TERM PROJECT #2

CODE:

TermProject#2 Version control

TProject#2.py

```python
ss Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


# Helper Functions for Traversals
def preorder_traversal(root):
    if root is None:
        return []
    return [root.value] + preorder_traversal(root.left) + preorder_traversal(root.right)


def inorder_traversal(root):
    if root is None:
        return []
    return inorder_traversal(root.left) + [root.value] + inorder_traversal(root.right)


def postorder_traversal(root):
    if root is None:
        return []
    return postorder_traversal(root.left) + postorder_traversal(root.right) + [root.value]


# Function to Construct Binary Tree from Expression
def construct_expression_tree(expression):
    operators = set(['+', '-', '*', '/', '^'])
    stack = []

    for char in expression.split():
        if char not in operators:
            stack.append(Node(char))
        else:
            node = Node(char)
```

for i, tree in enumerate(trees_...

TermProject#2 Version control

TProject#2.py

```python
        else:
            node = Node(char)
            node.right = stack.pop()
            node.left = stack.pop()
            stack.append(node)

    return stack[-1] if stack else None


# Function to Construct Binary Tree from Matrix
1 usage
def construct_tree_from_matrix(vertices, left_children, right_children):
    nodes = {v: Node(v) for v in vertices}

    for parent, left, right in zip(vertices, left_children, right_children):
        if left != '-':
            nodes[parent].left = nodes[left]
        if right != '-':
            nodes[parent].right = nodes[right]

    return nodes[vertices[0]]


# Example Implementation for Part A (Equations)
expressions = [
    "3 5 * 4 5 * 6 7 - + -",
    "a b + c * d e - -",
    "a b ^ c d + + e f * g h + / +",
    "a b + c d e f ^ - * /",
    "a b - c + d e + f g / * *",
    "5 2 + 2 1 - * 2 9 + 7 2 - 1 - + / 8 *"
]

trees_part_a = [construct_expression_tree(expr) for expr in expressions]

print("Part A: Traversals")
for i, tree in enumerate(trees_part_a, 1):
    print(f"Equation {i}:")
    print("Preorder:", preorder_traversal(tree))
    print("Inorder:", inorder_traversal(tree))
    print("Postorder:", postorder_traversal(tree))
```

TermProject#2 Version control

TProject#2.py

```python
for i, tree in enumerate(trees_part_a, 1):
    print(f"Equation {i}:")
    print("Preorder:", preorder_traversal(tree))
    print("Inorder:", inorder_traversal(tree))
    print("Postorder:", postorder_traversal(tree))
    print()

# Example Implementation for Part B (Matrices)
vertices_list = [
    ["r", "a", "b", "c", "d", "e", "f", "g", "h"],
    ["r", "a", "b", "c", "d", "e", "f", "g"],
    ["r", "a", "b", "c", "d", "e", "f"],
    ["r", "a", "b", "c", "d", "e", "f", "g", "h", "i"]
]

left_children_list = [
    ["a", "b", "-", "e", "-", "-", "-", "-", "-"],
    ["a", "c", "-", "-", "-", "-", "-", "-"],
    ["a", "-", "d", "f", "-", "-", "-"],
    ["a", "c", "e", "g", "-", "i", "-", "-", "-", "-"]
]

right_children_list = [
    ["-", "c", "d", "f", "-", "g", "-", "h", "-"],
    ["b", "d", "e", "-", "-", "f", "g", "-"],
    ["b", "c", "e", "-", "-", "-", "-"],
    ["b", "d", "f", "h", "-", "-", "-", "-", "-", "-"]
]

trees_part_b = [
    construct_tree_from_matrix(vertices, left, right)
    for vertices, left, right in zip(vertices_list, left_children_list, right_children_list)
]

print("Part B: Traversals")
for i, tree in enumerate(trees_part_b, 1):
    print(f"Matrix {i}:")
    print("Preorder:", preorder_traversal(tree))
    print("Inorder:", inorder_traversal(tree))
    print("Postorder:", postorder_traversal(tree))
    print()
```

for i, tree in enumerate(trees_...

OUTPUT:

```
Z:\DSALG01\TermProject#2\.venv\Scripts\python.exe Z:\DSALG01\TermProject#2\TProject#2.py
Part A: Traversals
Equation 1:
Preorder: ['-', '*', '3', '5', '+', '*', '4', '5', '-', '6', '7']
Inorder: ['3', '*', '5', '-', '4', '*', '5', '+', '6', '-', '7']
Postorder: ['3', '5', '*', '4', '5', '*', '6', '7', '-', '+', '-']

Equation 2:
Preorder: ['-', '*', '+', 'a', 'b', 'c', '-', 'd', 'e']
Inorder: ['a', '+', 'b', '*', 'c', '-', 'd', '-', 'e']
Postorder: ['a', 'b', '+', 'c', '*', 'd', 'e', '-', '-']

Equation 3:
Preorder: ['+', '+', '^', 'a', 'b', '+', 'c', 'd', '/', '*', 'e', 'f', '+', 'g', 'h']
Inorder: ['a', '^', 'b', '+', 'c', '+', 'd', '+', 'e', '*', 'f', '/', 'g', '+', 'h']
Postorder: ['a', 'b', '^', 'c', 'd', '+', '+', 'e', 'f', '*', 'g', 'h', '+', '/', '+']

Equation 4:
Preorder: ['/', '+', 'a', 'b', '*', 'c', '-', 'd', '^', 'e', 'f']
Inorder: ['a', '+', 'b', '/', 'c', '*', 'd', '-', 'e', '^', 'f']
Postorder: ['a', 'b', '+', 'c', 'd', 'e', 'f', '^', '-', '*', '/']

Equation 5:
Preorder: ['*', '+', '-', 'a', 'b', 'c', '*', '+', 'd', 'e', '/', 'f', 'g']
Inorder: ['a', '-', 'b', '+', 'c', '*', 'd', '+', 'e', '*', 'f', '/', 'g']
Postorder: ['a', 'b', '-', 'c', '+', 'd', 'e', '+', 'f', 'g', '/', '*', '*']

Equation 6:
Preorder: ['*', '/', '*', '+', '5', '2', '-', '2', '1', '+', '+', '2', '9', '-', '-', '7', '2', '1', '8']
Inorder: ['5', '+', '2', '*', '2', '-', '1', '/', '2', '+', '9', '+', '7', '-', '2', '-', '1', '*', '8']
Postorder: ['5', '2', '+', '2', '1', '-', '*', '2', '9', '+', '7', '2', '-', '1', '-', '+', '/', '8', '*']

Part B: Traversals
Matrix 1:
Preorder: ['r', 'a', 'b', 'd', 'c', 'e', 'g', 'h', 'f']
Inorder: ['b', 'd', 'a', 'e', 'g', 'h', 'c', 'f', 'r']
Postorder: ['d', 'b', 'h', 'g', 'e', 'f', 'c', 'a', 'r']

Matrix 2:
```

```
Matrix 2:
Preorder: ['r', 'a', 'c', 'd', 'b', 'e', 'f', 'g']
Inorder: ['c', 'a', 'd', 'r', 'b', 'e', 'f', 'g']
Postorder: ['c', 'd', 'a', 'g', 'f', 'e', 'b', 'r']

Matrix 3:
Preorder: ['r', 'a', 'c', 'f', 'b', 'd', 'e']
Inorder: ['a', 'f', 'c', 'r', 'd', 'b', 'e']
Postorder: ['f', 'c', 'a', 'd', 'e', 'b', 'r']

Matrix 4:
Preorder: ['r', 'a', 'c', 'g', 'h', 'd', 'b', 'e', 'i', 'f']
Inorder: ['g', 'c', 'h', 'a', 'd', 'r', 'i', 'e', 'b', 'f']
Postorder: ['g', 'h', 'c', 'd', 'a', 'i', 'e', 'f', 'b', 'r']
```