**PSG COLLEGE OF TECHNOLOGY, COIMBATORE-04**

**DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES**

**II Semester MSc Software Systems**

**18XW26 Data Structures Lab**

**Data Structures Lab – Binary Search Tree**

**Problem sheet - 10**

1. Write a C/C++ program to construct a binary search tree BST for the following elements.

10, 6, 8, 200, 89, 65, 43, 1, 56, 345, 2, 54

Perform the following operations on the constructed BST

1. Display the elements of the tree using inorder, preorder and postorder traversals
2. Find and print the highest and lowest element of the tree
3. Display the elements which are divisible by 5
4. Display odd elements of the tree
5. Display the elements in decreasing order
6. Display the elements in increasing order
7. Find and display prime numbers of the tree
8. Search an element on the constructed BST
9. Given two binary trees, return true if they are structurally identical -- they are made of nodes with the same values arranged in the same way.

int sameTree(struct node\* a, struct node\* b) { }

1. Change a tree so that the roles of the left and right pointers are swapped at every node.

So the tree...

4

/ \

2 5

/ \

1 3

is changed to...

4

/ \

5 2

/ \

3 1

The solution is short, but very recursive. As it happens, this can be accomplished without changing the root node pointer, so the return-the-new-root construct is not necessary. Alternately, if you do not want to change the tree nodes, you may construct and return a new mirror tree based on the original tree.

void mirror(struct node\* node) { }

1. Compute maximum depth of a binary tree, which is the number of nodes along the longest path from the root node down to the farthest leaf node. The maxDepth of the empty tree is 0, the maxDepth of the tree shown in question 3 is 4.
2. We'll define a "root-to-leaf path" to be a sequence of nodes in a tree starting with the root node and proceeding downward to a leaf (a node with no children). We'll say that an empty tree contains no root-to-leaf paths. So for example, the following tree has exactly four root-to-leaf paths:

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

Root-to-leaf paths:

path 1: 5 4 11 7

path 2: 5 4 11 2

path 3: 5 8 13

path 4: 5 8 4 1

For this problem, we will be concerned with the sum of the values of such a path -- for example, the sum of the values on the 5-4-11-7 path is 5 + 4 + 11 + 7 = 27.

Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

int hasPathSum(struct node\* node, int sum) { }

1. For each node in a binary search tree, create a new duplicate node, and insert the duplicate as the left child of the original node. The resulting tree should still be a binary search tree.

So the tree...

2

/ \

1 3

is changed to...

2

/ \

2 3

/ /

1 3

/

1

As with the previous problem, this can be accomplished without changing the root node pointer.

void doubleTree(struct node\* node) { }

1. Given a plain binary tree, examine the tree to determine if it meets the requirement to be a binary search tree. To be a binary search tree, for every node, all of the nodes in its left tree must be <= the node, and all of the nodes in its right subtree must be > the node. Consider the following examples...
2. 5 -> TRUE

/ \

2 7

1. 5 -> FALSE,

/ \

6 7

1. Given a binary tree, print out all of its root-to-leaf paths as defined above. This problem is a little harder than it looks, since the "path so far" needs to be communicated between the recursive calls.

Hint: In C, C++, and Java, probably the best solution is to create a recursive helper function printPathsRecur(node, int path[], int pathLen), where the path array communicates the sequence of nodes that led up to the current call. Alternately, the problem may be solved bottom-up, with each node returning its list of paths. This strategy works quite nicely in Lisp, since it can exploit the built in list and mapping primitives.

Given a binary tree, print out all of its root-to-leaf paths, one per line.

void printPaths(struct node\* node) { }

***Sample Tree code:***

struct Tree

{

int d;

struct Tree \*r,\*l;

};

struct Tree\* min(struct Tree\*);

void insert(struct Tree\*\* t, struct Tree \*x)

{

if(\*t==NULL)

{

\*t=x; return;

}

if(x->d<(\*t)->d)

insert(&(\*t)->l, x);

else

insert(&(\*t)->r,x);

}

void printin(struct Tree \*t)

{

//printf("\n The inorder traversal is:");

if(t!=NULL)

{

printin(t->l);

printf(" %d",t->d);

printin(t->r);

}

}

int main()

{

struct Tree \*root,\*x; int a;

root=NULL;

for(int i=1;i<9;i++)

{

x=(struct Tree \*)malloc(sizeof(struct Tree));

x->l=NULL;

x->r=NULL;

printf("Enter a number:");

scanf("%d",&(x->d));

insert(&root,x);

}

printin(root);

}