Assignment 9: Binary Search Trees

CS 301

March 18, 2024

Now that you have seen what Binary Search Trees look like and operate, and how they may be traversed, let's have you implement one in Python, and analyze your implementation for running time of the various operations.

Implement a `BinaryTree` class in Python, naming your code file `BinaryTree.py`. You may choose to implement a node and reference style of `BinaryTree`, or a list of lists style of `BinaryTree`, but the implementation should have the following functions (along with the constructor that creates an empty BST):

1. `insert(item)` inserts the value in item at the correct position in the BST so far. Returns nothing.
2. `search(item)` returns `True` if it exists in the BST, `False` otherwise.
3. `sortedlist()` returns the values currently in the BST in ascending sorted order in a list. If there are no values in the tree, return an empty list.
4. `reverseSortedList()` returns the values currently in the BST in descending sorted order in a list. If there are no values in the tree, return an empty list.

For each method, describe the running time. Identify under what conditions would be the best case, worst case and average case running time for each method in big Oh notation, and why would that be the case in each situation. Particularly for the worst case running time, try to identify the flaw in BST's general definition that leads to the worst case. Provide your answers in comments in the code.

Grading for this assignment will be on:

1. Correctness of implementation.
2. Identifying best, worst and average running time for each method.
3. Identifying the reasons for each running time case.