CSCI2100C Data Structures

Tutorial 10: *AVL Tree and Traversal*

LI Muzhi
mzli@cse.cuhk.edu.hk

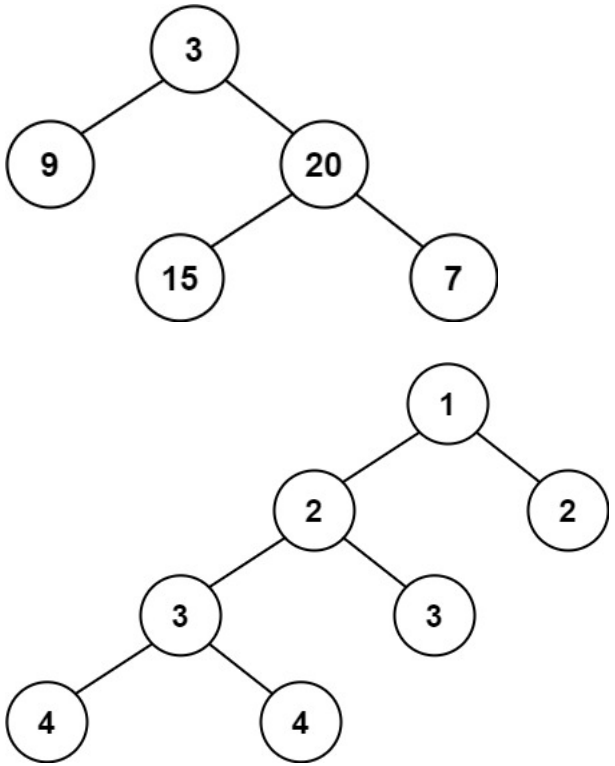
Slides made by Mr. LI Yanwei, TA of CSCI2100B (2020-21 Spring)

Exercise 1

Balanced Binary Tree

Question: Given a binary tree ***t***, determine if it is height-balanced. For this problem, a height-balanced binary tree is defined as:

a binary tree in which the left and right subtrees of *every* node differ in height by no more than 1.



Example 1:

Input: t = [3,9,20,null,null,15,7]
Output: true

*An example to illustrate values in the tree, **NOT** the structure of the tree.*

Example 2:

Input: t = [1,2,2,3,3,null,null,4,4]
Output: false

```
typedef struct BinaryTreeCDT
*BinaryTreeADT;

typedef  struct TreeNodeCDT
*TreeNodeADT;

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
                    BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);


bool isBalanced(BinaryTreeADT t) {
    //  Please type your code here

}
```

Adapted from LeetCode Question #110: [Balanced Binary Tree]

Exercise 1

Balanced Binary Tree

Definition: a binary tree in which the left and right subtrees of *every* node differ in height by ***no more than 1***.

Base case:

If tree root is NULL, it is viewed as a binary tree.

Condition: t = [null]

Result: true

Common case:

According to the definition, if the difference between the depth of left-subtree and right-subtree should ***no more than 1****.*

Condition: |maxDepth(t->lst) - maxDepth(t->rst) |<=1

Result: true

```
typedef struct BinaryTreeCDT
*BinaryTreeADT;

typedef  struct TreeNodeCDT
*TreeNodeADT;
```

```
BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
                    BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);
```

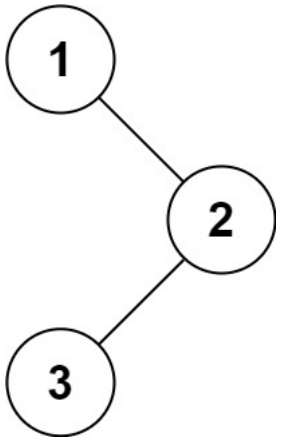
```
int maxDepth(BinaryTreeADT t) {
    if (TreeIsEmpty(t)) return 0;
    else
    {
        int lDepth = maxDepth(LeftSubtree(t));
        int rDepth = maxDepth(RightSubtree(t));
        if (lDepth> rDepth) return lDepth+1;
        else return rDepth+1;
     }
}

bool isBalanced(BinaryTreeADT t) {
    //  Please type your code here
    if (TreeIsEmpty(t)) return true;
    int lDepth = maxDepth(LeftSubtree(t));
    int rDepth = maxDepth(RightSubtree(t));
    if ((lDepth- rDepth>1) || (lDepth- rDepth<-1))
     {return false;}
    return (isBalanced(LeftSubtree(t)) &&
            isBalanced(RightSubtree(t)));
}
```

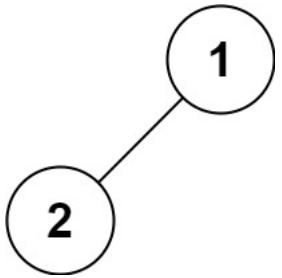
Adapted from LeetCode Question #110: Balanced Binary Tree

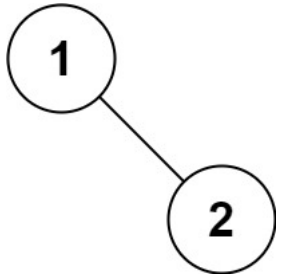
Exercise 2

Binary Tree Inorder Traversal

Question: Given the a binary tree ***t***, return *the inorder traversal of its nodes' values*.







Example 1:

Input: t = [1,null,2,3]
Output: [1,3,2]

Example 2:

Input: t = [1,2]
Output: [2,1]

Example 3:

Input: t = [1,null,2]
Output: [1,2]

```
typedef struct BinaryTreeCDT
*BinaryTreeADT;


typedef  struct TreeNodeCDT
*TreeNodeADT;

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
                        BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);


int* inorderTraversal(BinaryTreeADT t,
int* returnSize) {
    // The returned array must be malloced
    //  Please type your code here

}
```

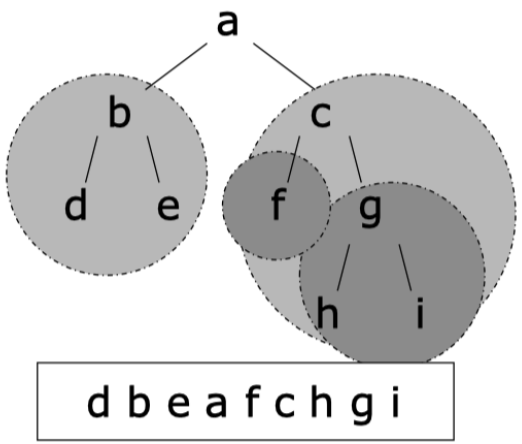
Adapted from LeetCode Question #94: [Binary Tree Inorder Traversal]

Exercise 2

Binary Tree Inorder Traversal

Definition: In an in-order traversal of a tree, we 'visit' the tree nodes in this order as:

- ***First:*** Visiting nodes in the ***left*** subtrees using inorder traversal;
- ***Then:*** Visiting the ***root***;
- ***Finally:*** Visiting nodes in the ***right*** subtrees using inorder traversal.




```
typedef struct BinaryTreeCDT *BinaryTreeADT;
typedef  struct TreeNodeCDT  *TreeNodeADT;

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
                        BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);
```

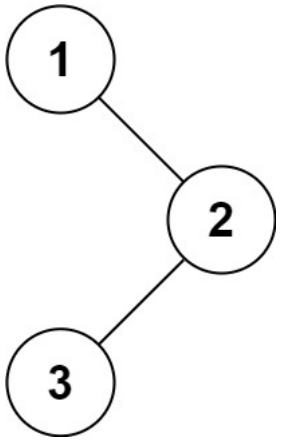
```
void create(BinaryTreeADT t, int* res,
int* index){
    if (TreeIsEmpty(t)) return NULL;
    create(LeftSubtree(t), res, index);
    res[(*index)++] = GetNodeData(Root(t));
    create(RightSubtree(t), res, index);
}
```

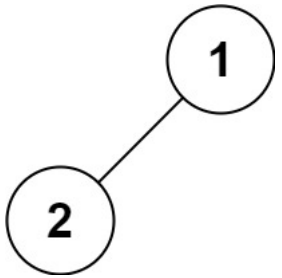
```
int* inorderTraversal(BinaryTreeADT t,
int* returnSize) {
    // The returned array must be malloced
    //  Please type your code here
    * returnSize = 0;
    int* res = malloc(sizeof(int) * 100);
    create(t, res, returnSize);
    return res;
}
```

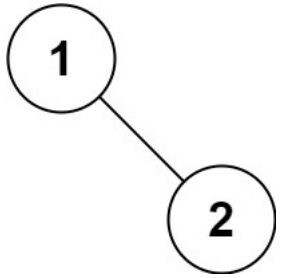
Exercise 3

Binary Tree Preorder Traversal

Question: Given the a binary tree ***t***, return *the preorder traversal of its nodes' values*.







Example 1:

Input: t = [1,null,2,3]
Output: [1,2,3]

Example 2:

Input: t = [1,2]
Output: [1,2]

Example 3:

Input: t = [1,null,2]
Output: [1,2]

```
typedef struct BinaryTreeCDT
*BinaryTreeADT;


typedef  struct TreeNodeCDT
*TreeNodeADT;

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
                        BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);


int* preorderTraversal(BinaryTreeADT t,
int* returnSize) {
    // The returned array must be malloced
    //  Please type your code here

}
```

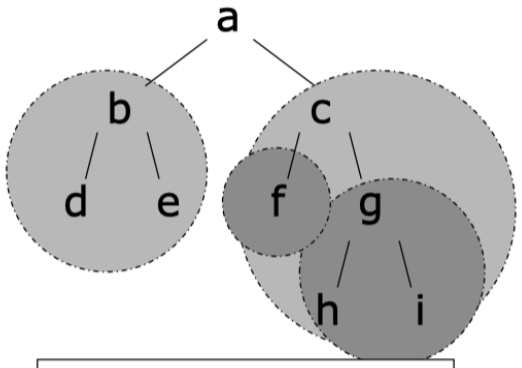
Adapted from LeetCode Question #144: [Binary Tree Preorder Traversal]

Exercise 3

Binary Tree Preorder Traversal

Definition: In an pre-order traversal of a tree, we 'visit' the tree nodes in this order as:

- ***First:*** Visiting the ***root***;
- ***Then:*** Visiting nodes in the ***left*** subtrees using preorder traversal;
- ***Finally:*** Visiting nodes in the ***right*** subtrees using preorder traversal.



Inorder d b e a f c h g i

preorder a b d e c f g h i

```
typedef struct BinaryTreeCDT
*BinaryTreeADT;

typedef  struct TreeNodeCDT
*TreeNodeADT;
```

```
typedef struct BinaryTreeCDT {
    TreeNodeADT rt;
    BinaryTreeADT lst;
    BinaryTreeADT rst;};

struct TreeNodeCDT{int val;};
```

```
void create(BinaryTreeADT t, int* res,
int* index){
    if (TreeIsEmpty(t)) return NULL;
    res[(*index)++] = GetNodeData(Root(t));
    create(LeftSubtree(t), res, index);
    create(RightSubtree(t), res, index);
}
```

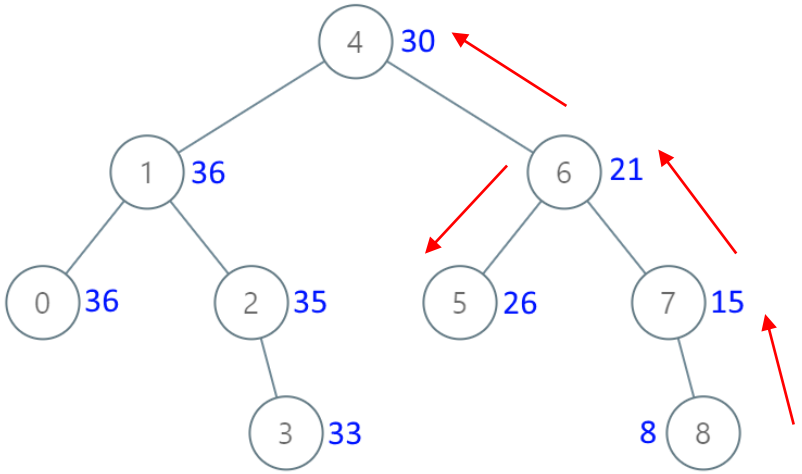
```
int* preorderTraversal(BinaryTreeADT t,
int* returnSize) {
    // The returned array must be malloced
    //  Please type your code here
    * returnSize = 0;
    int* res = malloc(sizeof(int) * 100);
    create(t, res, returnSize);
    return res;
}
```

Adapted from LeetCode Question #144: Binary Tree Preorder Traversal

Exercise 4

Convert BST to Greater Tree

Question: Given the root ***t*** of a Binary Search Tree (BST), convert it to a *Greater Tree* such that every key of the original BST is changed to the *original key plus sum of all keys* ***greater than*** the original key in BST.



An example to illustrate values in the tree, ***NOT*** *the structure of the tree.*

Example:

Input: t = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]

Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]

Adapted from LeetCode Question #538: Covert BST to Greater Tree

```
typedef struct BinaryTreeCDT
*BinaryTreeADT;

typedef  struct TreeNodeCDT
*TreeNodeADT;

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
                    BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);

TreeADT convertBST(TreeADT t){
   //  Please type your code here

}
```

Exercise 4

Convert BST to Greater Tree

Remind: A balanced binary search tree satisfies:

- The ***left subtree*** of a node contains only nodes with keys ***less than*** the node's key.
- The ***right subtree*** of a node contains only nodes with keys ***greater than*** the node's key.
- Both left and right subtrees must be binary search trees.

Base case:

If tree root is NULL, we just return the NULL.
Condition: t = [null]
Result: null

Common case:

We first visit all the nodes in the right subtree and get the accumulated value, then visit nodes in the left subtree.
Step: visit t->rst; accumulate value; visit t->lst;
Result: the converted tree

```
typedef struct TreeCDT *TreeADT;
```

```
typedef  struct TreeNodeCDT *TreeNodeADT;
```

Adapted from LeetCode Question #538: Covert BST to Greater Tree

```
typedef struct TreeCDT {
    TreeNodeADT rt;
    TreeADT lst;
    TreeADT rst;};

struct TreeNodeCDT{int val;};

TreeADT visitBST(TreeADT t, int *acc_val){
    if (TreeIsEmpty(t)) return t;
    TreeADT rst = visitBST(RightSubtree(t),  acc_val);
    int root_val = GetNodeData(Root(t)) + *acc_val;
    TreeNodeADT root = NewTreeNode(root_val);
    t = NonemptyBinaryTree(root, LeftSubtree(t), rst);
    *acc_val = GetNodeData(Root(t));
   TreeADT lst = visitBST(LeftSubtree(t), acc_val);
    t = NonemptyBinaryTree(root, lst, RightSubtree(t));
    return t;
}

TreeADT convertBST(TreeADT t){
    //  Please type your code here
    int acc_val = 0;
    t = visitBST(t, &acc_val);
    return t;
}
```

Thanks!

Contact me for more questions.

Muzhi Li
mzli@cse.cuhk.edu.hk