Practical 9

```
def minimax(node, depth, player, game_tree, values, path):


  if depth == 0 or node not in game_tree:
    return values[node], [node]


  if player == "MAX":
    best = float(-100000)
    move = []
    for child in game_tree[node]:
      val, child_path = minimax(child, depth - 1, "MIN", game_tree, values, path)
      if val > best:
        best = val
        move = [node] + child_path
    return best, move



  else:  # MIN player
    best = float(100000)
    move = []
    for child in game_tree[node]:
      val, child_path = minimax(child, depth - 1, "MAX", game_tree, values, path)
      if val < best:
        best = val
        move = [node] + child_path
    return best, move
```

```
if __name__ == "__main__":
    game_tree = {
        'A': ['B', 'C', 'D'],
        'B': ['E', 'F'],
        'C': ['G'],
        'D': ['H', 'I'],
        'E': ['J', 'K'],
        'F': ['L'],
        'G': ['M'],
        'H': ['N', 'O']

    }

    values = {
        'J': -6,
        'K': -1,
        'L': 2,
        'M': 9,
        'N': -9,
        'O': -8,
        'I': 9
    }
```

```
result, path = minimax('A', 3, "MAX", game_tree, values, [])
print("Optimal value of the tree is:", result)
print("Decision path followed:", " -> ".join(path))
```