CS 313 - Project 3
Binary Search Tree

February 16, 2023

1 Project Overview

For this project, 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. We will focus on building the simple BST to prepare us for a self-balancing binary tree such as a red-black tree.

2 Program Requirements:

2.1 Write BinarySearchTree class.

You will need to write a BinarySearchTree class. To receive any credit, your class must follow the specifications below exactly:

- Class Name: **BinarySearchTree**
- Methods:
  - $\_\_init\_\_(< BinarySearchTree > self) \rightarrow < Nonetype > None$
    - Complexity: $O(1)$.
    - Instance variables:
      - $< Node >$ $\_root$: This variable contains the head of our linked list. Note: must be initialized as None.
      - Any other instance variables you want.
  - $insert(< BinarySearchTree > self, < Tuple > item) \rightarrow < Bool > returnValue$
    - Complexity: $O(lg(n)) \sim O(n)$.
    - Valid Input: A tuple $(id, value)$ containing the following:
      - id: An integer in the bound $(-\infty, +\infty)$.
      - $value$: Any Python object.
    - Error Handling: Return False on invalid input. Do not raise an exception for invalid input.
    - Description: The *insert* method will insert a new item into the tree. It will do so by creating a new node with the tuple then adding it to the tree. This should also preserve the ordering requirement that all nodes on the right side of a subtree are greater than the value in the root of that subtree, and all elements on 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 the first element of the tuple. 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.
  - $delete(< BinarySearchTree > self, < int > id) \rightarrow < Bool > returnValue$
    - Complexity: $O(lg(n)) \sim O(n)$

    * Valid Input: An integer in the bound $(-\infty, +\infty)$.
    * Error Handling: Raise a **TreeIsEmpty** exception if the method is called when the tree is empty.
    * Description: The *delete* method will remove a node from the tree and return True, else it will return False. When deleting a node, delete the node whose ID matches the integer *id* from the input parameter.
    * Note: Consider the following four cases for delete:
        - If the said node has no children: Delete it by setting both the nodes' parent pointer and the parents' corresponding child pointer to *None*.
        - If the said node has one child: Delete it by making its parent point to its child and setting all of its pointers to None. (There is a useful method in BSTNode for this).
        - If the said node has two children: First delete the successor. Then replace the said node's content with the successor's content. 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 and the method should return False.
- $find(< BinarySearchTree > self, < int > id) \rightarrow < tuple/bool > returnValue$
    * Complexity: $O(lg(n)) \sim O(n)$
    * Valid Input: An integer in the bound $(-\infty, +\infty)$.
    * Error Handling: Raise a **TreeIsEmpty** exception if the method is called when the tree is empty.
    * Description: This method takes an integer *id* and returns the tuple in the tree whose ID matches that value. If no such tuple exists in the tree, return False.

3 Submission Requirements:

Submit a single file **p3.py** to canvas.

4 Grading:

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

- Correctness: (60 points)
    - You wrote the class methods as specified.
    - Your class methods meet the complexity requirements.
    - You utilize a tree of nodes 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 points)
    - Program contains a class named: BinarySearchTree
    - The class contains methods as defined above.
    - The method signatures were implemented as specified.
- Elegance: (15 points)
    - See the programming guide posted on canvas for information on elegance.

5 Remarks:

- 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.
- Pay attention to the complexity bounds mentioned in the method descriptions. Your implementation must run within these bounds.
- For this assignment you are not allowed to use a list. This assignment should be done using BSTNode which makes a tree, like how we did with the linked list in assignment 1. You are also not allowed to use any of the python built-in functions or 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.
- Warning: You are not allowed to import anything into the source file with the classes. All programs must be written in basic Python 3.8+.
- You will need to make your own main for testing your code using the methodology described in the lab. Testing your own code is a very common practice in the industry.
- 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 objects. Keep this in mind while developing your data structures. You will need to use the methodology discussed in lab to figure out all corner cases and account for them. Your programs must be robust (i.e. does not crash when given bad input or told to peak/front when the stack/queue is empty etc.)
- As mentioned before, please do not use method annotations.
- 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.