CPSC-50300 Algorithms and Data Structures

Spring 2022

Final Exam

**Question-1 (10 points)** In a binary tree class, implement a Python method equal, that takes in another binary tree (other) as parameter and returns true if both binary trees are equal, otherwise the function returns False. What is the time complexity of your function?

class BinaryTree:

    class \_Node:

        def \_\_init\_\_(self, element, left = None, right = None):

            self.\_left = left

            self.\_right = right

            self.\_element: int = element

    def \_\_init\_\_(self):

        self.\_root = None

        self.\_size = 0

    def add\_root(self, e):

        if self.\_root:

            raise Exception()

        node = self.\_Node(e)

        self.\_root = node

        return node

    def add\_left(self, e, p):

        node = self.\_Node(element=e)

        p.\_left = node

        return node

    def add\_right(self, e, p):

        node = self.\_Node(element=e)

        p.\_right = node

        return node

    def equal(self,root1, root2) :

        if (root1 == None and root2 == None) :

            return True

        elif (root1 != None and root2 == None) :

            return False

        elif (root1 == None and root2 != None) :

            return False

        else:

            if (root1.\_element == root2.\_element and

                self.equal(root1.\_left, root2.\_left)

                and self.equal(root1.\_right, root2.\_right)) :

                return True

            else:

                return False

btree = BinaryTree()

root = btree.add\_root(4)

node = btree.add\_left(2, root)

btree.add\_left(1, node)

btree.add\_right(3, node)

btree1 = BinaryTree()

root2 =  btree1.add\_root(3)

node = btree1.add\_left(2, root2)

btree1.add\_left(1, node)

btree1.add\_right(3, node)

print("Are trees equal? ", btree.equal(root, root2))

**Time Complexity:** Time complexity of equal function is **O(n)**.

**Question-2: (10 points)** consider an array-based binary tree implementation, write a method find\_ansestors, that takes in an index i and returns all ancestors of node located at index i. What is time complexity of your function?

class ArrayBinaryTree:

    def \_\_init\_\_(self):

        self.\_heap = []

    def find\_ancestors(self, root, i):

        if root == None:

            return False

        if root.data == i:

            return True

        if (self.find\_ancestors(root.left, i) or self.find\_ancestors(root.right, i)):

            print(root.data),

            return True

        return False

**Time Complexity:** Time complexity of find\_ancestors function is **O(n)** because of recursion.

**Question-3 (5 points)** Show the binary tree that corresponds to the following array-based implementation of a binary tree. Does this represent a heap and/or binary search tree (i.e., exhibits heap property and/or binary search property)?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 8 | 10 | 7 | 6 | 1 | 5 | 3 |

**Representation of this sequence of elements in Binary Tree:**

**15**

/ \

**8 10**

/ \ / \

**7 6 1 5**

/

**3**

According to visualization of tree, this is not binary search tree rather it is max heap because it its parent node or root is greater than its leaf nodes which satisfies the condition of max heap.

**Question-4 (10 points)** in binary search tree, write a function that takes in a root, p, and checks whether the tree rooted in p is a binary search tree or not. What is time complexity of your function?

import sys

def is\_bst(self, p):

    if self.check\_bst(p, -sys.maxsize-1, sys.maxsize):

        print('Tree is Binary Search tree.')

    else:

        print('Tree is not Binary Search tree. ')

def check\_bst(self, p, minimum, maximum):

    if p is None:

        return True

    if p.\_element < minimum or p.\_element > maximum:

        return False

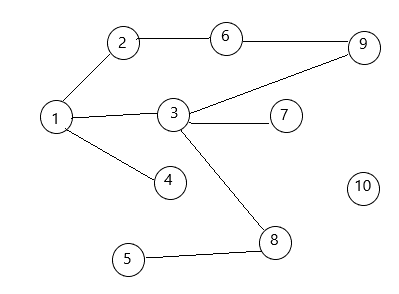
    return check\_bst(p.\_left, minimum, p.\_element) and check\_bst(p.\_right, p.\_element, maximum)

is\_bst(root)

**Time Complexity:** Time complexity of is\_bst function is **O(n)** because of recursion.

**Question-5 (10 points)** Study the below graph, and answer the following questions:

1. What is the corresponding adjacency matrix
2. Show order of nodes if a Depth-First-Search is invoked on node 8
3. Show order of visiting nodes, if a Breadth-First-Search is invoked at node 6



**Adjacency matrix:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Node-1** | **Node-2** | **Node-3** | **Node-4** | **Node-5** | **Node-6** | **Node-7** | **Node-8** | **Node-9** | **Node-10** |
| **Node-1** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Node-2** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Node-3** | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| **Node-4** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Node-5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **Node-6** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **Node-7** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Node-8** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **Node-9** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Node-10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Depth-First-Search traversal – start at node 5**

**8**

/ \

**3 5**

/ \

**7 9**

/

**6**

/

**2**

/

**1**

/

**4**

DFS = 8, 3, 7, 9, 6, 2, 1, 4, 5

**Breadth-First-Search traversal – start at node 5**

**6**

/ \

**2 9**

/ /

**1 3**

/ / \

**4**  **7 8**

\

**5**

DFS = 6, 2, 9, 1, 3, 4, 7, 8, 5