lab09 : Used Car Lot - Part Two

num	ready?	description	assigned	due
lab09	true	Used Car Lot - Part Two	Sun 11/27 11:59PM	Sun 12/04 11:59PM

In this lab, you'll have the opportunity to practice:

- Modifying classes in Python
- Further implementing Binary Search Tree (BST) data structure supporting removal functionality
- Testing your functionality with pytest

Note: This lab will be dependent on your previous lab. Certain tests from the previous lab will be autograded in this week's lab. It is important that you start this lab early so you can utilize our office hours to seek assistance / ask clarifying questions during the weekdays before the deadline if needed!

Introduction

The goal for this lab is to take your existing Used Car Lot program in Lab08 that will manage cars for a second-hand car dealership, and support removing Cars from the lot. As a reminder, all Cars have a `make`, `model`, `year`, and `price`, which can be used to determine the value of cars in relation to each other. All Cars will be managed with a Binary Search Tree (BST) where the BST nodes are sorted by `make`, then `model`.

In order to remove the cars for this lab, you will define a `removeCar` method in the `CarInventory` class that will remove Cars with the same `make`/`model`/`year`/`price` from a `CarInventoryNode`'s cars list. After removing a Car and no cars exist in the `CarInventoryNode`'s cars list, you will then need to remove the node from the BST while preserving the BST property.

You will also write pytests in `testFile.py` illustrating your behavior works correctly. This lab writeup will provide some test cases for clarity, but the Gradescope autograder will run different tests shown here. It's important to thoroughly test your program with various cases!

Instructions

You will need to copy over all your files from Lab08 and modify two files:

- `CarInventory.py` - Defines a CarInventory (BST) class that is an ordered collection of a Dealership's Cars. You will be adding to your existing `CarInventory` class.
- `testFile.py` - This file will contain your pytest functions that tests the overall correctness of your class definitions.

Your starter code for this assignment will be your program from Lab08, and you'll have to add the additional specifications defined below.

You should organize your lab work in its own directory. This way all files for a lab are located in a single folder. Also, this will be easy to import various files into your code using the `import / from` technique shown in lecture.

CarInventory.py

The `CarInventory.py` file will contain the definition of a `CarInventory` class. This will keep track of the cars a dealership has, implemented as a BST. The `CarInventory` will create `CarInventoryNode` objects using `Car` objects based on their `make` and `model`. `Car` objects with the same make and model will be appended to a list based on insertion order within the `CarInventoryNode` object. For further specifications regarding existing requirements, reference the Lab08 page.

In addition to the methods created before, the following methods are required to be implemented:

- `getSuccessor(self, make, model)` - attempts to finds the `CarInventoryNode` with the `make` and `model`, and returns the CarInventoryNode with the next greatest value (using the same heirarchy of `make`, then `model`). Returns `None` if there is no `CarInventoryNode` with the specified `make` and `model`, or if the `CarInventoryNode` is the maximum and has no successor. **Note, this includes the successor of any `CarInventoryNode` in the BST if it exists, not just the successor used for BST maintenance.**
- `removeCar(self, make, model, year, price)` - attempts to find the `Car` with the specified `make`, `model`, `year`, and `price`, and removes it the `CarInventoryNode`'s cars list. If the list is empty after removing the `Car`, remove the `CarInventoryNode` from the BST entirely. Returns `True` if the `Car` was successfully removed, and `False` if the `Car` is not present in the `CarInventory`. If there are duplicate cars within a `CarInventoryNode`'s car list that matches the specifications, you will just remove the first matching `Car` object in the cars list.

A note if you have implemented `CarInventoryNode` comparators: If you have implemented `CarInventoryNode` comparators in last week's lab, in your `__eq__` comparator overload, before you check for the `make` and the `model`, you should check if the right-hand-side is `None`. If it is `None`, you should return `False`. This is because of a quirk about how Python handles comparators between overloaded comparators and `None`.

Examples

Given an example BST:

```
bst = CarInventory()

car1 = Car("Mazda", "CX-5", 2022, 25000)
car2 = Car("Tesla", "Model3", 2018, 50000)
car3 = Car("BMW", "X5", 2022, 60000)
car4 = Car("BMW", "X5", 2020, 58000)
car5 = Car("Audi", "A3", 2021, 25000)

bst.addCar(car1)
bst.addCar(car2)
bst.addCar(car3)
bst.addCar(car4)
bst.addCar(car5)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          BMW,X5,[Car(BMW,X5,2022,60000),Car(BMW,X5,2020,58000)]    Tesla,Model3,[Car(Tesla, Model3,2018,50000)]
#                 /
#  Audi,A3,[Car(Audi,A3,2021,25000)]
```

InOrder Traversal

Using the `CarInventory` after the `addCar` methods above, an example of the `inOrder()` string format for removal is given below after removing the following Car:

```
bst.removeCar("BMW", "X5", 2020, 58000)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          BMW,X5,[Car(BMW,X5,2022,60000)]    Tesla,Model3,[Car(Tesla,Model3,2018,50000)]
#                 /
#  Audi,A3,[Car(Audi,A3,2021,25000)]

assert bst.inOrder() == \
"""\
Make: AUDI, Model: A3, Year: 2021, Price: $25000
Make: BMW, Model: X5, Year: 2022, Price: $60000
Make: MAZDA, Model: CX-5, Year: 2022, Price: $25000
Make: TESLA, Model: MODEL3, Year: 2018, Price: $50000
"""
```

and if we then remove the following car, the `CarInventoryNode` will be removed from the BST. The `CarInventory` and `inOrder()` string format is given below in this case:

```
bst.removeCar("BMW", "X5", 2022, 60000)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          Audi,A3,[Car(Audi,A3,2021,25000)]    Tesla,Model3,[Car(Tesla,Model3,2018,50000)]


assert bst.inOrder() == \
"""\
Make: AUDI, Model: A3, Year: 2021, Price: $25000
Make: MAZDA, Model: CX-5, Year: 2022, Price: $25000
Make: TESLA, Model: MODEL3, Year: 2018, Price: $50000
"""
```

PreOrder Traversal

Using the `CarInventory` after the `addCar` methods above, an example of the `preOrder()` string format is given below after removing the following Cars:

```
bst.removeCar("BMW", "X5", 2020, 58000)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          BMW,X5,[Car(BMW,X5,2022,60000)]    Tesla,Model3,[Car(Tesla,Model3,2018,50000)]
#                 /
#  Audi,A3,[Car(Audi,A3,2021,25000)]

assert bst.preOrder() == \
"""\
Make: MAZDA, Model: CX-5, Year: 2022, Price: $25000
Make: BMW, Model: X5, Year: 2022, Price: $60000
Make: AUDI, Model: A3, Year: 2021, Price: $25000
Make: TESLA, Model: MODEL3, Year: 2018, Price: $50000
"""

bst.removeCar("BMW", "X5", 2022, 60000)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          Audi,A3,[Car(Audi,A3,2021,25000)]    Tesla,Model3,[Car(Tesla,Model3,2018,50000)]


assert bst.preOrder() == \
"""\
Make: MAZDA, Model: CX-5, Year: 2022, Price: $25000
Make: AUDI, Model: A3, Year: 2021, Price: $25000
Make: TESLA, Model: MODEL3, Year: 2018, Price: $50000
"""
```

PostOrder Traversal

Using the `CarInventory` after the `addCar` methods above, an example of the `postOrder()` string format is given below after removing the following Cars:

```
bst.removeCar("BMW", "X5", 2020, 58000)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          BMW,X5,[Car(BMW,X5,2022,60000)]    Tesla,Model3,[Car(Tesla,Model3,2018,50000)]
#                 /
#  Audi,A3,[Car(Audi,A3,2021,25000)]

assert bst.postOrder() == \
"""\
Make: AUDI, Model: A3, Year: 2021, Price: $25000
Make: BMW, Model: X5, Year: 2022, Price: $60000
Make: TESLA, Model: MODEL3, Year: 2018, Price: $50000
Make: MAZDA, Model: CX-5, Year: 2022, Price: $25000
"""

bst.removeCar("BMW", "X5", 2022, 60000)

#                              Mazda,CX-5,[Car(Mazda,CX-5,2022,25000)]
#                             /                                      \
#          Audi,A3,[Car(Audi,A3,2021,25000)]    Tesla,Model3,[Car(Tesla,Model3,2018,50000)]


assert bst.postOrder() == \
"""\
Make: AUDI, Model: A3, Year: 2021, Price: $25000
Make: TESLA, Model: MODEL3, Year: 2018, Price: $50000
Make: MAZDA, Model: CX-5, Year: 2022, Price: $25000
"""
```

These are just a few simple examples illustrating the functionality of removing a Car from the `CarInventory` cars list, and removing the `CarInventoryNode` from the `CarInventory`. Gradescope will thoroughly test various cases. As always, it's important to thoroughly test your own code with various possible cases.

Other than the required methods, feel free to implement any helper methods that you think are useful in your implementation. The automated tests will test only your implementation of the required methods and certain methods from last week by creating a `CarInventory` containing various `Cars` with different `make`, `model`, `year`, and `price` attributes. The `removeCar()` and `addCar()` methods will be run, with `getCar()`, `getSuccessor()`, `inOrder()`, `preOrder()`, and `postOrder()` being used to verify that the `CarInventory` is fully functional. You should be sure that Lab08 is working correctly, and write tests to confirm your program for this lab is working properly.

testFile.py

This file should test all of the new methods in `CarInventory.py` using pytest. Think of and create your own various scenarios and edge cases when testing your code according to the given descriptions. For the `getSuccessor` method, your tests should test the general case and the case used for BST maintenance. For the `removeCar` method, you tests should cover Cases 1, 2, and 3 (as discussed in lecture) at a minimum, as well as only removing a Car from the `CarInventoryNode`'s cars list without removing the `CarInventoryNode`. Even though Gradescope will not use this file when running automated tests (Gradescope will use other tests), it is important to provide this file with various test cases (testing is important!!).

A note about Gradescope tests: Gradescope will use your functions to correctly check the state of your `Car`s and `CarInventory` with many scenarios. In order to test if everything is in the correct state, these tests use your CarInventory's `preOrder` / `inOrder` / `postOrder` traversals and `addCar` methods, as well as getting the string representation of your `Car`s and `CarInventoryNode`s to run other tests. It is important to ensure your `preOrder` / `inOrder` / `postOrder` traversals, your various string representations, and `CarInventory`'s `addCar` methods work correctly first or else many of the other tests may not pass.

Of course, feel free to reach out / post questions on Piazza as they come up!

Submission

Once you're done with writing your class definitions and tests, submit the following files to Gradescope's Lab09 assignment:

- `Car.py`
- `CarInventoryNode.py`
- `CarInventory.py`
- `testFile.py`

There will be various unit tests Gradescope will run to ensure your code is working correctly based on the specifications given in this lab.

If the tests don't pass, you may get some error message that may or may not be obvious at this point. Don't worry - take a minute to think about what may have caused the error. Try writing more test cases to see if you're able to reproduce the problem. If you're still not sure why you're getting the error, feel free to ask your TAs or Learning Assistants.

* Lab09 created by Priyanka Banerjee and Xingbu Qin, and adapted / updated by Richert Wang (F22)