DS2030 Data Structures and Algorithms for Data Science
Lab 4 (Take Home) Due on September 12 11.59pm

Instructions

- You are to use Python as the programming language. Use may use Visual Studio Code (or any other editor you are comfortable with) as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. You are encouraged engage in high level discussions with your classmates; however ensure to include their names in the report/code documentation. If you refer to any source on the Internet, include the corresponding citation in the report/code documentation. If we find that you have copied code from your classmate or from the Internet, you will get a straight fail grade in the course.
- The submission must be a zip file with the following naming convention - rollnumber.zip. The Python files should be contained in a folder named after the question number.
- Include appropriate comments to document the code. Include a `read me` file containing the instructions on for executing the code. The code should run on institute linux machines.
- Upload your submission to moodle by the due date and time. Do not email the submission to the instructor or the TA.

This lab will improve your understanding of tree structures, and traversals.

Python Lab for Identical and Isomorphic Binary Trees

Given two binary trees $T1$ and $T2$, the program below checks if the trees are identical or isomorphic. Two trees are **identical** if they have the same structure and node values. Two trees are **isomorphic** if they can be made identical by a series of swaps of the left and right children of some nodes. If the trees are isomorphic, the program also counts the number of valid ways to transform $T1$ into $T2$ using swaps of left and right children.

Input Format

The input consists of two text files representing two binary trees. Each line in the files represents a node with its structure: node_number, parent_node_number, left_child_number, right_child_number. The node_number starts from 1 (root). If any entry is 0, it means the corresponding child or parent is absent.

Python Code

```
class TreeNode:
    def __init__(self, node_number, parent, left, right):
        self.node_number = node_number
        self.parent = parent
        self.left = left
        self.right = right

def build_tree_from_file(filename):
```

```
    """
    Build a binary tree from the input file.

    :param filename: str - The filename containing tree data.
    :return: the tree
    """


    return nodes[1] if 1 in nodes else None

def are_identical(root1, root2):
    """
    Check if two binary trees are identical.

    :param root1: TreeNode - Root of the first binary tree.
    :param root2: TreeNode - Root of the second binary tree.
    :return: bool - True if trees are identical, False otherwise.
    """

def are_isomorphic(root1, root2):
    """
    Check if two binary trees are isomorphic.

    :param root1: TreeNode - Root of the first binary tree.
    :param root2: TreeNode - Root of the second binary tree.
    :return: int - The number of ways to transform T1 into T2 using swaps.
    """

def count_isomorphic_ways(root1, root2):
    """
    Count the number of valid ways to transform T1 into T2 using swaps.

    :param root1: TreeNode - Root of the first binary tree.
    :param root2: TreeNode - Root of the second binary tree.
    :return: int - The number of valid ways.
    """


def main():
    # Load trees from input files
    tree1_root = build_tree_from_file('tree1.txt')
    tree2_root = build_tree_from_file('tree2.txt')

    # Check if trees are identical

    # Check if trees are isomorphic print the numbre of valid ways to transform T1 into
    T2

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

Code Fragment 1: Python code to check identical and isomorphic binary trees

Traversal Checks for Isomorphic Trees

After determining if the trees are isomorphic, we perform the following traversal checks:

1. Pre-order Traversal Check

The **Pre-order Traversal** of a binary tree visits the root node first, followed by the left subtree, and then the right subtree. To check if the pre-order traversal of $T1$ can be rearranged to match the pre-order traversal of $T2$ using the allowed swaps, we proceed as follows:

- Perform a pre-order traversal on both trees $T1$ and $T2$.

- Compare the traversal results to determine if the pre-order sequence of $T1$ can be rearranged to match $T2$.
- Swaps can be performed between left and right children at any node to achieve this rearrangement.

2. Post-order Traversal Check

The **Post-order Traversal** of a binary tree visits the left subtree first, then the right subtree, and finally the root node. To check if the post-order traversal of $T1$ can be rearranged to match the post-order traversal of $T2$ using the allowed swaps, we proceed as follows:

- Perform a post-order traversal on both trees $T1$ and $T2$.
- Compare the traversal results to determine if the post-order sequence of $T1$ can be rearranged to match $T2$.
- Similar to pre-order checks, swaps between left and right children at any node are allowed to achieve this rearrangement.

1 References

1. M Goodrich, R Tamassia, and M. Goldwasser, "Data Structures and Algorithms in Python", 1^{st} edition, Wiley, 2013.