Lab 3 - Implementation of Set ADT Using Binary Search Trees

Due Date: Lab sessions March 1 – 12, 2021
Assessment: 5% of the total course mark.

Description:

In this lab you will implement the Set abstract data type using binary search trees. We will consider sets of integers where each integer will be stored in a tree node. Therefore, the number of nodes in the binary search tree must be equal to the size of the set.

You will have to write a `Java` class `BSTSet` for this purpose. To implement the tree nodes you must use the `Java` class `TNode` provided in this lab. Additionally, you will need to implement `Java` classes `MyStack` and `MyQueue` to perform non-recursive tree traversals of a `BSTSet`. You are *not permitted* to use any predefined `Java` methods or classes from `Java API`, other than for input and output, unless specified otherwise. You must also write a test class `TestBSTSet`. Please compute the asymptotic run time and space complexity of all methods and be ready to present them to the TA with explanations at your demo.

Definitions:

- A *set* is an unordered collection of elements with no repetition. Examples are the set of real numbers, the set of integer numbers or the set consisting of numbers $1, 2, 30$.
- For this lab we will only be considering representing finite sets consisting of elements which are integers. Examples: $\{0, 34, 78, 1000\}$, $\{4, 5, 890, 65535\}$, $\{0, 1, 2, \cdots, 65534, 65535\}$, $\{\}$ are all valid sets.
- The *union* of two sets, say A and B, is written as $A \cup B$ and is the set which contains all of the elements in either A or B or both. Example: If $A = \{3, 8, 14, 15\}$ and $B = \{2, 8, 9, 15, 100\}$, then $A \cup B = \{2, 3, 8, 9, 14, 15, 100\}$ (notice that there are no repeated elements in a set).
- The *intersection* of two sets A and B is written as $A \cap B$ and is the set which contains the elements that are common to A and B. Examples: If $A = \{3, 8, 14, 15\}$ and $B = \{2, 8, 9, 15, 100\}$, then $A \cap B = \{8, 15\}$. If $A = \{17, 20, 38\}$ and $B = \{200\}$, then $A \cap B = \{\}$, which is termed the *empty set*.
- The *difference* of two sets A and B is written as $A - B$ and is the set which contains the elements that are in A but not in B. Examples: If $A = \{3, 8, 12, 9\}$ and $B = \{9, 12, 15, 100\}$, then $A - B = \{3, 8\}$ and $B - A = \{15, 100\}$.

Specifications:

- You must use the `Java` class `TNode` given below, to implement the tree nodes. Classes `TNode` and `BSTSet` must be contained in the same package.

```
public class TNode{
        int element;
        TNode left;
        TNode right;
        TNode(int i, TNode l, TNode r)
            { element =i; left = l; right = r; }
}//end class
```

- Class `BSTSet` must contain **only** one `private` field named `root`, of type `TNode`, which is a reference to the root of the tree. No other fields are allowed. When the set is empty you should have `root==null`.
- Class `BSTSet` must contain at least the following constructors:
  - `public BSTSet()` - initializes the `BSTSet` object to represent the empty set (an empty tree).
  - `public BSTSet(int[] input)` - initializes the `BSTSet` object to represent the set containing all elements in array `input`, without repetitions. For example, if the array is: $5, 7, 4, 5$, then the corresponding set is $\{5, 7, 4\}$.
- Class `BSTSet` must contain the following `public` methods:
  1) `public boolean isIn(int v)`: Returns `true` if integer `v` is an element of `this BSTSet`. It returns `false` otherwise.
  2) `public void add(int v)`: Adds `v` to `this BSTSet` if `v` was not already an element of `this BSTSet`. It does nothing otherwise.
  3) `public boolean remove(int v)`: Removes `v` from `this BSTSet` if `v` was an element of `this BSTSet` and returns `true`. Returns `false` if `v` was not an element of `this BSTSet`.
  4) `public BSTSet union(BSTSet s)`: Returns a new `BSTSet` which represents the union of `this BSTSet` and `s`. This method should not modify the input sets.
  5) `public BSTSet intersection(BSTSet s)`: Returns a new `BSTSet` which represents the intersection of `this BSTSet` and `s`. This method should not modify the input sets.
  6) `public BSTSet difference(BSTSet s)`: Returns a new `BSTSet` which represents the difference of `this BSTSet` and `s`. This method should not modify the input sets.
  7) `public int size()`: Returns the number of elements in `this set`.
  8) `public int height()`: Returns the height of `this BSTSet`. Height of empty set is -1.
  9) `public void printBSTSet()`: Outputs the elements of `this set` to the console, in increasing order.

     `private void printBSTSet(TNode t)`: Outputs to the console the elements stored in the subtree rooted in `t`, in increasing order.

     For the two printing methods specified above you **must** use the following code

```
public void printBSTSet(){
    if(root==null)
```

```
            System.out.println("The set is empty");
        else{
            System.out.print("The set elements are: ");
            printBSTSet(root);
            System.out.print("\n");
        }
    }
    private void printBSTSet(TNode t){
        if(t!=null){
            printBSTSet(t.left);
            System.out.print(" " + t.element + ", ");
            printBSTSet(t.right);
        }
    }
```

9) `public void printNonRec()`: Prints the integers in `this BSTSet` in increasing order. This method is **nonrecursive** and uses a stack to implement the inorder traversal (use the class `MyStack`). See tutorial notes for the algorithm.

10) `public void printLevelOrder()`: Prints the integers in `this BSTSet` in level order, using a queue (use the class `MyQueue`). See tutorial notes for the algorithm.

NOTES:

- Compute the asymptotic run time and space complexity of all methods implemented as a function of the set size. Be prepared to present your derivations to the TA at demo time and provide verbal justification for your analysis.
- You are permitted to implement other `private` methods inside class `BSTSet` as required. However, you are not permitted to modify the class `TNode`. Additionally, class `BSTSet` must contain only one field which is at `TNode` reference named `root`.
- You may use the code from the lecture notes on binary search trees, stack and queues, or from the tutorial. However, you may need to adapt the code and provide complete references.
- To get credit for the lab you must demonstrate your code (i.e., class `BSTSet` and the test class) in front of a TA during your lab session. A 50% penalty will be applied for either a late demo or if the electronic submission of the code is late.

SUBMISSION INSTRUCTIONS:

NO REPORT IS NEEDED. Submit the source code for each of the classes `BSTSet` and `TestBSTSet` in a separte text file. Include your student number in the name of each file. For instance, if your student number is 12345 then the files should be named as follows: BSTSet12345.txt, TestBSTSet12345.txt, etc. Submit the files in the Assignment folder on Avenue by the end of your designated lab session.

BONUS:

A bonus of 1% of the course mark will be awarded if the second constructor creates a set stored in a tree of **minimum height** and, additionally, methods `union` and `intersection` guarantee that the tree storing the result (in other words, the union or intersection of the two input sets) has **minimum height** as well. You may write additional private methods for this purpose. You need to briefly describe in a comment at the beginning of the code of the method the main idea of your algorithm for each of the above methods and be prepared to justify this verbally to the TA during the demo.