Binary Trees: Practice Problems

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I
Warmup Problem 1:

Searching for a node in a BST

```c
int find (struct tree_node *current_ptr, int val) {  
    // Check if there are nodes in the tree.
    if (current_ptr != NULL) {  
        // Found the value at the root.
        if (current_ptr->data == val)
            return 1;
        // Search to the left.
        if (val < current_ptr->data)
            return find(current_ptr->left, val);
        // Or...search to the right.
        else
            return find(current_ptr->right, val);
    }

    return 0;
}
```
Warmup Problem 2:

- Searching for a node in an arbitrary tree
  - Not a BST
  - Doesn’t have the ordering property

```c
int Find(struct tree_node *current_ptr, int val) {
    if (current_ptr != NULL) {
        if (current_ptr->data == val)
            return 1;
        return (Find(current_ptr->left, val) ||
                Find(current_ptr->right, val))
    } else
        return 0;
}
```
Warmup Problem 3:
- Summing the values of nodes in a tree

```c
int add(struct tree_node *current_ptr) {
    if (current_ptr != NULL)
        return current_ptr->data +
                add(current_ptr->left) +
                add(current_ptr->right);
    else
        return 0;
}
```
Binary Trees: Practice Problems

Count Nodes:
- Write a function that counts (and returns) the number of nodes in a binary tree

```c
int count(struct tree_node *root) {
    if (current_ptr != NULL)
        return 1 + count(root->left)+ add(root->right);
    else
        return 0;
}
```

Details:
- If the “root” is not NULL, then the root increases our count
  - Shown by the return of 1
- We then call `count` on the left and right subtrees of root
Count Leaf Nodes:

Write a function that counts (and returns) the number of leaf nodes in a binary tree

```c
int numLeaves(struct tree_node *p) {
    if (p != NULL) {
        if (p->left == NULL && p->right == NULL)
            return 1;
        else
            return numLeaves(p->left) + numLeaves(p->right);
    } else
        return 0;
}
```
Print Even Nodes:

Write a function that prints out all even nodes in a binary search tree

```c
int printEven(struct tree_node *current_ptr) {
    if (current_ptr != NULL) {
        if (current_ptr->data % 2 == 0)
            printf("%d ", current_ptr->data);
        printEven(current_ptr->left);
        printEven(current_ptr->right);
    }
}
```

This is basically just a traversal

Except we added a condition (IF) statement before the print statement
Print Odd Nodes (in ascending order):

Write a function that prints out all **odd** nodes, in a binary search tree, in ascending order.

```c
int printOddAsc(struct tree_node *current_ptr) {
    if (current_ptr != NULL) {
        printOddAsc (current_ptr->left);
        if (current_ptr->data % 2 == 1)
            printf("%d ", current_ptr->data);
        printOddAsc (current_ptr->right);
    }
}
```

The question requested **ascending** order

- This requires an **inorder** traversal
- So we simply changed the order of the statements
Brief Interlude: FAIL Picture
Binary Trees: Practice Problems

Compute Height:

Write a recursive function to compute the height of a tree

- Defined as the length of the longest path from the root to a leaf node
- For the purposes of this problem,
  - a tree with only one node has height 1
  - and an empty tree has height 0
- Your function should make use of the following struct:

```c
struct tree_node {
    int data;
    struct tree_node* left;
    struct tree_node* right;
};
```
**Compute Height:**

```c
int height(struct tree_node* root) {
    int leftHeight, rightHeight;
    if(root == NULL) {
        return 0;
    }
    leftHeight = height(root->left);
    rightHeight = height(root->right);
    if(leftHeight > rightHeight) {
        return leftHeight + 1;
    }
    return rightHeight + 1;
}
```
Find Largest:

Write a recursive function that returns a pointer to the node containing the largest element in a BST

This one should be easy:
This is a BST, meaning it has the ordering property
So where is the largest node located
  - either the root or the greatest node in the right subtree

Your function should make use of the following struct:

```c
struct tree_node {
    int data;
    struct tree_node* left;
    struct tree_node* right;
};
```
### Find Largest:

```c
struct node* largest(struct tree_node *B) {

    // if B is NULL, there is no node
    if (B == NULL)
        return NULL;
    // If B's right is NULL, that means B is the largest
    else if (B->right == NULL)
        return B;
    // SO if B's right was NOT equal to NULL,
    // There is a right subtree of B.
    // Which means that the largest value is in this
    // subtree. So recursively call B's right.
    else
        return largest(B->right);
}
```
Number of Single Children:

- In a binary tree, each node can have zero, one, or two children
- Write a recursive function that returns the number of nodes with a single child
- Your function should make use of the following struct:

```c
struct tree_node {
    int data;
    struct tree_node* left;
    struct tree_node* right;
};
```
Number of Single Children:

```c
int one (struct tree_node *p) {
    if (p != NULL) {
        if (p->left == NULL)
            if (p->right != NULL)
                return 1 + one(p->right);
        else if (p->right == NULL)
            if (p->left != NULL)
                return 1 + one(p->left);
        else
            return one(p->left) + one(p->right);
    }
}
```
WASN’T THAT SPICY!
Daily Demotivator

FUTILITY
You’ll Always Miss 100% of the Shots You Don’t Take, and, Statistically Speaking, 99% of the Shots You Do.
Binary Trees: Practice Problems

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I