Dr. S. Alper SERT
alper.sert@ceng.metu.edu.tr
Cihad TEKİNBAŞ
ctekinbas@ceng.metu.edu.tr

Assignment 02
METU CENG310 Fall 2023-2024
Data Structures and Algorithms with Python

Start Date: December 15th, 2021
Due Date: December 22th, 2022

1 Linked Binary Tree

In this assignment, you are expected to complete the LinkedBinaryTree class, namely `LinkedBinaryTree`, that represents a binary tree using a linked structure. This means that each node in the tree is an object, and nodes are connected through references (links). The class includes methods for constructing and manipulating the binary tree. `LinkedBinaryTree` will have five incomplete methods: `def fill(self,elements):`, `def levelorder(self):`,`def preorder(self, p=None):`,`def inorder(self, p=None):`and ,`def postorder(self, p=None):`.`fill` to fill the binary tree with elements in a level-order manner. Level-order filling ensures that the resulting tree is balanced, and it's a common way to construct binary trees, which you will implement in this assignment. `levelorder`,`preorder`,`levelorder`,`inorder` and `postorder`methodS will traverse the tree in level-order,in-order,pre-order and post-order fashion and print traversal of the tree rooted at Position p.

2 Code Templates

In this assignment, we will provide some code templates that can boost your progress. Please find them in the `tree.py` file. You can make slight changes to the template if you would like, however, the expected functionality (reading,traversals, filling and printing tree, etc.) should be in your code.

3 The `LinkedBinaryTree` Class

This class should implement a general tree ADT by adopting a link-based approach. In doing so, you are expected to use the base class `Tree` whose implementation is provided in the Chapter 8 of the textbook (The code is provided in the `tree.py` file.). As you have already learned, by inheriting a base class, all the members and the methods of the base class are inherited to the child class, in other words, they become directly usable by the instances of child class. However, some of the methods of the `Tree` base class are not implemented, so you will implement them in the `LinkedBinaryTree` class.

Also note that, as we covered in our lectures, we will have a distinction between the node and position objects. Nodes will contain the actual data and other references such as parent and children that form the tree. Positions are just pointers to the nodes of the tree, indicating their positions. Here you are not expected to change or improve the `Position` subclass and related `_validate` and `_make_position` methods.

For this assignment, you are not expected to implement methods for node replacing or deletions. So, you can safely consider this tree as a write-only structure.

4 TODO

`def fill(self, elements)`

The `fill` method is responsible for populating the binary tree with elements provided in a list. The elements are added to the tree in a level-order manner, ensuring that each level is filled from left to right. The method takes a list of elements as its parameter and uses a level-order traversal approach to add nodes to the tree. The root of the tree is set as the first element in the list, and subsequent elements are added as left and right children of the existing nodes in the order they appear in the list.

`def levelorder(self)` The `levelorder` method in the provided Python code performs a level-order traversal of the binary tree, visiting nodes level by level from left to right. It utilizes a breadth-first approach, starting from the root and systematically exploring each level before moving to the next. The traversal is implemented using a queue data structure.

`def preorder(self, p=None)`

The `preorder` method in the provided Python code generates a preorder traversal of the binary tree rooted at a specified position (p). Preorder traversal is a depth-first traversal that explores the tree in the order of root, left subtree, and right subtree.

```
def inorder(self, p=None)
```

The `inorder` method generates an inorder traversal of a binary tree rooted at a specified position or the entire tree if no position is provided. In an inorder traversal, the left subtree is explored first, followed by the current node, and then the right subtree.

```
def postorder(self, p=None)
```

The `postorder` method generates a postorder traversal of a binary tree rooted at a specified position or the entire tree if no position is provided. In a postorder traversal, the left and right subtrees are explored first, followed by the current node

```
def read_elements_from_file(self, filename)
```

The `read_elements_from_file` method reads elements from a file, assuming one element per line, and returns a list containing these elements.

```
1
2
3
4
5
```

5 Appendices

This assignment document has two appendices: `tree.py`, `test.txt`.

6 Delivery Instructions

Please hand in your module as a single file named as `tree.py` over ODTUClass by 11:59pm on due date. An Assignment-02 page will be generated soon after the start date of this assignment. Should you have any questions pertaining to this assignment, please ask them in advance (rather than on the due date) for your own convenience. Whatever IDE you use, you have to make sure that your module could be run on a Python interpreter.

```python
from tree import LinkedBinaryTree
tree = LinkedBinaryTree()

# Fill the tree with elements in a level-order manner
elements = tree.read_elements_from_file("test.txt")
tree.fill(elements)

# Display tree properties
print("Tree size:", len(tree))
print("Root element:", tree.root().element())
print("Left child of the root:", tree.left(tree.root()).element())
print("Right child of the root:", tree.right(tree.root()).element())
tree.levelOrder()
print("Level traversal:", list(tree.levelOrder()))
# Perform traversals and print the elements
print("Preorder traversal:", list(tree.preorder()))
print("Inorder traversal:", list(tree.inorder()))
print("Postorder traversal:", list(tree.postorder()))
#Output
#Tree size: 5
#Root element: 1
#Left child of the root: 2
#Right child of the root: 3
#Level traversal: ['1', '2', '3', '4', '5']
```

```
#Preorder traversal: ['1', '2', '4', '5', '3']
#Inorder traversal: ['4', '2', '5', '1', '3']
#Postorder traversal: ['4', '5', '2', '3', '1']
```