```
class Node:
    def __init__(self, name, children=None, value=None):
        self.name = name
        self.children = children if children is not None else []
        self.value = value

def display_tree(node, depth=0):
    indent = '  ' * depth
    if node.children:
        value_str = f"({node.value})" if node.value is not None else ''
        print(f"{indent}{node.name}{value_str}")
    else:
        print(f"{indent}{node.name}({node.value})")
    for child in node.children:
        display_tree(child, depth + 1)

def compute_min_max(node, is_max):
    if not node.children:  # leaf node
        return node.value

    if is_max:
        max_value = float('-inf')
        for child in node.children:
            child_value = compute_min_max(child, False)
            max_value = max(max_value, child_value)
        node.value = max_value
        return max_value
    else:
        min_value = float('inf')
        for child in node.children:
            child_value = compute_min_max(child, True)
            min_value = min(min_value, child_value)
        node.value = min_value
        return min_value

def get_optimal_path(node):
    path = [node]
    current = node
    while current.children:
        for child in current.children:
            if child.value == current.value:
                current = child
                path.append(current)
                break
    return path

# Create leaf nodes
L = Node('L', [], 2)
M = Node('M', [], 3)
N = Node('N', [], 5)
O = Node('O', [], 9)
P = Node('P', [], 0)
Q = Node('Q', [], 7)
R = Node('R', [], 4)
S = Node('S', [], 2)
T = Node('T', [], 1)
U = Node('U', [], 5)
V = Node('V', [], 6)

# Build parents
E = Node('E', [L, M], None)
F = Node('F', [N, O], None)
G = Node('G', [P], None)
H = Node('H', [Q, R], None)
I = Node('I', [S, T], None)
J = Node('J', [U, V], None)

# Build B, C, D
B = Node('B', [E, F], None)
C = Node('C', [G, H], None)
D = Node('D', [I, J], None)

# Root A
A = Node('A', [B, C, D], None)
```

```
# 1: Display the initial tree structure
print("Initial Tree Structure:")
display_tree(A)

# 2: Compute min-max with Max first
compute_min_max(A, is_max=True)
print("\nTree After Min-Max (Max First):")
display_tree(A)

# 3: Get and display the optimal path
optimal_path_max = get_optimal_path(A)
print("\nOptimal Path (Max First):", ' -> '.join(node.name for node in optimal_path_max))

# Reset internal node values for Min first
for node in [A, B, C, D, E, F, G, H, I, J]:
    node.value = None

# 4: Compute min-max with Min first
compute_min_max(A, is_max=False)
print("\nTree After Min-Max (Min First):")
display_tree(A)

# 5: Get and display the optimal path
optimal_path_min = get_optimal_path(A)
print("\nOptimal Path (Min First):", ' -> '.join(node.name for node in optimal_path_min))
```

Output:

```
Initial Tree Structure:

A

 B

  E

   L(2)

   M(3)
```

```
    F
      N(5)
      O(9)
  C
    G
      P(0)
    H
      Q(7)
      R(4)
  D
    I
      S(2)
      T(1)
    J
      U(5)
      V(6)

Tree After Min-Max (Max First):
A(3)
  B(3)
    E(3)
      L(2)
      M(3)
    F(9)
      N(5)
      O(9)
  C(0)
    G(0)
      P(0)
    H(7)
      Q(7)
      R(4)
  D(2)
    I(2)
      S(2)
      T(1)
    J(6)
      U(5)
      V(6)

Optimal Path (Max First): A -> B -> E -> M

Tree After Min-Max (Min First):
A(4)
  B(5)
    E(2)
      L(2)
      M(3)
    F(5)
      N(5)
      O(9)
  C(4)
  G(0)
   P(0)
  H(4)
   Q(7)
   R(4)
 D(5)
  I(1)
   S(2)
   T(1)
  J(5)
   U(5)
   V(6)
```

Optimal Path (Min First): A -> C -> H -> R