Namal University, Mianwali

Department of Electrical Engineering

Course Title: EE-253 Data Structures and Algorithms

LAB MANUAL

Lab 9 Trees in Python

V1: Designed by: Engr. Ali Hasnain

In previous lab, we have studied the concept of double ended queue (deque) in python. We have also studied how to implement a deque using array and performed different operations (insert_front(), insert_back(), delete_front(), delete_back(), etc.) on deque.

1. Lab Objectives

The objective of this lab is to introduce students to the concept of tree in python. In this lab, the students will enable the concepts of implementing a binary search tree (BST), accessing and searching the elements and operations which can be performed on BST. Students will be provided with examples, followed by performing lab tasks.

2. Lab Outcomes

- CLO:1 Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO:3 Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

- Software
    - IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as ‘Example’) are for practice purposes only while others (marked as ‘Task’) have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Background

Tree Data Structure:
A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy. Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.

Binary Search Tree (BST):
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.

In figure 1, we can see that the parent node has greater key value then the key value of left subtree but lesser key value then the key value of right subtree.

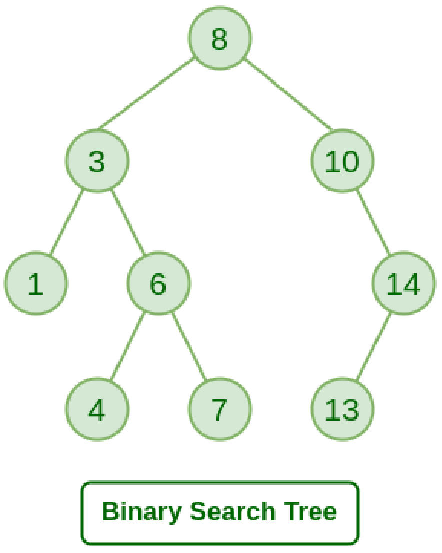

Figure 1 Example of BST

Basic Operation of BST:
Following are the basic operations of a BST:

- **Search** − Searches an element in a tree.
- **Insert** − Inserts an element in a tree.
- **Pre-order Traversal** − Traverses a tree in a pre-order manner.
- **In-order Traversal** − Traverses a tree in an in-order manner.
- **Post-order Traversal** − Traverses a tree in a post-order manner.

Search Operation on BST:
Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

Insert Operation on BST:
Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

__Example 1:__ Below, we present an example of BST and we add two basic operations (insert and search).

__Source code:__
```
class BST:
  def __init__(self, data):
    self.left = None
    self.right = None
    self.data = data

# Insert method to create nodes
  def insert(self, data):
    if self.data:
      if data < self.data:
        if self.left is None:
          self.left = BST(data)
        else:
          self.left.insert(data)
      elif data > self.data:
        if self.right is None:
          self.right = BST(data)
        else:
          self.right.insert(data)
      else:
        self.data = data
# search method to compare the value with nodes
  def search(self, key):
    if key < self.data:
      if self.left is None:
        return str(key)+" Not Found"
      return self.left.search(key)
    elif key > self.data:
      if self.right is None:
            return str(key)+" Not Found"
          return self.right.search(key)
        else:
          print(str(self.data) + ' is found')

root = BST(54)
root.insert(34)
root.insert(46)
root.insert(12)
root.insert(23)
root.insert(5)
print("Insertion Done")
print("Searching Start")
print(root.search(5))
print(root.search(12))
```

Output:

```
Insertion Done
Searching Start
5 is found
None
12 is found
None
```

Pre-order Traversal Operation on BST:

The pre-order traversal operation in a BST visits all its nodes. However, the root node in it is first printed, followed by its left subtree and then its right subtree.

In-order Traversal Operation on BST:

The in-order traversal operation in a BST visits all its nodes in the following order:

- Firstly, we traverse the left child of the root node/current node, if any.
- Next, traverse the current node.
- Lastly, traverse the right child of the current node, if any.

Post-order Traversal Operation on BST:

Like the other traversals, post-order traversal also visits all the nodes in a BST and displays them. However, the left subtree is printed first, followed by the right subtree and lastly, the root node.

Example 2: Below, we present an example of BST and we add two basic operations (insert and pre-order traversal).

Source code:

```
class BST:
  def __init__(self, data):
    self.left = None
    self.right = None
    self.data = data

# Insert method to create nodes
  def insert(self, data):
    if self.data:
      if data < self.data:
        if self.left is None:
          self.left = BST(data)
        else:
          self.left.insert(data)
      elif data > self.data:
        if self.right is None:
          self.right = BST(data)
        else:
          self.right.insert(data)
      else:
        self.data = data

# Print the tree
  def Preorder(self):
    print(self.data)
    if self.left:
      self.left.Preorder()
    if self.right:
      self.right.Preorder()

root = BST(54)
root.insert(34)
root.insert(46)
root.insert(12)
root.insert(23)
root.insert(5)
print("Preorder Traversal of Binary Search Tree: ")
root.Preorder()
```

Output:

```
Preorder Traversal of Binary Search Tree:
54
34
12
5
```

23
46

6. Lab Tasks:

Task 1: Using above example1 code, insert 10 data elements and make a method (Inorder) for in-order traversal of data in BST. Also make a method (print_bst) to print the BST.

Task 2: Using above example2 code, insert 8 data elements and make a method (Postorder) for post-order traversal of data in BST. Also make a method (print_bst) to print the BST.

Task 3: Using above code, insert 15 data elements and make a method (delete) to delete the nodes from BST. Also make a method (print_bst) to print the BST before and after deletion of data.

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested byothers. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend forthe instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	

V1: Designed by: Engr. Ali Hasnain