CS102L Data Structures & Algorithms
Lab Week 15

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Spring 2024

Contents

Introduction

1.1 Instructions

- This lab will contribute 1% towards your final grade.
- The deadline for submission of this lab is at the end of lab time.
- The lab must be submitted online via CANVAS. You are required to submit a .zip file that contains all the *.py* files.
- The .zip file should be named as *Lab_15_aa1234.zip* where *aa1234* will be replaced with your student id.
- **Files that don't follow the appropriate naming convention will not be graded.**
- **Your final grade will comprise of both your submission and your lab performance.**

1.2 Marking scheme

This lab will be marked out of 100.

- 50 Marks are for the completion of the lab.
- 50 Marks are for progress and attendance during the lab.

1.3 Lab Objectives

The objective of this lab is:

- To learn how to create Binary Search Tree in Python

1.4 Late submission policy

There is no late submission policy for the lab.

1.5 Use of AI

Taking help from any AI-based tools such as ChatGPT is strictly prohibited and will be considered plagiarism.

1.6 Viva

Course staff may call any student for Viva to provide an explanation for their submission.

Prerequisites

2.1 Pytest

Pytest is a popular testing framework in Python known for its simplicity and power in facilitating efficient and scalable testing for applications and software. In order to use Pytest, one first needs to install it and create test functions prefixed with *test_* in a Python file. To install Pytest, you can write the following command on the terminal:

```
pip install pytest
```

After installing Pytest, one can execute all the tests by typing the following command on the terminal:

```
pytest
```

If you want to test any particular file, then you will have to specify the filename as follows:

```
pytest <filename>.py
```

In case the above mentioned pytest commands do not work on your system, even though pytest is installed, try using the following commands:

```
py -m pytest
```

```
py -m pytest <filename>.py
```

If all the tests pass successfully, then you should receive a similar output.

```
PS C:\Work\Spring2024\DSA\Lab1> pytest test_Question1.py
============================= test session starts ==============================
platform win32 -- Python 3.11.2, pytest-7.4.4, pluggy-1.3.0
rootdir: C:\Work\Spring2024\DSA\Lab1
collected 10 items

test_Question1.py ..........

============================== 10 passed in 0.06s ==============================
```

However, if all the tests fail, then you should receive a similar output.

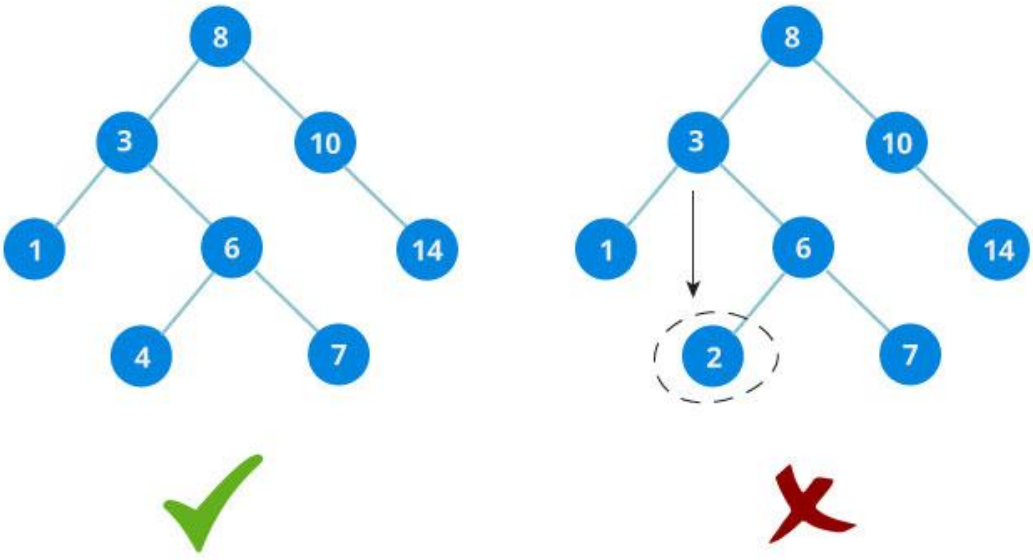
```
PS C:\Work\Spring2024\DSA\Lab1> pytest test_Question1.py
============================= test session starts ==============================
platform win32 -- Python 3.11.2, pytest-7.4.4, pluggy-1.3.0
rootdir: C:\Work\Spring2024\DSA\Lab1
collected 10 items

test_Question1.py FFFFFFFFFF

================================== FAILURES ===================================
_____________________________ test_question1[1st10-1st20-e54dc15ccafa608142ffa6340b035c327f972e24882052cfc3345f240f4ee237] ___
```

Lab Exercises

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.



This Binary Search Tree shown in Fig 3.1 has resulted from inserting the following keys in the order given: $8, 3, 10, 1, 6, 14, 4, 7$ and 13. We can represent the Binary Search Tree using Nested Dictionary in Python.

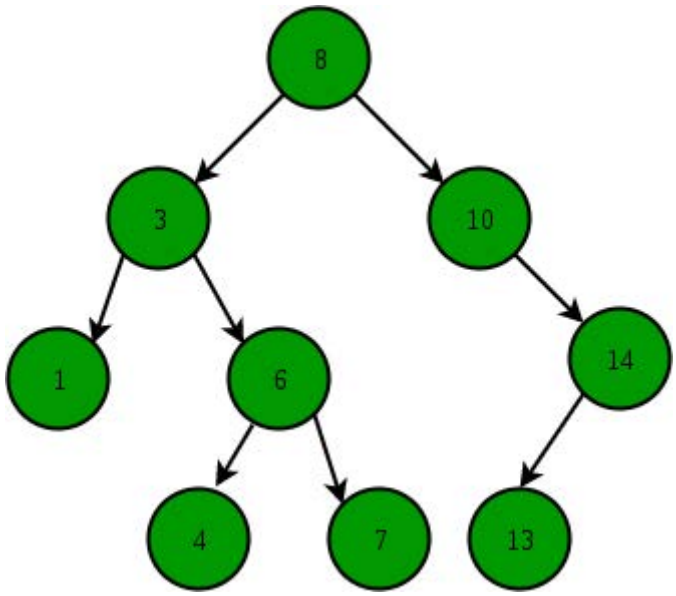


Figure 3.1: Resultant Binary Search Tree.

```
 {'value': 8, 'right': {'value': 10, 'left': {}, 'right':
 {'value': 14, 'left': {'value': 13, 'left': {}, 'right':
 {}}, 'right': {}}}, 'left': {'value': 3, 'left': {'value
': 1, 'left': {}, 'right': {}}, 'right': {'value': 6, '
left': {'value': 4, 'left': {}, 'right': {}}, 'right': {'
value': 7, 'left': {}, 'right': {}}}}}
```

3.1 Helper Functions

- **Insertion of a key**
  A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node. Write a function `insert(bst, key)` that takes a Binary Search Tree and key as a parameter and insert key into the Binary Search Tree.

- **Searching a key**
  To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree. Write a function `exist(bst, key)` that takes a Binary Search Tree and key as a parameter and returns `True` if the key exists; `False` otherwise.

- **Minimum Value in Binary Search Tree**
  Just traverse the node from root to left recursively until left is `NULL`. The node whose left is `NULL` is the node with minimum value. Write a function

`minimum(bst, starting_node)` that takes a Binary Search Tree & a starting node as a parameter and returns the minimum node in the BST from the starting node.

- **Maximum Value in Binary Search Tree**
  Just traverse the node from root to right recursively until right is `NULL`. The node whose right is `NULL` is the node with maximum value. Write a function `maximum(bst, starting_node)` that takes a Binary Search Tree and a starting node as a parameter and returns the maximum node in the BST from the starting node.

- **In-order Traversal in Binary Search Tree**

  1. Traverse the left subtree, i.e., call Inorder(left-subtree).
  2. Visit the root.
  3. Traverse the right subtree, i.e., call Inorder(right-subtree).

  Write a function `inorder_traversal(bst, result)` that takes a Binary Search Tree and a `result` array and stores in-order traversal of the Binary Search Tree in the `result` array.

- **Pre-order Traversal in Binary Search Tree**

  1. Visit the root.
  2. Traverse the left subtree, i.e., call Preorder(left-subtree).
  3. Traverse the right subtree, i.e., call Preorder(right-subtree).

  Write a function `preorder_traversal(bst, result)` that takes a Binary Search Tree and a `result` array and stores the pre-order traversal of the Binary Search Tree.

- **Post-order Traversal in Binary Search Tree**

  1. Traverse the left subtree, i.e., call Postorder(left-subtree).
  2. Traverse the right subtree, i.e., call Postorder(right-subtree).
  3. Visit the root.

  Write a function `postorder_traversal(bst, result)` that takes a Binary Search Tree and a `result` array and stores the post-order traversal of the Binary Search Tree.

- **In-order Successor in Binary Search Tree**
  In a Binary Search Tree, the In-order Successor of a given node is defined as the node with the smallest value greater than the value of the given node (Learn more here).

  Write a function `successor(BST, key, successor_node)` that takes a Binary Search Tree, a key and a successor (that is None by default) and returns the successor of that key.

These functions should be implemented in the file **HelperFunctions.py**, accessible within the Lab 15 module on CANVAS.

3.1.1 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_HelperFunctions.py
```

```
py -m pytest test_HelperFunctions.py
```

3.2 Question 1

1. Using the helper function `insert(bst, key)`, create the binary search tree that results from inserting the following keys in the order given: 68, 88, 61, 89, 94, 50, 4, 76, 66, and 82.
2. Using the helper function `exist(bst, key)`, check whether key 50 exists in resultant Binary Search Tree.
3. Using the helper function `exist(bst, key)`, check whether key 49 exists in resultant Binary Search Tree.
4. Using the helper function `minimum(bst, starting_node)`, find the node with the minimum value in resultant Binary Search Tree from starting node = 68.
5. Using the helper function `minimum(bst, starting_node)`, find the node with the minimum value in resultant Binary Search Tree from starting node = 88.
6. Using the helper function `maximum(bst, starting_node)`, find the node with the maximum value in resultant Binary Search Tree from starting node = 68.
7. Using the helper function `maximum(bst, starting_node)`, find the node with the maximum value in resultant Binary Search Tree from starting node = 61.
8. Using the helper function `inorder_traversal(bst, result)`, perform in-order traversal of the Binary Search Tree.
9. Using the helper function `preorder_traversal(bst, result)`, perform pre-order traversal of the Binary Search Tree.
10. Using the helper function `postorder_traversal(bst, result)`, perform post-order traversal of the Binary Search Tree.

11. Using the helper function `successor(BST, key, successor_node)`, find the successor of 76.

These functions should be implemented in the file **Questions1.py**, accessible within the Lab 15 module on CANVAS.

3.2.1 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question1.py
```

```
py -m pytest test_Question1.py
```

3.3 Question 2

In this task, you should create a new Binary Search Tree and organize input strings in the list `["begin", "do", "else", "end", "if", "then", "while"]` such that the keyword in the root is alphabetically bigger than all the keywords in the left subtree and smaller than all the keywords in the right subtree (and this holds for all nodes). Lastly, print this updated BST.

This function should be implemented in the file **Question2.py**, accessible within the Lab 15 module on CANVAS.

3.3.1 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question2.py
```

```
py -m pytest test_Question2.py
```