Dr.Erdem KAYA
ekaya@metu.edu.tr
Mehmet Taha ŞAHİN
staha@metu.edu.tr

Assignment 02
METU CENG310 Fall 2021-2022
Data Structures and Algorithms with Python

Start Date: December 23rd, 2021
Due Date: January 2nd, 2022

1 Product Categorizer

In this assignment, you are expected to implement a categorizer class, namely `ProductCategorizer`, that takes in categorical data and maintains the categorical relationships such as parent-child and sibling relationships. `ProductCategorizer` will have two methods: `def fill_tree(self):` and `def print_tree(self):`. `fill_tree` will read categorical data from a text file and fill up an internally maintained general tree based on the structure that can be inferred from the data. The tree that will be used by `ProductCategorizer` will be an instance of `LinkedTree`, which you will implement in this assignment. The `print_tree` method will traverse the tree in a pre-order and post-order fashion and print the categorical data in a more proper way compared to the input file.

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 (adding nodes, traversals, filling and printing tree, etc.) should be in your code.

3 The `LinkedTree` 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 `LinkedTree` 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.

A linked tree is formed with nodes that are connected to each other. Nodes will be implemented with the `_Node` nested class. It should store only data members that point to element (the reference to the data stored), parent (reference to parent node, None for root), and a reference to the list that will be holding the children of the node. By convention, the `_children` reference should be `None` if a node does not have any children.

Most of the methods of `LinkedTree` are the ADT operations that are common to any kind of trees: `__len__`, `root`, `parent`, `num_children`, `children`, `is_root`, `is_leaf`. You will implement two methods for adding a new node to the tree: `_add_root` and `_add_nonroot_node`, as their names speak for themselves, they will be used for adding the root and non-root nodes, respectively. However, you should hide away this detail from the users of your class. Due to that reason, you will also implement a (conventionally) public method, `add_node`, that is smart enough to decide whether to use `_add_root` or `_add_nonroot_node` method.

The `LinkedTree` should also provide generators yielding the nodes according to well-known traversing algorithms. The `_traverse_preorder` method should yield nodes according to the pre-order traverse algorithm whilst `_traverse_postorder` should do the same with post-order traverse approach. Again, we would like to offer this functionality to the users of the `LinkedTree` class as a public method, namely `all_nodes`, which will take a parameter (`mode`) and decide whether to adopt pre-order or post-order traversal.

The users of the `LinkedTree` should also be able to get the list of nodes that forms a path from a given position to root (`get_path_to_root`).

And finally, given a position and a value, `LinkedTree` should be able to find the child node of position having the inquired value (`find_child_by_value`). For example, if position `p` has a child node `c` with value `23.4`, the call `find_child_by_value(p, 23.4)` should return a `Position` object pointing to node `c`.

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 The `ProductCategorizer` Class

`ProductCategorizer` is a simple class that has two basic operations: Read categorical data from a text file and print them out in pre- or post-oder manner.

The `fill_tree` method should take a file path as a parameter and read the categorical data line by line. An example data file is provided in `Assignment2Input.txt`. In the input file, data looks like this:

```
World
World,Asia
World,Asia,China
```

Each line in the file should be considered as a path in the tree starting from root to a particular node. For example, for the line `World,Asia,China`; `World` should be the root node, `Asia` must be one of its children, and `China` has to be a child of `Asia`. So the filling up the tree can be done in the following manner (You can do it in different ways as well if you would like to): Read each line of the input file in the order they are listed in the file and check whether the given path exists or not. If it exists do nothing, otherwise create the nodes in the tree such that the parent-child relationship that can be inferred from the input file exists in the tree. For example, for the example input file, after the first line, a root node with a string element `'World'` will be created. After the second line, a child node of root with a string element `'Asia'` will be created.

An important point to mention is that the values maintained in the children of a particular node have to be distinct. In other words, there cannot be two children of a node with the same value.

We should also be able to print out the content of our tree according post- and pre-order of the nodes into an output text file. For that reason, we will need to implement the `print_tree` method that will print out the tree in both post- and pre-order limit and write to a text file. Example output files are provided in the assignment page as `Assignment2OutputPre.txt` and `Assignment2OutputPost.txt`. The method should generate two output files for post- and pre-order traversals, and the hierarchy should be explicit in the output files with tabs (When deciding the number of tabs in a line, you may want to use the depth property of a node.).

5 Appendices

This assignment document has four appendices: `tree.py`, `Assignment2Input.txt`, `Assignment2OutputPre.txt`, and `Assignment2OutputPost.txt`.

6 Delivery Instructions

Please hand in your module as a single file named as `categorize.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.

```
from categorize import ProductCategorizer

pc = ProductCategorizer('Assignment2Input.txt')
pc.fill_tree()
pc.print_tree()
```