class Node:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.left = None  
 self.right = None  
  
  
def inorder\_traversal(node):  
 return (inorder\_traversal(node.left) if node.left else []) + [node.value] + \  
 (inorder\_traversal(node.right) if node.right else [])  
  
  
def preorder\_traversal(node):  
 return [node.value] + (preorder\_traversal(node.left) if node.left else []) + \  
 (preorder\_traversal(node.right) if node.right else [])  
  
  
def postorder\_traversal(node):  
 return (postorder\_traversal(node.left) if node.left else []) + \  
 (postorder\_traversal(node.right) if node.right else []) + [node.value]  
  
  
# Part A: Binary Trees for Equations  
tree1 = Node('-')  
tree1.left = Node('\*')  
tree1.left.left = Node('3')  
tree1.left.right = Node('5')  
tree1.right = Node('+')  
tree1.right.left = Node('\*')  
tree1.right.left.left = Node('4')  
tree1.right.left.right = Node('5')  
tree1.right.right = Node('-')  
tree1.right.right.left = Node('6')  
tree1.right.right.right = Node('7')  
  
tree2 = Node('-')  
tree2.left = Node('\*')  
tree2.left.left = Node('+')  
tree2.left.left.left = Node('a')  
tree2.left.left.right = Node('b')  
tree2.left.right = Node('c')  
tree2.right = Node('-')  
tree2.right.left = Node('d')  
tree2.right.right = Node('e')  
  
tree3 = Node('+')  
tree3.left = Node('+')  
tree3.left.left = Node('^')  
tree3.left.left.left = Node('a')  
tree3.left.left.right = Node('b')  
tree3.left.right = Node('+')  
tree3.left.right.left = Node('c')  
tree3.left.right.right = Node('d')  
tree3.right = Node('/')  
tree3.right.left = Node('\*')  
tree3.right.left.left = Node('e')  
tree3.right.left.right = Node('f')  
tree3.right.right = Node('+')  
tree3.right.right.left = Node('g')  
tree3.right.right.right = Node('h')  
  
tree4 = Node('/')  
tree4.left = Node('+')  
tree4.left.left = Node('a')  
tree4.left.right = Node('b')  
tree4.right = Node('\*')  
tree4.right.left = Node('c')  
tree4.right.right = Node('-')  
tree4.right.right.left = Node('d')  
tree4.right.right.right = Node('^')  
tree4.right.right.right.left = Node('e')  
tree4.right.right.right.right = Node('f')  
  
tree5 = Node('\*')  
tree5.left = Node('+')  
tree5.left.left = Node('-')  
tree5.left.left.left = Node('a')  
tree5.left.left.right = Node('b')  
tree5.left.right = Node('c')  
tree5.right = Node('\*')  
tree5.right.left = Node('+')  
tree5.right.left.left = Node('d')  
tree5.right.left.right = Node('e')  
tree5.right.right = Node('/')  
tree5.right.right.left = Node('f')  
tree5.right.right.right = Node('g')  
  
tree6 = Node('\*')  
tree6.left = Node('/')  
tree6.left.left = Node('\*')  
tree6.left.left.left = Node('+')  
tree6.left.left.left.left = Node('5')  
tree6.left.left.left.right = Node('2')  
tree6.left.left.right = Node('-')  
tree6.left.left.right.left = Node('2')  
tree6.left.left.right.right = Node('1')  
tree6.left.right = Node('+')  
tree6.left.right.left = Node('+')  
tree6.left.right.left.left = Node('2')  
tree6.left.right.left.right = Node('9')  
tree6.left.right.right = Node('-')  
tree6.left.right.right.left = Node('-')  
tree6.left.right.right.left.left = Node('7')  
tree6.left.right.right.left.right = Node('2')  
tree6.left.right.right.right = Node('1')  
tree6.right = Node('8')  
  
  
# Part B: Binary Trees from Matrix Representations  
def create\_tree\_matrix\_1():  
 nodes = {ch: Node(ch) for ch in 'rabcdefgh'}  
 nodes['r'].left, nodes['r'].right = nodes['a'], nodes['b']  
 nodes['a'].left, nodes['a'].right = nodes['c'], nodes['d']  
 nodes['b'].left, nodes['b'].right = nodes['e'], nodes['f']  
 nodes['e'].left = nodes['g']  
 nodes['g'].left, nodes['g'].right = nodes['h'], None  
 return nodes['r']  
  
  
def create\_tree\_matrix\_2():  
 nodes = {ch: Node(ch) for ch in 'rabcdefg'}  
 nodes['r'].left, nodes['r'].right = nodes['a'], nodes['b']  
 nodes['a'].left, nodes['a'].right = nodes['c'], nodes['d']  
 nodes['b'].left, nodes['b'].right = nodes['e'], nodes['f']  
 nodes['e'].left = nodes['g']  
 return nodes['r']  
  
  
def create\_tree\_matrix\_3():  
 nodes = {ch: Node(ch) for ch in 'rabcdef'}  
 nodes['r'].left, nodes['r'].right = nodes['a'], nodes['b']  
 nodes['a'].left, nodes['a'].right = nodes['d'], None  
 nodes['b'].left, nodes['b'].right = nodes['c'], nodes['e']  
 nodes['d'].right = nodes['f']  
 return nodes['r']  
  
  
def create\_tree\_matrix\_4():  
 nodes = {ch: Node(ch) for ch in 'rabcdefghi'}  
 nodes['r'].left, nodes['r'].right = nodes['a'], nodes['b']  
 nodes['a'].left, nodes['a'].right = nodes['c'], nodes['d']  
 nodes['b'].left, nodes['b'].right = nodes['e'], nodes['f']  
 nodes['e'].left, nodes['e'].right = nodes['g'], nodes['h']  
 nodes['g'].right = nodes['i']  
 return nodes['r']  
  
  
def print\_traversals(tree, name):  
 print(f"{name} Traversals:")  
 print(f" Inorder: {inorder\_traversal(tree)}")  
 print(f" Preorder: {preorder\_traversal(tree)}")  
 print(f" Postorder: {postorder\_traversal(tree)}\n")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Part A: Equations to Binary Trees\n")  
 print\_traversals(tree1, "Equation 1: (3 \* 5) - ((4 \* 5) + (6 - 7))")  
 print\_traversals(tree2, "Equation 2: ((a + b) \* c) - (d - e)")  
 print\_traversals(tree3, "Equation 3: ((a ^ b) + (c + d)) + ((e \* f) / (g + h))")  
 print\_traversals(tree4, "Equation 4: (a + b) / (c \* (d - (e ^ f)))")  
 print\_traversals(tree5, "Equation 5: ((a - b) + c) \* ((d + e) \* (f / g))")  
 print\_traversals(tree6, "Equation 6: (((5 + 2) \* (2 - 1)) / ((2 + 9) + ((7 - 2) - 1))) \* 8")  
  
 print("Part B: Binary Trees from Matrix Representations\n")  
 tree\_matrix1 = create\_tree\_matrix\_1()  
 print\_traversals(tree\_matrix1, "Matrix 1")  
 tree\_matrix2 = create\_tree\_matrix\_2()  
 print\_traversals(tree\_matrix2, "Matrix 2")  
 tree\_matrix3 = create\_tree\_matrix\_3()  
 print\_traversals(tree\_matrix3, "Matrix 3")  
 tree\_matrix4 = create\_tree\_matrix\_4()  
 print\_traversals(tree\_matrix4, "Matrix 4")



