

# SIT320 — Advanced Algorithms

## Pass Task 4: Trees

### About this Task

At the completion of the module (**Module 4: Trees**), you are required to fill a lesson review by doing following activities.

Your tutor will then review your submission and will give you feedback. If your submission is incomplete they will ask you to include missing parts. They can also ask follow-up questions, either to clarify something, or to double check your understanding of certain concepts.

### Task List

- **(0)** Provide a short overview of what you learned in the module. This should be based on your learning summary from lecture (seminar), module content on cloud Deakin, your interaction with Unit Chair/Tutors/Peers, your research in the library or the internet and/or your interaction with chatGPT (make sure to provide the prompts you use).

**For doing following activities, we have provided you some Python code. You can download it from the “resource section” of the module. This code is only provided for those students who are new to Python and for tutors to better support them. You are welcome to have your own implementations of the algorithms built from scratch.**

- **(1)** Modify the BST so that it can keep track of its balance. For the purpose of this question, a balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differs by more than one.
  - You are expected to create a new field in each node which stores the balance of the node.
  - Note, when a node is inserted and deleted, the balance must be updated.
  - Write a function to test the balance of a tree, which returns True or False depending if the tree is balanced or not.
- **(2)** Write an algorithm (and code) to find the first common ancestor of two nodes in a binary search tree. Your function will take as input arguments: two nodes and a Binary Search Trees (BST). Your algorithm should first search the nodes and then determine what is the first common ancestor node. The common ancestor node should be returned.
- **(3)** In the seminar, we discussed how various trees such as AVL or Red Black trees make extensive use of rotation operation. In fact, if you can perform rotations easily, you can code insertion and deletion in AVL and RB trees. Write an algorithm (and code) to perform following form of rotations on any specified node of a BST? You can assume that nodes P, Q and C

## SIT320 — Advanced Algorithms

does exist in your tree. Your function should take the tree as well as a node at which to perform rotation, as input arguments, and then return a tree in which rotation is performed:

- a left rotation,
- a right rotation,
- a left-right rotation,
- a right-left rotation.

**(\*) Make sure you provide test cases to test the working of your code.**

**(\*\*) You are welcome to write your code at the end of ipynb code provided to you.**