```python
import ast
from matplotlib import pyplot as plt
# common = { 'col_offset', 'lineno'}
# Python Language Dictionary = PLD refers to Python version - 3.5.2
from PLD2 import PLD
import pickle

def add_to_list(base, sep, bits):
    return [sep.join([base, bit]) for bit in bits]


class SourceTree(object):

    def __init__(self):
        from textfile import text
        self.text = text
        self.tree = ast.parse(self.text)
        self.head = self.tree
        self.name = 'tree'
        self.parent_array = {}
        self.child_array = {}
        self.depth_dict = {0: 1}
        self.nodes_at_a_depth = {}
        self.nodevals = []
        self.end_list = []
        self.count = 0
        self.queue = [(self.count, self.name)]
        self.function_defs = {}

    def add_to_endlist(self, item):

        self.end_list.append(item)

    def queue_peek(self):

        return self.queue[0]

    def queue_add(self, item):
        self.queue.append(item)

    def queue_del(self, item):

        try:
            self.queue.remove(item)
        except:
            print(item," was not found in the queue list, but attempt was made 
to remove. called by queue_del(child)")

    def get_child_of(self, item):
        number, value = item
        return self.child_array[number]

    def get_parent_of(self, item):
        number, value = item
        return self.parent_array[number]

    def get_children_of(self, item):
        (count, parent_full_name) = item
        child_list = [(count, parent_full_name)]
        print('starting analysis of ', parent_full_name)
        parent_is_list = eval('isinstance(self.{0}, list)'.format(parent_full_name))
        if parent_is_list:
            child_list_len = len(eval('self.'+parent_full_name))
            list_of_children = list(map(lambda x: '[{0}]'.format(x), range(child_list_len)))
            child_list.extend(add_to_list(parent_full_name, '', list_of_children))
            print('children from list  ', child_list)
        else:
            parent = eval('type(self.{0})'.format(parent_full_name))
            parent_value = str(parent)

            # hackey extraction of type name from a node, but they don't have names - need alternative
            parent_val = (parent_value)[13:-2] if parent_value.startswith("<class '_ast.") else parent_value[
                                            8:-2]
            if parent_val in PLD:
                sub_list = PLD[parent_val]
                for unit in sub_list:
                    child_list.append('.'.join([parent_full_name, unit]))
            else:
                print("'",parent_val, "' is not in the PLD dictionary ")
                print("child_list returning for '{0}' analysis = {1}".format(parent_full_name, child_list))
        return child_list




    def print_out(self):
        for row in sorted(self.end_list):
            rowlen = len(str(row[1]))
            print(row[0], ' ' * (60 - rowlen), row[1], ' ', eval('self.'+row[1]))

    def mainrun(self):
        while len(self.queue) > 0:
            parent_sent = self.queue[0]
            current_nodes_children = self.get_children_of(parent_sent)
            for child in current_nodes_children:
                current_count = self.count
                if child == parent_sent:
                    print(child, ' has returned')
                    self.add_to_endlist(parent_sent)
                    current_parent = parent_sent[0]
                    self.queue_del(child)
                else:
                    print(child, ' is a new node for the queue list')
                    self.count += 1
                    self.queue_add((self.count, child))
                    self.parent_array[self.count] = current_parent
                    self.child_array.setdefault(current_parent, []).append(self.count)
            print('queue list currently holds : ', self.queue)
            print('end_list currently holds : ', self.end_list)
            print('Node count is at ', self.count)
        print('end result = ', self.end_list)
        print('The Parent reference dictionary contains :',self.parent_array)
        print('The Children reference dictionary contains :',self.child_array)
        self.print_out()
        self.height()
        self.reverse_height()
        self.get_function_defs()
        self.nodeval()
        return self.end_list, self.child_array, self.parent_array


    def get_depth(self, item):
        if item in self.depth_dict.keys():
            self.depth = self.depth_dict[item]
        else:
            self.depth = 1 + self.get_depth(self.parent_array[item])
            self.depth_dict[item] = self.depth
        return self.depth

    def height(self):
        for item in self.parent_array:
            self.get_depth(item)
        print(self.depth_dict)
        return self.depth_dict

    def reverse_height(self):
        for key, value in self.depth_dict.items():
            self.nodes_at_a_depth.setdefault(value, []).append(key)
        self.layer_widths = [(key, len(value)) for key, value in self.nodes_at_a_depth.items()]

    def nodeval(self):
        for item in self.end_list:
            num, valstr = item
            valtype = eval('str(type((self.{0})))'.format(valstr))
            carn = self.child_array[num] if num in self.child_array.keys() else []
            valtype2 = valtype
            if "class '_ast." in valtype:
                valtype2 = str(valtype)[13:-2]
            else:
                valtype2 = str(valtype)[8:-2]

            self.nodevals.append((num, carn, valstr, valtype, valtype2,  eval('self.'+item[1])))
            print((num, carn, valstr, valtype, valtype2, eval('self.'+item[1])))

    def get_function_defs(self):
        pass
        # this function is to gather the node elements that are FunctionDef in type and all of their child nodes
        # each entry in the dictionary is for a separate FunctionDef




def main():
    structure = SourceTree()
    end_list, car, par = structure.mainrun()
    print('max width is {0} node columns, with {1} rows'.format(max([val[1] for val in structure.layer_widths]), len(structure.layer_widths)))
    print('structure.nodevals=',structure.nodevals)

    with open('filename.pickle', 'wb') as file_handle:
        pickle.dump(structure.nodevals , file_handle)


if __name__=='__main__':
    main()
```