Lab 3 - Description

(Binary Search Tree)

Lab Overview:

For this lab, we will be taking what we've learned in class and using it to implement a Binary Search Tree (BST). In CS, a binary search tree is used in a variety of application domains from routers to data compression. In this lab we will focus on building the simple BST to prepare us for a self-balancing binary tree such as a red-black tree.

Core Tasks:

1. Write the BST class.

Program Requirements:

Task 1: Write the BST class.

For task 1, you will need to write a *BinarySearchTree* class. The class is detailed as follows:

- $\_init\_(\text{self}) \rightarrow \boldsymbol{None}$
  - **Complexity**: O(1)
  - **Description:** This is the constructor for the class. You will need to store the following here:
    - A variable *_root*
    - Any other instance variables you want.
    - **Note:** Our implementation of the BST will be a dynamic data structure (i.e. it grows as the user inserts tickets).
- $insert(\text{self}, ticket) \rightarrow \boldsymbol{Boolean}$
  - **Complexity:** O(lg(n)) ~ O(n)
  - **Valid Input:** An object of type: MealTicket. Return False on invalid input.
  - The insert method will insert a MealTicket into the tree. This should also preserve the ordering requirement that all nodes in the right side of a subtree are greater than the value in the root of that subtree, and all elements in the left side are lesser than the value in the root of the subtree. As done in the last lab, ordering will be done on the basis of a meal ticket's ID. It will then return True or False depending on if the insert was successful.
  - **Note:** To simplify things, we will not test your binary search tree with duplicate elements.
- $delete(\text{self}, ticketID) \rightarrow \boldsymbol{Boolean}$
  - **Complexity:** O(lg(n)) ~ O(n)
  - **Valid Input:** A positive integer in the range (0, ∞]. Return False on invalid input.
  - **Description:** The delete method will remove a node from the tree and return True, else it will return False. When deleting a value, delete the node whose ticketID matches the integer ticketID from input parameter. If said node has no children, simply remove it by setting its parent to point to null instead of it. If said node has

one child, delete it by making its parent point to its child. If said node has two children, then replace it with its successor. The successor of a node is the left-most node in the node's right subtree. If the value specified by delete does not exist in the tree, then the structure is left unchanged.

- $find(\text{self}, ticketID) \rightarrow \boldsymbol{MealTicket|Boolean}$
    - **Complexity:** O(lg(n)) ~ O(n)
    - **Valid Input:** A positive integer in the range (0, ∞]. Return False on invalid input.
    - **Description:** Find takes an int ticketID and returns the ticket in the tree whose ID matches that value. If no such ticket exists in the tree, return False.

Note 1: The methods listed above are the only methods that will be tested. However, you may add additional methods as you please. We have provided a skeleton code to get you started be sure to read and understand the auxiliary methods provided before you start.

Note 3: Pay attention to the complexity bounds mentioned in the method descriptions. Your implementation **<u>must</u>** run within these bounds.

Note 4: For this assignment you **<u>are not allowed to use a list.</u>** This assignment should be done using nodes that make up a linked list like how we did assignment 1. You are also not allowed to use any of the python built-in functions are libraries for this assignment. Everything can be done with simple python. The skeleton codes contain everything you need to implement the methods above, so you don't really need to add anything.

Program Robustness:
For this assignment, a skeleton code and the mealticket file is provided. Feel free to make your own main to test your code. However, only the BST class (and any related classes/functions) should be turned in. I will be grading with my own testing script.

Warning: I will be testing your code with a more robust main that will check all of the corner cases and uses a wider variety of tickets. Keep this in mind while developing your data structures. The description above gives you a solid idea on what needs to be done but does not describe all corner cases. Your programs **<u>must</u>** be robust (i.e. do not crash when given bad input or told to do things out of order.)

Remarks:

All programs written in this class will be done using the newest version of Python available in the lab (Python 3.7/3.8). This is because Python 3.5+ is platform independent (i.e. you can code on a PC and run it on a Mac). The computers in KLA 26 all have python on them so make sure your code runs on the computers in the lab as it will be tested on an identical system. Feel free to install python on your own computer and bring it to the lab.

If you are new to python, here are some super helpful resources to get you up to speed:

1. Cheat sheets: https://www.pythoncheatsheet.org/ ← Most recommended! Many thumbs up!
2. Python documentation: https://docs.python.org/3/

All programming assignments are to be done **individually**. Your code will be looked at with professional software for cheating. ***Warning:*** This includes using online sources. (e.g. Do not go online and copy code from stack overflow. People have tried this before. You will fail.) Be extra careful with your code. Do not ever show your work to anyone other than the TA (me) or the professor. They will most likely copy your work and your will both fail.

Submission Requirements:

In order to receive any credit for the assignment the student ***must*** do the following:

1. Name your program **"<Duck-ID>_lab3.py"**. (i.e. my duck ID is jhall10 so my submission would be named **jhall10_lab3.py**. **Note:** your duck-ID is the same as your email id and the username to log on to CIS computers ***not*** your 951… number that is your UO PID.
2. Submit your python file onto Canvas.

That's it! Make sure that you test your code on the lab computers to make sure it works.

Grading:

Your work will be graded along three primary metrics: Correctness, Completeness, and Elegance.

Correctness: (60% of total grade)

- You wrote the class methods as specified.
- Your class methods meet the complexity requirements.
- You utilize a linked list to build your BST.
- You implement the correct algorithms.
- Your classes are robust, fault tolerant and follow the specified behavior on invalid input.

Completeness (25% of total grade)

- Program contains a class named: *BinarySearchTree*
- The class contains methods as defined above.
- The method signatures were implemented as specified.

Elegance: (11% of total grade)

- See grading guide posted on canvas for information on elegance.

Late Policy:

The late policy for this class is a bit unique. Make sure to read the info below carefully:
Your homework is always due on a Tuesday at 11:59 pm. On Wednesday, I will take 10% off of the total points available. On Thursday, I will take 20% off. On Friday 30%, no homework will be accepted after Friday. I do not start grading the homework until the following Monday. Keep this in mind when submitting your assignment: It may be better for you to seek help and submit the homework a day late than to submit on time and fail.

If you encounter an unfortunate event or are working with a disability: Please email or speak to me. I am super flexible and am always on your side. I will give extensions as needed and am willing to work with you to make sure you get the most out of this course.