LAB ASSIGNEMENT 5 FOR ADA

M. Tech. (I.T.) – 1st Year

Deadline: 13-September, 2018, 04:00 PM

Important Instructions: ***You are required to submit hand-written copies of your algorithms and their time complexity analysis to the TAs in the LAB. Although the assignments are group wise, each student needs to submit these handwritten copies individually. Deadline for submission of these hand-written copies are same as the deadline of the assignments, and those will be collected by our TAs during your program evaluations in the LAB.***

You need to explain your algorithms and codes and time complexity analysis by yourself. You have to generate suitable input and output values (appropriate test cases) to demonstrate the correct functioning of your program. Not only the final output, you are supposed to display (print) the snapshots of your data-structures after every step of execution of your program (for example, after every iteration, in an iterative algorithm).

NOTE: LAB Assignments are group wise but their evaluations would be individual. TAs are requested to ask sufficient questions to each individual member of a group and assign marks according to their individual performances. However, all the members of a group may show/submit a single copy of their programs.

ASSIGNMENT OF PROBLEMS TO GROUPS: -

GROUP-X = Problem No. X.

(Example: Group-1 = Problem No. 1, Group-2 = Problem No. 2., … , Group-20 = Problem No. 20)

Groups are as defined by TA Fahiem in his mail (declaring results for Tutorial 2)

---

1. Write a C/C++ program to determine if a given binary tree is
   a. Strictly binary
   b. Complete
   c. Almost complete

   Analyze time complexity for all the algorithms you have written.

2. Two binary trees are similar if they are both empty or if they are both nonempty, their left trees are similar, and their right trees are ***similar***. Write a C/C++ program to determine if two binary trees are similar. (Ignore the info field of the nodes, you have to compare only the structures of the trees for the similarity.) Analyze time complexity for all the algorithms you have written.

3. Two binary trees are mirror similar if they are both empty or if they are both nonempty and the left subtree of each is ***mirror similar*** to the right subtree of the other. Write a C/C++ program to determine if two binary trees are mirror similar. (Ignore the info field of the nodes, you have to compare only the structures of the trees for the similarity.) Analyze time complexity for all the algorithms you have written.

4. Two binary trees are ***similar*** if they are both empty or if they are both nonempty, their left trees are similar, and their right trees are similar. Write a C/C++ program to determine whether or not

one binary tree is similar to some subtree of another. Analyze time complexity for all the algorithms you have written.

5. Two binary trees are ***mirror similar*** if they are both empty or if they are both nonempty and the left subtree of each is mirror similar to the right subtree of the other. Write a C/C++ program to determine whether or not one binary tree is mirror similar to some subtree of another. Analyze time complexity for all the algorithms you have written.

6. Write a C/C++ program that accepts a pointer to a binary search tree and deletes the smallest element from the tree. Analyze time complexity for all the algorithms you have written.

7. Show how to implement an ascending priority queue as a binary search tree. Present a C/C++ program for the operations *pqinsert* and *pqmindelete* on a binary search tree. Analyze time complexity for all the algorithms you have written.

8. Write a C/C++ program that accepts binary search tree representing an expression and returns the infix version of the expression that contains only those parentheses that are necessary. Analyze time complexity for all the algorithms you have written.

9. Write a C/C++ function that accepts a pointer to a node and returns *TRUE* if that node is root of a valid binary tree and *FALSE* otherwise. Analyze time complexity for all the algorithms you have written.

10. Write a C/C++ function that accepts a pointer to a binary tree and a pointer to a node of the tree and returns the level of the node in the tree. Analyze time complexity for all the algorithms you have written.

11. Write a C/C++ function that accepts a pointer to a binary tree and returns a pointer to a new binary tree that is the mirror image of the first. Analyze time complexity for all the algorithms you have written.

12. Write a C/C++ program to perform the following experiment: Generate 100 random numbers. As each number is generated insert into an initially empty binary search tree. When all 100 numbers have been inserted, print the level of the leaf with the largest level and the level of the leaf with the smallest level. Repeat this process 50 times. Print out a table with a count of how many of the 50 runs resulted in a difference between the maximum and minimum leaf level of 0, 1, 2, 3 and so on. Analyze time complexity for all the algorithms you have written.

13. Write a C/C++ routine to Travers a binary tree in *preorder* and *postorder* using stacks. Analyze time complexity for all the algorithms you have written.

14. Implement using C/C++ *inorder* traversal, *maketree, setleft* and straight for right in-threaded binary trees under the sequential representation. Analyze time complexity for all the algorithms you have written.

15. Write C/C++ functions to create a binary tree given: the *preorder* and *inorder* traversals of that tree. Analyze time complexity for all the algorithms you have written.

16. Write C/C++ functions to create a binary tree given: the *preorder* and *postorder* traversals of that tree. Analyze time complexity for all the algorithms you have written.

17. Construct an ordered Tree in which a node is declared as a dynamic variable. Print Preorder, Inorder and Postorder Traversal of the Tree. Analyze time complexity for all the algorithms you have written.

18. Construct an ordered Tree for the Expressions. Your tree should be able to represent the expression which includes the OPERATORS (a+b, a-b, a*b, a/b, a%b, sin(a), log(a), pow(a,b), abs(a), f(a,b,c)).
    Now create the *evaltree* function to evaluate the Expression Tree. Analyze time complexity for all the algorithms you have written.

19. Construct an ordered Tree for the Expressions. Your tree should be able to represent the expression which includes the OPERATORS (a+b, a-b, a*b, a/b, a%b, sin(a), log(a), pow(a,b), abs(a), f(a,b,c)).
    Print Preorder, Inorder and Postorder Traversal of the Tree. Analyze time complexity for all the algorithms you have written.

20. Construct an ordered Tree and perform the following two operations.
    a. *Setsons* Operation
       Inputs are: *the target node* and *list of sons*.
       In *the target node* you have to specify the node location/number/pointer in the tree under whom you want to set the *sons.*
    b. *addson* operation.