Karan Bharaj (T00693289). Assignment 4- COMP 2231

Question 1

`BackPainAnalyzer` is tested on as per Listing 19.6 shown in page 746, and then tested with another two test cases. This is done after correcting completing the `LinkedBinaryTree` and `DecisionTree` classes.

```
          ======== QUESTION 1 ========

--- CASE 1- Test case as per Listing 19.6 ---
So, you're having back pain.
Did the pain occur after a blow or jolt?
Y
Do you have difficulty controlling your arms or legs?
N
Do you have pain or numbness in one arm or leg?
Y
You may have a muscle or nerve injury.
------- End of CASE 1 -------
```

- Test case as per Listing 19.6. Output is the same as the example shown.

```
-------- CASE 2 --------
So, you're having back pain.
Did the pain occur after a blow or jolt?
Y
Do you have difficulty controlling your arms or legs?
Y
Emergency! You may have damaged your spinal cord.
------- End of CASE 2 -------
```

- This test case is an alternate traversal of the tree. The output is as expected (agrees with Figure 19.13 in the textbook).

```
-------- CASE 3 --------
So, you're having back pain.
Did the pain occur after a blow or jolt?
N
Do you have a fever?
N
Do you have persistent morning stiffness?
Y
You may have an inflammation of the joints.
------- End of CASE 3 -------
```

- This test case is another alternate traversal of the tree. The output is as expected (agrees with Figure 19.13 in the textbook).

`DecisionTree` is now tested on a new, more complex decision tree that has an additional level. Three different traversals of the tree are tested and shown below.

```
         ======== QUESTION 1 ========

-------- CASE 1 --------
Looking for a Holiday Destination? Try this out!
Do you like sunshine?
Y
Do you like the beach?
Y
Do you prefer to visit a small island?
N
Do you want to stay in North America?
N
Visit Lisbon, Portugal!
------- End of CASE 1 -------
```

• First test case that recommends Lisbon as the holiday destination based on user input.

```
-------- CASE 2 --------
Looking for a Holiday Destination? Try this out!
Do you like sunshine?
N
Do you like skiing?
Y
Do you want to visit a European country?
Y
Would you like to go to Eastern Europe?
Y
Visit the Brezovica ski resort in Kosovo!
------- End of CASE 2 -------
```

• Second test case that recommends skiing in Kosovo as the holiday destination based on user input.

```
-------- CASE 3 --------
Looking for a Holiday Destination? Try this out!
Do you like sunshine?
Y
Do you like the beach?
N
Do you like wildlife?
Y
Would you like to travel to Africa?
Y
Visit Kenya and witness the Big Five!
------- End of CASE 3 -------
```

• Third test case that recommends a visit to Kenya to experience the wildlife as the holiday destination based on user input.

Question 2

Part 1

The missing methods in `LinkedBinarySearchTree` are completed. These methods are removeMax(), removeAllOccurrences(), find(), findMin(), findMax(), getLeft(), getRight(). The following are test cases for these methods:

```
          ======== QUESTION 1 ========
-- PART 1- Test cases for all newly implemented methods --

-- New Linked Binary Search Tree created with "5" --
------------ as the root element ------------

-- Nine integers are now added to the tree --
-- Inorder representation of the tree: --
1
2
2
2
2
3
5
5
7
9

-- Execution of removeMax() method: --
The largest element now removed from the tree is: 9
-- Inorder representation of the tree after removal: --
1
2
2
2
2
3
5
5
7

-- Execution of removeAllOccurrences() method: --
-- Inorder representation of the tree after removal: --
1
3
5
5
7

-- Execution of find() method: --
The attempt to find 3 in the tree: 3
-- Inorder representation of the tree after find(): --
1
3
5
5
7
```

- New Linked Binary Search Tree created with the integer “5” as the root.
- Additional 9 integers added to the tree. Tree printed out in an Inorder fashion.
- removeMax() called, which removes and returns the largest integer (9).
- Printout of the tree shows that 9 is no longer part of the tree.
- removeAllOccurences() called to remove all occurrences of the integer “2” in the tree.
- Printout shows that “2” is no longer part of the tree.
- find() called to identify if the integer “3” is part of the tree. It is found and returned.
- Printout of the tree shows that find() does not change the contents of the tree.

```
-- Execution of findMin() method: --
The smallest element in the tree is: 1
-- Inorder representation of the tree after findMin(): --
1
3
5
5
7

-- Execution of findMax() method: --
The largest element in the tree is: 7
-- Inorder representation of the tree after findMax(): --
1
3
5
5
7

-- Execution of getLeft() method: --
The left subtree of the tree (in inOrder fashion) is:
1
3

-- Inorder representation of the tree after getLeft(): --
1
3
5
5
7

-- Execution of getRight() method: --
The right subtree of the tree (in inOrder fashion) is:
5
7

-- Inorder representation of the tree after getRight(): --
1
3
5
5
7

-------- END of PART 1 --------
```

- findMin() called to return the smallest element (the integer “1”) in the tree.
- Subsequent printout of the tree shows that it remains unchanged after findMin() method is called.

- findMax() called to return the largest element (the integer “7”) in the tree.
- Subsequent printout of the tree shows that it remains unchanged after findMax() method is called.
- getLeft() called to return the subtree in inorder fashion.
- Subsequent printout of the tree shows that it remains unchanged after getLeft() method is called.

- getRight() called to return the subtree in inorder fashion.
- Subsequent printout of the tree shows that it remains unchanged after getRight() method is called.

Part 2

Edge cases for the methods removeAllOccurrences() and find() in `LinkedBinarySearchTree` are called below. It must be noted that this part of the code (i.e. "Part 2") is commented out in the Question2Driver.java file so as to allow the rest of the code to run. However, when run, the following errors are thrown:

When removeAllOccurences() is called on the Linked Binary Search tree from Part 1 to remove all occurrences of the Integer "2", the following "ElementNotFoundException" is thrown because "2" is no longer in the tree.

```
----- PART 2- EDGE case for the newly implemented methods -----

-- Execution of removeAllOccurrences() method to remove 2: --
Exception in thread "main" jsjf.exceptions.ElementNotFoundException: The target element is not in this LinkedBinarySearchTree
```

When find() is called on the Linked Binary Search tree from Part 1 to return any occurrence of the Integer "8", the following "ElementNotFoundException" is thrown because "8" is not in the tree.

```
----- PART 2- EDGE case for the newly implemented methods -----

-- Execution of find() method to try find 8: --
Exception in thread "main" jsjf.exceptions.ElementNotFoundException: The target element is not in this LinkedBinarySearchTree
```

Part 3

A balance tree method called "bruteForceBalance()" is created as part of `LinkedBinarySearchTree` that uses the brute force method to balance a tree. Two degenerate trees are created and balanced below using bruteForceBalance().

```
-- PART 3- Brute force balance on degenerate trees --

-- CASE 1- Creation of a degenerate tree --
-- Inorder representation of the degenerate tree: --
3
5
9
12
18
20

Size of the degenerate tree: 6
Height of the degenerate tree: 5
Root element of the degenerate tree: 3

-- Implementation of the brute force balance method --
-- Inorder representation of the balanced tree: --
3
5
9
12
18
20

Size of the balanced tree: 6
Height of the tree after balancing: 2
Root element of the balanced tree: 12

Root of left subtree: 5
Left subtree of root:
3
5
9

Root of right subtree: 20
Right subtree of root:
18
20

------- END of CASE 1 -------
```

- A linked binary search tree is created and elements are added to create a degenerate tree.
- Inorder representation of this degenerate tree is returned.
- The tree has 6 elements (its size), but a height of 5. To be balanced, its height should be 2 (using $\log_2 n$ method).
- The bruteForceBalance() method is called on the tree.
- As expected, the inorder representation of the tree remains unchanged
- Size of the tree remains unchanged. However, the height is reduced to 2, and the root of the tree is now 12.
- The left child of the root is 5, and the left subtree is as also shown in inorder fashion.
- The right child of the root is 20, and the right subtree is as also shown in inorder fashion.
- Given the height and subsequent printouts, it is shown that the tree is now balanced.

```
-- CASE 2- Creation of a degenerate tree --
-- Inorder representation of the degenerate tree: --
5
9
12
14
18
22
26
30

Size of the degenerate tree: 8
Height of the degenerate tree: 7
Root element of the degenerate tree: 30

-- Implementation of the brute force balance method --
-- Inorder representation of the balanced tree: --
5
9
12
14
18
22
26
30

Size of the balanced tree: 8
Height of the tree after balancing: 3
Root element of the balanced tree: 18

Root of left subtree: 12
Left subtree of root:
5
9
12
14

Root of right subtree: 26
Right subtree of root:
22
26
30

------- END of CASE 2 -------
```

- A linked binary search tree is created and elements are added to create a degenerate tree.
- Inorder representation of this degenerate tree is returned.
- The tree has 8 elements, but a height of 7. To be balanced, its height should be 3 (using $\log_2 n$ method).
- The bruteForceBalance() method is called on the tree.
- As expected, the inorder representation of the tree remains unchanged.
- Size of the tree remains unchanged. However, the height is reduced to 3, and the root of the tree is now 18.
- The left child of the root is 12, and the left subtree is as also shown in inorder fashion.
- The right child of the root is 20, and the right subtree is as also shown in inorder fashion.
- Given the height and subsequent printouts, it is shown that the tree is now balanced.

Part 4

In this part, the tree in case 2 that was balanced has 5 more elements added to it to make it a degenerate tree again. The "bruteForceBalance()" method is then called on the tree again to balance it. This is shown below:

```
--- PART 4- Insertions into balanced tree to create a degenerate ---
--- tree, and is then rebalanced again using bruteForceBalance() ---
-- Inorder representation of new degenerate tree: --
0
1
2
3
4
5
9
12
14
18
22
26
30

Size of the degenerate tree: 13
Height of the degenerate tree: 8
Root element of the degenerate tree: 18

-- Implementation of the brute force balance method --
-- Inorder representation of the balanced tree: --
0
1
2
3
4
5
9
12
14
18
22
26
30

Size of the balanced tree: 13
Height of the tree after balancing: 3
Root element of the balanced tree: 9

Root of left subtree: 3
Left subtree of root:
0
1
2
3
4
5

Root of right subtree: 22
Right subtree of root:
12
14
18
22
26
30

-------- END of PART 4 --------
```

- 5 additional number are added to the tree to unbalance it.
- Inorder representation of the this new degenerate tree is returned to show its contents.
- The tree has 13 elements, and height of 8. To be balanced, the height should be 3 (using $\log_2 n$ method). The root element is still 18.
- The bruteForceBalance() method is executed on the tree.
- As expected, the inorder representation of the tree remains unchanged.
- The balanced tree still has 13 elements. However, height is reduced to 3 and the new root is 9.
- The left child of the root is 3, and the left subtree is as also shown in inorder fashion.
- The right child of the root is 22, and the right subtree is as also shown in inorder fashion
- Given the height and subsequent printouts, it is shown that the tree is now balanced.