

ENGR 131 – Winter 2020-2021 – Programming Assignment 2

Depth First Search on a Binary Tree

1 Introduction

Binary trees are a type of data structure that consists of a list of nodes. Each node may have a parent. Each node may also have two children, called here (left_child, right_child). Nodes without a parent are called root nodes. Nodes without children are called leaf nodes. In the example below, node 0 is a root node, nodes 4,5,6,7,8 are leaf nodes. One of the main techniques for working with binary trees is to *traverse* the tree, visiting each node in a specific order. One important algorithm for traversing a binary tree is called *depth first search*. This is the algorithm you will use for this assignment. Depth first search explores as far as possible along the left_child path. When there are no more left children to explore, the algorithm backs out to the right_child and continues exploring. The depth first search algorithm is typically implemented using recursion. For the example tree shown in Figure 1, the depth first search order would be [7, 8, 3, 4, 1, 5, 6, 2, 0].

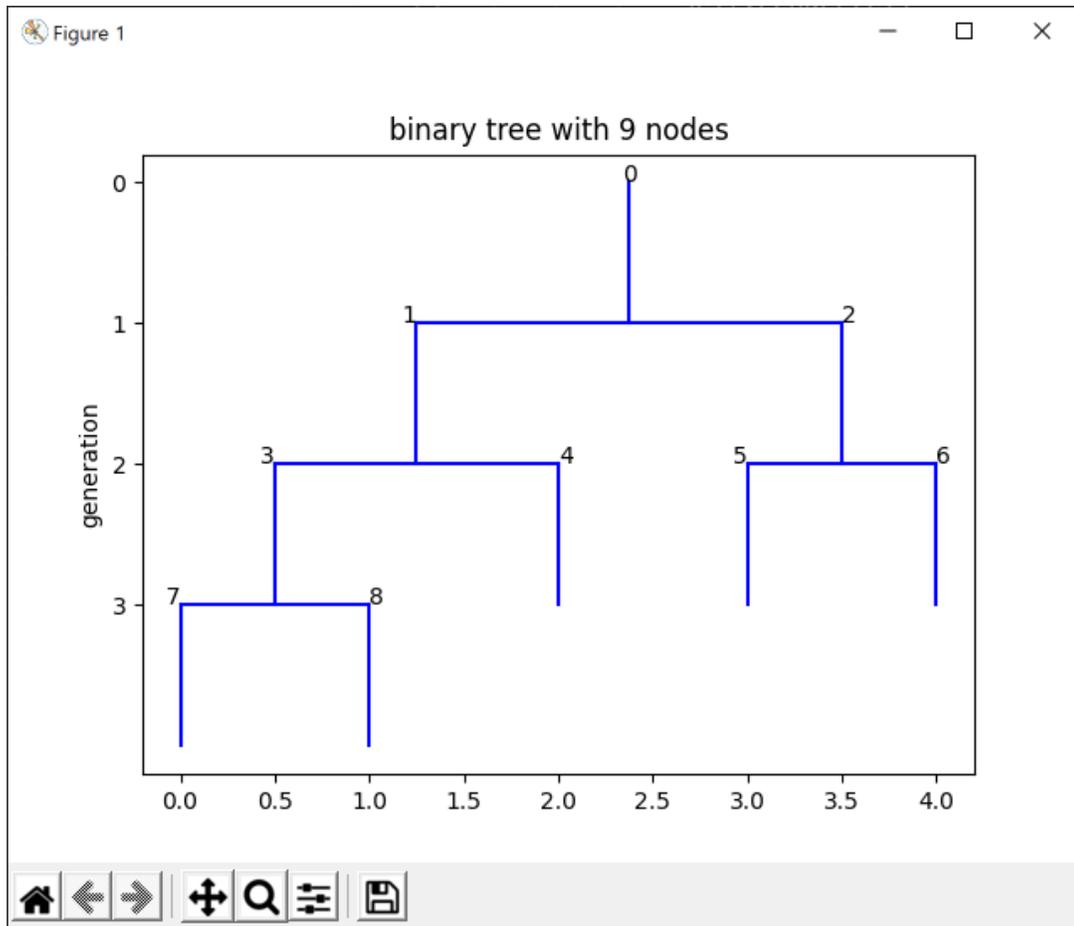


Figure 1: Example binary tree with 9 nodes. The root node (0) has no parent. The leaf nodes are [4,5,6,7,8]. Your submission will produce the same plot formatting, but with the number of nodes computed as specified in the problem specification. Note the alignment on the text nodes (center for root, right aligned for left_child and left aligned for right_child). The y axis is inverted so that increasing generations render down the page, with ticks draw only at integer generation values. The number of nodes in the binary tree was calculated using the function `myNumberOfNodes('me123@drexel.edu')`.

For this programming assignment, you will develop a Python class that implements the functions and data management associated with binary trees. The programming template contains a list of the class methods and member variables that you must implement.

Algorithm 1: Traversing a binary tree with depth first search to set (x,y) render locations for each node.

Result: A list of node_ids in depth first search order, with x and y values for each node
Initialize the binary tree structure with N nodes, initialize a variable XMAX to store the rightmost leaf node found so far, and call dfsTraverse initially on the root node.

```
1 def dfsTraverse(node, XMAX=0, y=0):
2     # if node has children
3     # traverse left child first
4     (dfs, XMAX) = dfsTraverse(node.left_child, XMAX, child_y)
5     # we get to this line after all our descendents left_children
6     # have been processed
7     (dfs, XMAX) = dfsTraverse(node.right_child, XMAX, child_y)
8     # set parent node locations midway between children
9     node.x = (node.left_child.x + node.right_child.x) / 2.0
10    node.y = y
11    # if node has no children, set leaf location at next integer
12    # value
13    node.x = XMAX
14    # we increment y at each generation, so the root is at y=0,
15    # its children are at y=1, etc...
16    node.y = y
17    XMAX += 1
18    # after processing children, add node to dfs list
19    dfs.append(node)
20    return (dfs, XMAX)
```

THE ASSIGNMENT

This programming assignment has **two** deliverables:

1. A Python program that implements functionality to generate a binary tree, produce a depth first search on its nodes, and plot the resulting tree. This program will be submitted to ZyBooks in ZyLab 18.25. The Pythonscript you write must conform to all requirements listed in the next section. The Zylab problem includes a code template that provides partial implementations for some of the required functions.
2. A short report documenting your program, including an explanation of how you implemented your solution. You will include with this a rendered version of a binary tree produced by your program that matches exactly the formatting in figure 1. The number of nodes in your binary tree will be computed as a function of your Drexel email address, as described in the next section. A PDF version of this report will be submitted via Drexel Learn. Your report must conform to all requirements listed below.

PYTHON PROGRAM REQUIREMENTS

1. Your program must define a `binaryTree` class with the following specifications:

- (a) Your `binaryTree` class must have as data attributes a dictionary called `nodeChildren` that maps a node to its (`left_child`,`right_child`) nodes, a dictionary called `nodeParent` that maps a node to its parent node, and a dictionary called `treeXY` that maps a node to its (`x`,`y`) location.
 - (b) Your `binaryTree` class must have a method called `addTrio(parent, left_child, right_child)` that adds a new pair of nodes (`left_child`, `right_child`) to the tree. If `parent` is the root then `parent` is added to the tree as well.
 - (c) Your `binaryTree` class must have a method called `leaves` that returns a list of all leaf nodes in the tree and a method called `nodes` that returns a list of all nodes in the tree.
 - (d) Your `binaryTree` class must have a method called `addNodes(nNodes)` adds at most `nNodes` to the tree, plus an additional root node if the tree is empty. For example, calling `addNodes(4)` on an empty tree would result in a tree with 5 nodes. Calling `addNodes(3)` on a tree that already has 3 nodes also results in a tree with 5 nodes since we don't need to add a new root node in this case. Why does the number of nodes have to be odd? Calling `addNodes(nNodes)` on a non-empty tree adds `nNodes` additional nodes to the tree. NOTE - the function `addNodes` in the template provided in Zybooks will implement this for you correctly.
 - (e) Your `binaryTree` class must have a method called `dfsTraverse` that returns a depth first search ordered list of nodes in the tree. See Figure 1 and the code template in Zybooks for more detail.
 - (f) Your `binaryTree` class must have a method called `render` that produces a plot of a binary tree with the number of nodes determined as described below. The leaf nodes are placed at integer locations in increasing order of their discovery by the depth first search. Parent nodes are set midway between each leaf node. Each node should have a vertical line extending from their `node.y` to `node.y+1`. Node labels should be printed at their (`x`,`y`) location, with `left_child` nodes right aligned, and `right_child` nodes left aligned. The root text label should be centered.
2. Your program must include a function at global scope called `myNumberOfNodes` that maps your drexel email address to the number of nodes in the plot that you will submit.

```

1     def myNumberOfNodes(emailAddress):
2         # pass your email address (you123@drexel.edu) as the
           parameter
3         hash=0
4         for c in emailAddress:
5             hash+=ord(c)
6         return 17+hash%15

```

Be sure to put your own Drexel email in for the `emailAddress` parameter.

REPORT REQUIREMENTS

1. Use a font size no bigger than 12pt and no smaller than 11pt.
2. Use 1-in. margins.
3. There is no length requirement or restriction.
4. Explain how you implemented the depth first search.
5. Explain when you invoke the depth first search.
6. Embed a figure showing a rendering of the binary tree with the number of nodes obtained from your `myNumberOfNodes()` function. See Figure 1 for an example.
7. Your figure must have a caption. Your figure caption starts with a name, *e.g.* Figure 1. You should then have a title sentence, indicating what the figure contains. You should then include a few descriptive sentences describing the figure. The last sentence of your figure caption must include your drexel email as well as the hash value and computed number of nodes from the `myNumberOfNodes` function (above).
8. Be sure to refer to the figure from the report text.
9. The report should include your name, lecture section, and lab section near the top.
10. The filename of your report should be of the format:
`<FirstName>-<LastName>-Programming-Assignment-2-DFS.pdf`