Lab09: Tree Traversal
CSE2050 Fall 2017
Instructor: Jeffrey Meunier & Wei Wei
TA: Jenny Blessing, Param Bidja, Yamuna Rajan & Zigeng Wang

1. Introduction

In this exercise, we will implement two Python functions in order to practice tree traversal algorithms.

2. Objectives

The purpose of this assignment is to help you:

- Practice preorder tree traversal and postorder tree traversal algorithms.
- Refresh knowledge on tree data structure.
- Get familiar with some tree traversal related applications.

***Note**: Before you start, if you are not familiar with tree, tree traversal algorithms, you are recommended to review your lecture notes and the additional materials from Dr. Don Sheehy.*

3. Background

3.1. Trees

Trees data types are ideal for representing hierarchical structure. Trees are composed of nodes and nodes have 0 or more children or child nodes. A node is called the parent of its children. Each node has (at most) one parent. If the children are ordered in some way, then we have an ordered tree. We are concerned primarily with rooted trees, i.e. there is a single special node called the root of the tree. The root is the only node that does not have a parent. The nodes that do not have any children are called leaves or leaf nodes.

There are many examples of hierarchical (tree-like) structures:

- Class Hierarchies (assuming single inheritance). The subclass is the child and the superclasses is the parent. The python class object is the root.
- File folders or directories on a computer can be nested inside other directories. The parent-child relationship encode containment.
- The tree of recursive function calls in the execution of a divide-and-conquer algorithm. The leaves are those calls where the base case executes.

Even though we use the family metaphor for many of the naming conventions, a family tree is not actually a tree. The reason is that these violate the one parent rule.

When we draw trees, we take an Australian convention of putting the root on the top and the children below. Although this is backwards with respect to the trees we see in nature, it does correspond to how we generally think of most other hierarchies.

3.2. Tree Traversal

Previously, all the collections we stored were either sequential (i.e., list, tuple, and str) or non-sequential (i.e., dict and set). The tree structure seems to lie somewhere between the two. There is some structure, but it's not linear. We can give it a linear (sequential) structure by iterating through all the nodes in the tree, but there is not a unique way to do this. For trees, the process of visiting all the nodes is called **tree traversal**. For ordered trees, there are two standard traversals, called **preorder** and **postorder**, both are naturally defined recursively.

In a preorder traversal, we *visit* the node first followed by the traversal of its children. In a postorder traversal, we traverse all the children and then visit the node itself. The *visit* refers to whatever computation we want to do with the nodes. The **printpreorder** method is a classic example of a **preorder** traversal and **printpostorder** method is a classic example of **postorder** traversal.

```
def printpreorder(self):
    print(self.data)
    for child in self.children:
        child.printtree()
```

```
def printpostorder(self):
    for child in self.children:
        child.printpostoder()
    print(self.data)
```

(All the contents in this section are almost directly copied and pasted from Dr. Don Sheehy's note online[1]. All the credit given to Dr. Don Sheehy.)

4. Assignment

In this assignment, we need to complete two methods in a given tree class. These two methods are both following the tree traversal philosophy.

4.1. Print Book Content

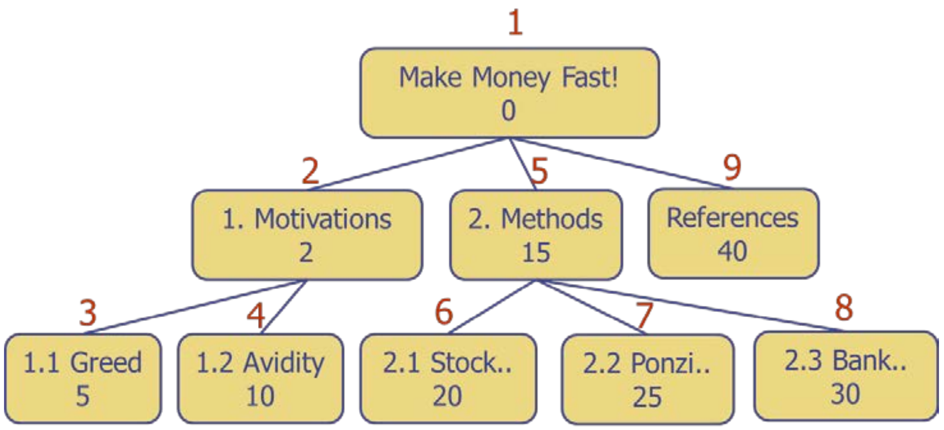
In the background section, we mentioned that trees are ideal for representing hierarchical structure. In our daily life, book, this object, follows a standard class hierarchy which can be represented in a tree manner, where a book contains chapters and a chapter contains different sections. In this subsection, we will write a method called **printbookcontent** which can print the table of contents of a book in a nice way.

[1] Cited from Don Sheehy's Notes. https://github.com/donsheehy/datastructures/blob/master/prose/13_trees.md

Now, we may have a look at the skeleton code, which initializes a table of book contents as an instance of Tree like the following,

```
Tbook = [('Make Money Fast!', 0), [('1. Motivations', 2), [('1.1 Greed', 5)], [('1.2
Avidity', 10)]], [('2. Methods', 15), [('2.1 Stock Fraud', 20)], [('2.2 Ponzi
Scheme', 25)], [('2.3 Bank Robbery', 30)]], [('References', 40)]]
treeBook = Tree(Tbook)
```

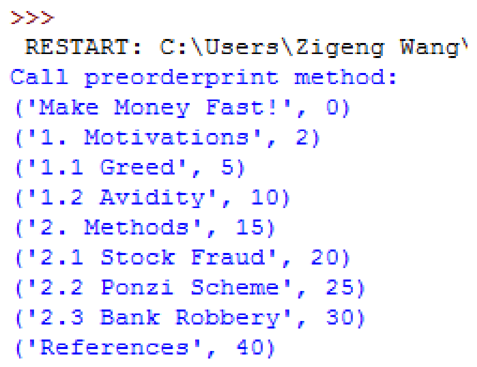
The tree can be visualized as the following. As we can see, the data contained in each node of the tree is a tuple. Each tuple contains the content name and the corresponding page.



An easy way to print the table of contents of this book is to call the **printpreorder** method which has been successfully implemented in the **Tree** class,

```
Tbook = [('Make Money Fast!', 0), [('1. Motivations', 2), [('1.1 Greed', 5)], [('1.2
Avidity', 10)]], [('2. Methods', 15), [('2.1 Stock Fraud', 20)], [('2.2 Ponzi
Scheme', 25)], [('2.3 Bank Robbery', 30)]], [('References', 40)]]
treeBook = Tree(Tbook)
print(" Call preorderprint method:")
treeBook.printpreorder()
```

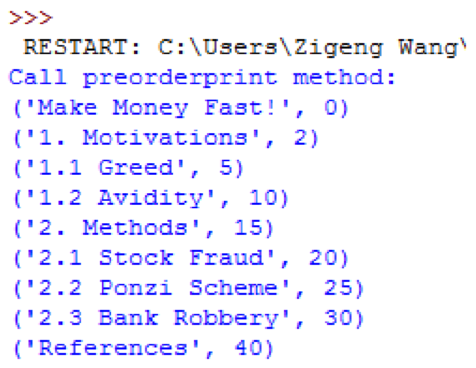
The output is,



```
>>>
 RESTART: C:\Users\Zigeng Wang\
Call preorderprint method:
('Make Money Fast!', 0)
('1. Motivations', 2)
('1.1 Greed', 5)
('1.2 Avidity', 10)
('2. Methods', 15)
('2.1 Stock Fraud', 20)
('2.2 Ponzi Scheme', 25)
('2.3 Bank Robbery', 30)
('References', 40)
```

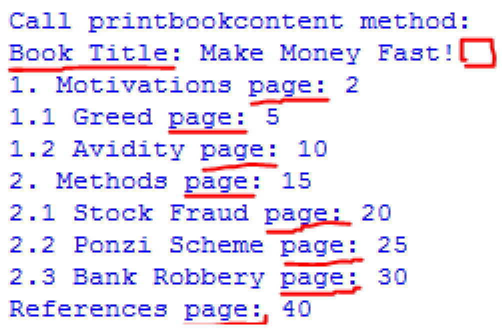
The output above shows the book contents clearly but not *nicely*. So, in this subsection, we will implement a modified version of **printpreorder** method, called **printbookcontent** method, so that the table of contents can be printed nicely as the following on the right-hand side:

```
print("Preorder Print Book Content:")
treeBook.printbookcontent()
```



```
>>>
 RESTART: C:\Users\Zigeng Wang\
Call preorderprint method:
('Make Money Fast!', 0)
('1. Motivations', 2)
('1.1 Greed', 5)
('1.2 Avidity', 10)
('2. Methods', 15)
('2.1 Stock Fraud', 20)
('2.2 Ponzi Scheme', 25)
('2.3 Bank Robbery', 30)
('References', 40)
```

original preorderprint method



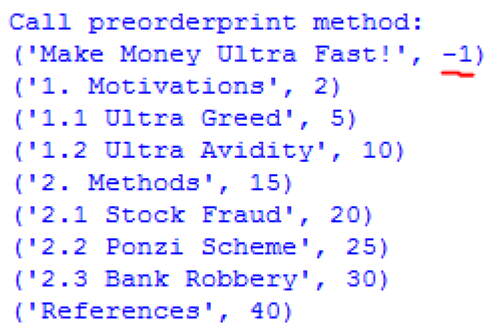
```
Call printbookcontent method:
Book Title: Make Money Fast!
1. Motivations page: 2
1.1 Greed page: 5
1.2 Avidity page: 10
2. Methods page: 15
2.1 Stock Fraud page: 20
2.2 Ponzi Scheme page: 25
2.3 Bank Robbery page: 30
References page: 40
```

printbookcontent method

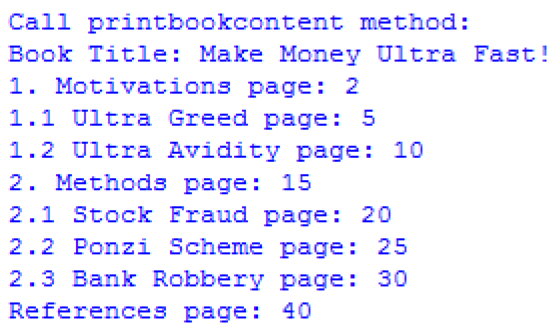
As we can see in the two figures above, the differences between the outputs of the two methods are clearly marked. Compared with the original **preorderprint** method, we make **printbookcontent** method really fetch and print the content of each node in the tree but not just pure tuples. Further, we added a ***<u>Book Title:</u>*** before the title of the book, and we skip printing the page information of the title. Also, for all the following chapters and sections, we added ***<u>page:</u>*** before each page number.

Another thing that we need to consider is that, when we are implementing the **printbookcontent** method, the method needs to be robust for different book titles and their corresponding pages. For example, the page of the book title can be 0, or -1, or 1 or None or even some special numbers that we not know. So, we cannot locate the node which contains book title by simply checking its page. Instead, we need to take advantage of the observation that book title node is always the root.

Following are two more examples, where **printbookcontent** method can always locate the book title node with different corresponding page numbers.



```
Call preorderprint method:
('Make Money Ultra Fast!', -1)
('1. Motivations', 2)
('1.1 Ultra Greed', 5)
('1.2 Ultra Avidity', 10)
('2. Methods', 15)
('2.1 Stock Fraud', 20)
('2.2 Ponzi Scheme', 25)
('2.3 Bank Robbery', 30)
('References', 40)
```



```
Call printbookcontent method:
Book Title: Make Money Ultra Fast!
1. Motivations page: 2
1.1 Ultra Greed page: 5
1.2 Ultra Avidity page: 10
2. Methods page: 15
2.1 Stock Fraud page: 20
2.2 Ponzi Scheme page: 25
2.3 Bank Robbery page: 30
References page: 40
```

```
Call preorderprint method:
('Make Money Super Fast!', None)
('1. Motivations', 2)
('1.1 Super Greed', 5)
('1.2 Super Avidity', 10)
('2. Super Methods', 15)
('2.1 Stock Fraud', 20)
('2.2 Ponzi Scheme', 25)
('2.3 Bank Robbery', 30)
('References', 40)
```

```
Call printbookcontent method:
Book Title: Make Money Super Fast!
1. Motivations page: 2
1.1 Super Greed page: 5
1.2 Super Avidity page: 10
2. Super Methods page: 15
2.1 Stock Fraud page: 20
2.2 Ponzi Scheme page: 25
2.3 Bank Robbery page: 30
References page: 40
```

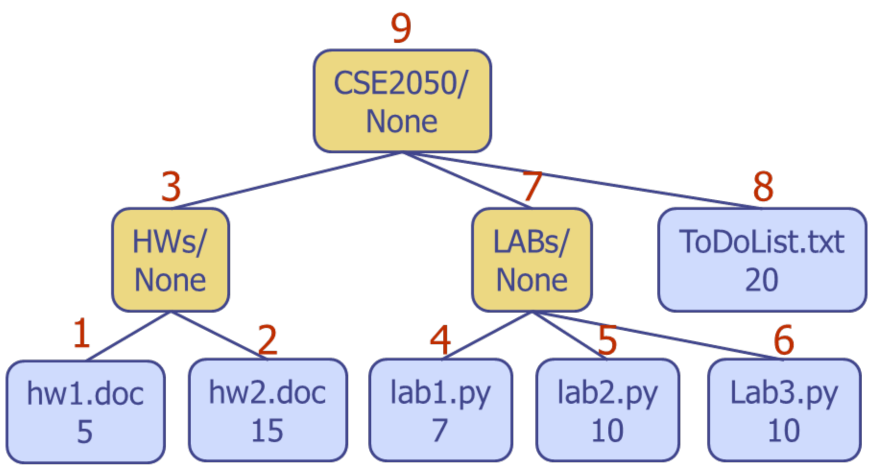
4.2. Space Computation in File System

In the background section, we mentioned that computer file system can be represented in a tree manner, where file folders or directories on a computer can be nested inside other directories. And, in this subsection, we will implement one method which helps to compute the space used by files in a directory and its subdirectories in a tree-structured file system.

As we can see in the skeleton code, we initialize a file system like the following,

```
Tfile = [('CSE2050/', None), [('HWs/', None), [('hw1.doc', 5)], [('hw2.doc', 15)]],
[('LABs/', None), [('lab1.py', 7)], [('lab2.py', 10)], [('lab3.py', 10)]],
[('ToDoList.txt', 20)]]
treeFile = Tree(Tfile)
```

We can visualize the file system as the following. Similar to the previous example, the data contained in each node of the tree is also a tuple. The tuple contains the folder or file name and the size of each file, while the space of each directory (folder) is temporally unknown.

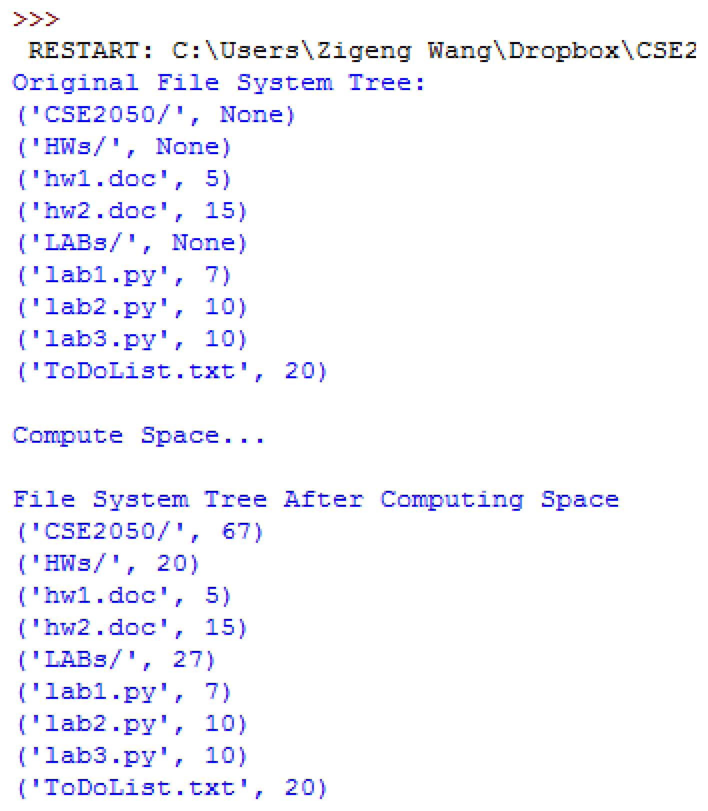


So, here we will implement the **computespace** method. This method will help to compute the space used by files in the root directory and its subdirectories (CSE2050/, HWs/ and LABs/). The size of the files and/or the size of the subdirectories will be summed up to be the size of their parent directory. This method can be implemented by taking advantage of the postorder tree traversal algorithm. In the file system tree figure, the red numbers above each of the node shows the recommended order that each node should be visited.

A simple example is shown as the following for your reference.

```
Tfile = [('CSE2050/', None), [('HWs/', None), [('hw1.doc', 5)], [('hw2.doc', 15)]],
[('LABs/', None), [('lab1.py', 7)], [('lab2.py', 10)], [('lab3.py', 10)]],
[('ToDoList.txt', 20)]]
treeFile = Tree(Tfile)
print("Original File System Tree:")
treeFile.printpreorder()
print("\nCompute Space...\n")
treeFile.computespace()
print("File System Tree After Computing Space")
treeFile.printpreorder()
```

The corresponding output is as the following. We can see, the original file system is initialized without the information of the space usage of directories. After calling the **computespace** method, the space used by files in the root directory and its subdirectories are updated.



```
>>>
 RESTART: C:\Users\Zigeng Wang\Dropbox\CSE2
Original File System Tree:
('CSE2050/', None)
('HWs/', None)
('hw1.doc', 5)
('hw2.doc', 15)
('LABs/', None)
('lab1.py', 7)
('lab2.py', 10)
('lab3.py', 10)
('ToDoList.txt', 20)

Compute Space...

File System Tree After Computing Space
('CSE2050/', 67)
('HWs/', 20)
('hw1.doc', 5)
('hw2.doc', 15)
('LABs/', 27)
('lab1.py', 7)
('lab2.py', 10)
('lab3.py', 10)
('ToDoList.txt', 20)
```

5. Submit your work to Mimir

Submit your code to Mimir after you complete your code. Please delete or comment out all the testing codes before your submission (We should only keep the **Tree** Class). The Mimir will automatically grade your submission based on different test cases. You can submit your code to Mimir up to **30 times** to refresh your existing score before the submission deadline.

6. Due date

This lab assignment is worth **2 points** in the final grade. It will be due by **11:59pm on Wednesday, Nov 22nd 2017**. A late submission penalty of **10% per day** will be deducted from your grade, starting at 12:00am.

7. Getting help

Start your project early, because you will probably not be able to get timely help in the last few hours before the assignment is due.

- Go to the office hours of instructors and TAs.
  - Prof. Wei Wei: Mon. 2:30 - 3:15pm, Wed. 2:30 - 3:15pm, Thurs. 2:30 - 3:15pm, Fri. 2:30 - 3:15pm @ITE258
  - Jenny Blessing: Fri. 12pm - 2pm @ITE140
  - Param Bidja: Tues. 2pm - 3pm @ITE140
  - Yamuna Rajan: Tues. 11am - 12pm, Wed. 9:30am - 10:30am @ITE140
  - Zigeng Wang: Mon. 3pm - 4pm @ITE140
- Post your questions on Piazza. Instructors, TAs and many of your classmates may answer your questions.
- Search the answer of your unknown questions on the Internet. Many questions asked by you might have been asked and answered many times online already.