CPSC-34000 Algorithms and Data Structures

Fall 2021

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** equal(self, other: BinaryTree):

**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, i: int):

**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 |

**Question-4 (5 points)** Given the sequence of numbers 1 through 15, what would be the order of insertion of these numbers into a binary search tree that would result in a binary search tree that exhibit worst and best search performance.

Order that results in worst-search performance

Resulting BST

Order that results in best-search performance

Resulting BST

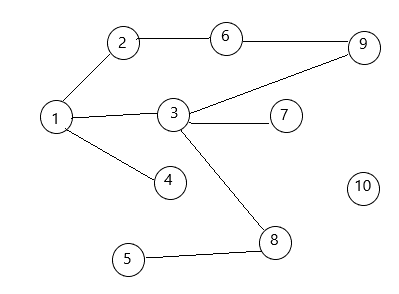
**Question-5 (10 points)** in binary search tree, write a function that takes in a root, p, and checks whether the tree rooted in p is a binary search tree or not. What is time complexity of your function?

**def** is\_bst(self, p: Node):

**Question-6 (5 points)** Describe the difference between a tree and hash-based implementations of Map ADT, discuss their advantages, disadvantages and application scenarios where each should be used.

**Question-7 (15 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 1
3. Show order of visiting nodes, if a Breadth-First-Search is invoked at node 1



Adjacency matrix

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

Depth-First-Search traversal – start at node 1

Breadth-First-Search traversal – start at node 1