CSCI 241 Data Structures
Laboratory Exercise 4

Objectives

- Implement Binary Search Tree
- Apply Binary Search Tree to problem solving

Implementation of Binary Search Tree

To access your assignment repository on GitHub, please click https://classroom.github.com/a/36YAojcG. You will need to clone the repository to your local machine so you can work and test it. Please refer to the Instruction of Lab 1 for the structure of the repository and where the java files are located.

Your task for this lab is to implement a Binary Search Tree class named "BST". As we discussed in class, your Binary Search Tree will be built upon a set of nodes. The Node class is provided to you in the starting code of your GitHub repository.

```
public class Node {
    int key;
    Node left, right, parent;

    public Node() {}

    public Node(int num) {
        key = num;
        left = null;
        right = null;
        parent = null;
    }
}
```

The most important thing is that a node as an object has four attributes: key, left, right, parent. Key holds the value of the node. Left and right refer to the left and right children nodes. Parent refers to the parent node.

To finish the design of your BST class, you will be working on the following attributes/methods:

- Insert: This method accepts one parameter, the target value (an integer), and will add a new node with that value into the Binary Search Tree. This method returns the root of the updated Binary Search Tree.

  public Node insert(int target)

- Search: This method accepts one parameter, the target value, and returns a boolean. If the target value is found, this method returns true, otherwise false.

  public boolean search(int target)

- Delete: This method accepts one parameter, the target value, and will delete the node with the target value if such a node exists. This method returns the root of the updated Binary Search Tree. You might need to create a set of other supporting methods for Delete method.

  public Node delete(int target)

- Traverse: This method performs an in-order traverse on all the nodes of the Binary Search Tree, and stores the traversed node keys in an integer array. The integer array will be returned by the method.

  public int[] traverse()

The above methods should NOT be declared as static, and you are required to use the given Node class.

Test your Binary Search Tree

Implement a method named "randomArray" in the file named "BSTTest.java" that tests a specific sorting algorithm. This method will take one parameter:

- size of a randomly generated array (int): e.g., `10`

This method will:

1. Generate and return a random array of the length specified by the parameter
2. The value range of each element should be between 0 and 1000 (inclusive)

```
>>> randomArray(5).
>>> [1, 2, 3, 7, 10]
```

Implement a method named "test" in the file named "BSTTest.java". This method should take advantage of the implemented randomArray and print out information as such:

```
Testing of BST starts.
Insert 3
Insert 2
Insert 5
Insert 100
Delete 3.
The traverse gives a BST as 3, 5, 100
Testing of BST ends.
```

Please note that the goal is not to manually code the printings of each line. Try your best to make this method as automatic as possible. There are no limits on what parameters this method shall accept or what it returns.

However, please make sure to use the method to perform testing for at least one case in your main method of BSTTest.java.

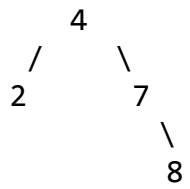
Application of your Binary Search Tree

Use your BST class to solve the following problem:

Given a binary search tree with non-negative values, find the minimum absolute difference between values of any two nodes.

Example:

Input:



Output: `1`

Explanation:

The minimum absolute difference is 1, which is the difference between 7 and 8.

Your solution class to this problem should be named as "BSTApp". The method that solves this problem should be declared static and named as "minDiff". minDiff takes one parameter, root of a binary search tree:

```
static public int minDiff(Node root)
```

You should also create a main method under this class, in which at least two test cases need to be given to test "minDiff" method by printing necessary information. For instance, the printed messages of an example are:

```
The traverse gives a BST as 2, 4, 7, 8.
The minimum absolute difference is 1.
```

Submission and Grading

The bulk of the implementation code is actually provided to you on https://github.com/Neo-Hao/data-structures. As a result, the automated testing will not be provided for this and only this lab. Setting up your testing code will mean significantly for you to get full points on this lab.

Please update the README file, and do not forget to make one last push to GitHub when everything is finished.

Please use comments to explain your program or any sections that are possibly difficult to understand. 10% of the lab points would be deducted if few or no comments can be found in your java files. If your java files fail to compile, you may risk getting a 0.