CS 106B Section 7 (Week 8) Solutions

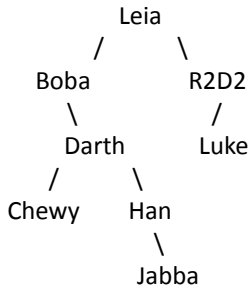
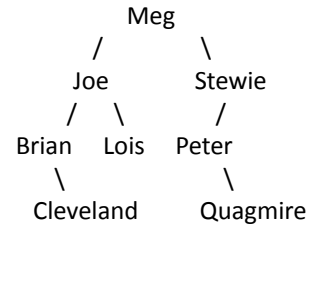
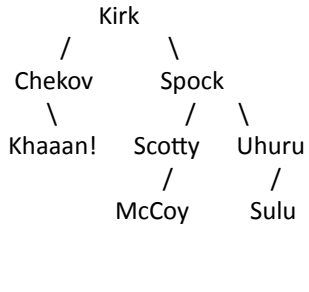
---

1. Traversals

a)	b)	c)
pre-order: 3 5 1 2 4 6 in-order: 1 5 3 4 2 6 post-order: 1 5 4 6 2 3	pre-order: 19 47 23 -2 55 63 94 28 in-order: 23 47 55 -2 19 63 94 28 post-order: 23 55 -2 47 28 94 63 19	pre-order: 2 1 7 4 3 5 6 9 8 In-order: 2 3 4 5 7 1 6 8 9 post-order: 3 5 4 7 8 9 6 1 2

---

2. BST Insertion

a)	b)	c)
		

---

3. height

```
int BinaryTree::height() {
    return height(root);
}

int BinaryTree::height(TreeNode* node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + max(height(node->left), height(node->right));
    }
}
```

---

4. countLeftNodes

```
int BinaryTree::countLeftNodes()  {
    return countLeftNodes(root);
}

int BinaryTree::countLeftNodes(TreeNode* node) {
    if (node == NULL) {
        return 0;
    } else if (node->left == NULL) {
        return countLeftNodes(node->right);
    } else {
        return 1 + countLeftNodes(node->left) + countLeftNodes(node->right);
    }
}
```

CS 106B Section 7 (Week 8) Solutions

---

5. isBalanced

```
bool BinaryTree::isBalanced() {
    return isBalanced(root);
}

bool BinaryTree::isBalanced(TreeNode* node) {
    if (node == NULL) {
        return true;
    } else if (!isBalanced(node->left) || !isBalanced(node->right)) {
        return false;
    } else {
        int leftHeight  = height(node->left);
        int rightHeight = height(node->right);
        return abs(leftHeight - rightHeight) <= 1;
    }
}
```

---

6. removeLeaves

```
void BinaryTree::removeLeaves() {
    removeLeaves(root);
}

void BinaryTree::removeLeaves(TreeNode*& node) {
    if (node != NULL) {
        if (node->left == NULL && node->right == NULL) {
            delete node;
            node = NULL;
        } else {
            removeLeaves(node->left);
            removeLeaves(node->right);
        }
    }
}
```

---

7. completeToLevel

```
void BinaryTree::completeToLevel(int k) {
    if (k < 1) {
        throw k;
    }
    completeToLevel(root, k, 1);
}

void BinaryTree::completeToLevel(TreeNode*& node, int k, int level) {
    if (level <= k) {
        if (node == NULL) {
            node = new TreeNode(-1);
        }
        completeToLevel(node->left,  k, level + 1);
        completeToLevel(node->right, k, level + 1);
    }
}
```

CS 106B Section 7 (Week 8) Solutions

---

8. tighten

```
void BinaryTree::tighten() {
    tighten(root);
}

void BinaryTree::tighten(TreeNode*& node) {
    if (node != NULL) {
        tighten(node->left);
        tighten(node->right);
        if (node->left == NULL && node->right != NULL) {
            TreeNode* trash = node;
            node = node->right;
            delete trash;
        } else if (node->left != NULL && node->right == NULL) {
            TreeNode* trash = node;
            node = node->left;
            delete trash;
        }
    }
}
```

---

9. limitPathSum

```
void BinaryTree::limitPathSum(int max) {
    limitPathSum(root, max, 0);
}

void BinaryTree::limitPathSum(TreeNode*& node, int max, int sum) {
    if (node != NULL) {
        sum += node->data;
        if (sum > max) {
            deleteTree(node);
            node = NULL;
        } else {
            limitPathSum(node->left,  max, sum);
            limitPathSum(node->right, max, sum);
        }
    }
}
```