Proposal for testing the data structure python library

This proposal is about How to test a data structure python library written by Daliprice in github. The data structure python library includes six parts: AVL tree, Binary Search, Heap, LinkList, Queue and Stack. The Heap, LinkList, Queue and Stack can be regarded as one part to test because they are basic data structure. Binary Search is a kind of searching algorithm to search the value in the set. This library can be found at https://github.com//Daliprince/DataStructure.

First of all, I will test the basic data structure like linklist, heap, stack and so on in this library. I will test whether these data structure are built in right way or not. For example, in stack data structure, I will test if the push function push the value into the stack and the pop function can pop the right value out of the stack. I will try different type of input to add in the stack, linklist, queue and heap to check whether these data structure can accept different types of input. For example, I will create an adjacency table using the linklist data structure to check the whether the linklist can accept the structure as the value input.

AVL tree is the second part I want to test. In the beginning, I will add different value into the tree to see whether the value is in the right position. Meanwhile, some value will be added into the AVL tree to test whether the rotateLeftRight () function, rotateRightLeft () function, rotateRight () function and rotateLeft () function can keep the tree balance. Also, I will add some same value into the tree to test how the system will deal to them. Meanwhile, some strange or invalid input will be added to test whether the system can detect them and declare the problem. What is more, I will test whether the AVL tree can accept letter as the input and whether the system can compare the two letter or how the system sorts the letters. Additionally, I will add some capital letter and their lower case letter to test how the system will deal with them and sort them in order. Finally, I will test the rebalnceTree () function to see whether it can judge which rotate function the system will apply to keep the tree balance correctly or not.

Finally, I will test the binary search part. In this part, insert () and remove () function should be tested to check whether the system can add the value in right position and whether it can remove the value correctly. To test this section, I will add two same value to check if the system can still accept these input and check whether they are in the right position. Meanwhile, I will remove some value which don't exist in the set to test how the system will deal with this situation. Also, I will test whether this system can accept different types of input, such as float or string as their value.

To sum up, the whole test will focus on the input value, such as its type and whether the system works correctly.