ppp qqq

…

n Title of Test n uuu vvv www

--------------------------

Total of all n tests xxxx yyyy zzzz

NOTE: Remember a Splay Tree should work best if 80% of the accesses are to 20% of the items. Should you have different tests to prove this?????

Required Test Scenario:

9

7

14

25

2

23

4

18

10

27

3

22

30

24

26

21

1

28

8

19

16

17

26

12

29

6

10

5

15

20

26

13

11

28

S 11

S 26

S 28

S 1

S 7

Q 5

I 17

D 22

S 11

D 16

D 28

I 15

D 33

S 29

S 17

Z 6

S 11

D 17

I 17

D 25

D 16

I 33

D 26

S 17

S 16

D 26

D 28

S 28

S 22

D 26

S 7