LAB CIE

MIN MAX ALGORITHM

CODE:

```
import math

def minimax(node, depth, maximizing_player, tree, path):

    current_path = path + [node]
    print(f"Visiting node {node} at depth {depth}. Path:
{current_path}")

    if depth == 0 or node not in tree or not tree[node]:
        print(f"Reached a leaf node or specified depth at node {node}.
Value: {node}")
        return node

    if maximizing_player:
        print(f"Maximizing player's turn at node {node}")
        max_eval = -math.inf
        for child in tree[node]:
            print(f"Exploring child {child} of node {node}")
            eval = minimax(child, depth - 1, False, tree, current_path)
            max_eval = max(max_eval, eval)
            print(f"Max evaluation at node {node} after considering
child {child}: {max_eval}")
        print(f"Optimal value for node {node} (maximizing player):
{max_eval}")
        return max_eval
    else:
        print(f"Minimizing player's turn at node {node}")
        min_eval = math.inf
        for child in tree[node]:
            print(f"Exploring child {child} of node {node}")
            eval = minimax(child, depth - 1, True, tree, current_path)
            min_eval = min(min_eval, eval)
            print(f"Min evaluation at node {node} after considering
child {child}: {min_eval}")
        print(f"Optimal value for node {node} (minimizing player):
{min_eval}")
        return min_eval


print("Enter the game tree as a dictionary where keys are parent nodes
and values are lists of child nodes.")
print("For leaf nodes, the value should be an empty list or the node
value itself (as an integer).")
```

```
print("Example: {'A': ['B', 'C'], 'B': [3, 12], 'C': [8, 2]}")

tree_input = input("Enter the game tree: ")
try:
    game_tree = eval(tree_input)
except Exception as e:
    print(f"Invalid input: {e}")
    game_tree = {}

for node, children in game_tree.items():
    if isinstance(children, list):
        game_tree[node] = [int(child) if isinstance(child, (str, int))
and child not in game_tree else child for child in children]


start_node = input("Enter the starting node: ")
depth_input = input("Enter the maximum depth to search: ")
try:
    max_depth = int(depth_input)
except ValueError:
    print("Invalid depth. Using default depth of 3.")
    max_depth = 3

print("\n--- Minimax Algorithm Steps ---")
optimal_value = minimax(start_node, max_depth, True, game_tree, [])
print("\n--- Result ---")
print(f"The optimal value for the starting node {start_node} is:
{optimal_value}")
```

OUTPUT:

```
Enter the game tree as a dictionary where keys are parent nodes and values are lists of child nodes.
For leaf nodes, the value should be an empty list or the node value itself (as an integer).
Example: {'A': ['B', 'C'], 'B': [3, 12], 'C': [8, 2]}
Enter the game tree: {'A' : ['B','C'],'B':['D','E'],'C':['F','G'],'D':['H','I'],'E':['J','K'],'F':['L','M'],'G':['N','O','P'],'H':[8,7],'I':[9,2,11],'J':[8],'K':[10,3],'L':[12,4,6],
Enter the starting node: A
Enter the maximum depth to search: 5

--- Minimax Algorithm Steps ---
Visiting node A at depth 5. Path: ['A']
Maximizing player's turn at node A
Exploring child B of node A
Visiting node B at depth 4. Path: ['A', 'B']
Minimizing player's turn at node B
Exploring child D of node B
Visiting node D at depth 3. Path: ['A', 'B', 'D']
Maximizing player's turn at node D
Exploring child H of node D
Visiting node H at depth 2. Path: ['A', 'B', 'D', 'H']
Minimizing player's turn at node H
Exploring child 8 of node H
Visiting node 8 at depth 1. Path: ['A', 'B', 'D', 'H', 8]
Reached a leaf node or specified depth at node 8. Value: 8
Min evaluation at node H after considering child 8: 8
Exploring child 7 of node H
Visiting node 7 at depth 1. Path: ['A', 'B', 'D', 'H', 7]
Reached a leaf node or specified depth at node 7. Value: 7
Min evaluation at node H after considering child 7: 7
Optimal value for node H (minimizing player): 7
Max evaluation at node D after considering child H: 7
Exploring child I of node D
Visiting node I at depth 2. Path: ['A', 'B', 'D', 'I']
Minimizing player's turn at node I
Exploring child 9 of node I
Visiting node 9 at depth 1. Path: ['A', 'B', 'D', 'I', 9]
Reached a leaf node or specified depth at node 9. Value: 9
Min evaluation at node I after considering child 9: 9
Exploring child 2 of node I
Visiting node 2 at depth 1. Path: ['A', 'B', 'D', 'I', 2]
Reached a leaf node or specified depth at node 2. Value: 2
Min evaluation at node I after considering child 2: 2
Exploring child 11 of node I
Visiting node 11 at depth 1. Path: ['A', 'B', 'D', 'I', 11]
Reached a leaf node or specified depth at node 11. Value: 11
Min evaluation at node I after considering child 11: 2
Optimal value for node I (minimizing player): 2
```

```
Optimal value for node D (maximizing player): 7
Min evaluation at node B after considering child D: 7
Exploring child E of node B
Visiting node E at depth 3. Path: ['A', 'B', 'E']
Maximizing player's turn at node E
Exploring child J of node E
Visiting node J at depth 2. Path: ['A', 'B', 'E', 'J']
Minimizing player's turn at node J
Exploring child 8 of node J
Visiting node 8 at depth 1. Path: ['A', 'B', 'E', 'J', 8]
Reached a leaf node or specified depth at node 8. Value: 8
Min evaluation at node J after considering child 8: 8
Optimal value for node J (minimizing player): 8
Max evaluation at node E after considering child J: 8
Exploring child K of node E
Visiting node K at depth 2. Path: ['A', 'B', 'E', 'K']
Minimizing player's turn at node K
Exploring child 10 of node K
Visiting node 10 at depth 1. Path: ['A', 'B', 'E', 'K', 10]
Reached a leaf node or specified depth at node 10. Value: 10
Min evaluation at node K after considering child 10: 10
Exploring child 3 of node K
Visiting node 3 at depth 1. Path: ['A', 'B', 'E', 'K', 3]
Reached a leaf node or specified depth at node 3. Value: 3
Min evaluation at node K after considering child 3: 3
Optimal value for node K (minimizing player): 3
Max evaluation at node E after considering child K: 8
Optimal value for node E (maximizing player): 8
Min evaluation at node B after considering child E: 7
Optimal value for node B (minimizing player): 7
Max evaluation at node A after considering child B: 7
Exploring child C of node A
Visiting node C at depth 4. Path: ['A', 'C']
Minimizing player's turn at node C
Exploring child F of node C
Visiting node F at depth 3. Path: ['A', 'C', 'F']
Maximizing player's turn at node F
Exploring child L of node F
Visiting node L at depth 2. Path: ['A', 'C', 'F', 'L']
Minimizing player's turn at node L
Exploring child 12 of node L
Visiting node 12 at depth 1. Path: ['A', 'C', 'F', 'L', 12]
Reached a leaf node or specified depth at node 12. Value: 12
Min evaluation at node L after considering child 12: 12
Exploring child 4 of node L
```

```
Min evaluation at node L after considering child 12: 12
Exploring child 4 of node L
Visiting node 4 at depth 1. Path: ['A', 'C', 'F', 'L', 4]
Reached a leaf node or specified depth at node 4. Value: 4
Min evaluation at node L after considering child 4: 4
Exploring child 6 of node L
Visiting node 6 at depth 1. Path: ['A', 'C', 'F', 'L', 6]
Reached a leaf node or specified depth at node 6. Value: 6
Min evaluation at node L after considering child 6: 4
Optimal value for node L (minimizing player): 4
Max evaluation at node F after considering child L: 4
Exploring child M of node F
Visiting node M at depth 2. Path: ['A', 'C', 'F', 'M']
Minimizing player's turn at node M
Exploring child 9 of node M
Visiting node 9 at depth 1. Path: ['A', 'C', 'F', 'M', 9]
Reached a leaf node or specified depth at node 9. Value: 9
Min evaluation at node M after considering child 9: 9
Optimal value for node M (minimizing player): 9
Max evaluation at node F after considering child M: 9
Optimal value for node F (maximizing player): 9
Min evaluation at node C after considering child F: 9
Exploring child G of node C
Visiting node G at depth 3. Path: ['A', 'C', 'G']
Maximizing player's turn at node G
Exploring child N of node G
Visiting node N at depth 2. Path: ['A', 'C', 'G', 'N']
Minimizing player's turn at node N
Exploring child 6 of node N
Visiting node 6 at depth 1. Path: ['A', 'C', 'G', 'N', 6]
Reached a leaf node or specified depth at node 6. Value: 6
Min evaluation at node N after considering child 6: 6
Optimal value for node N (minimizing player): 6
Max evaluation at node G after considering child N: 6
Exploring child O of node G
Visiting node O at depth 2. Path: ['A', 'C', 'G', 'O']
Minimizing player's turn at node O
Exploring child 14 of node O
Visiting node 14 at depth 1. Path: ['A', 'C', 'G', 'O', 14]
Reached a leaf node or specified depth at node 14. Value: 14
Min evaluation at node O after considering child 14: 14
Exploring child 12 of node O
Visiting node 12 at depth 1. Path: ['A', 'C', 'G', 'O', 12]
Reached a leaf node or specified depth at node 12. Value: 12
```

```
Visiting node 12 at depth 1. Path: ['A', 'C', 'G', 'O', 12]
Reached a leaf node or specified depth at node 12. Value: 12
Min evaluation at node O after considering child 12: 12
Optimal value for node O (minimizing player): 12
Max evaluation at node G after considering child O: 12
Exploring child P of node G
Visiting node P at depth 2. Path: ['A', 'C', 'G', 'P']
Minimizing player's turn at node P
Exploring child 20 of node P
Visiting node 20 at depth 1. Path: ['A', 'C', 'G', 'P', 20]
Reached a leaf node or specified depth at node 20. Value: 20
Min evaluation at node P after considering child 20: 20
Exploring child 10 of node P
Visiting node 10 at depth 1. Path: ['A', 'C', 'G', 'P', 10]
Reached a leaf node or specified depth at node 10. Value: 10
Min evaluation at node P after considering child 10: 10
Exploring child 2 of node P
Visiting node 2 at depth 1. Path: ['A', 'C', 'G', 'P', 2]
Reached a leaf node or specified depth at node 2. Value: 2
Min evaluation at node P after considering child 2: 2
Optimal value for node P (minimizing player): 2
Max evaluation at node G after considering child P: 12
Optimal value for node G (maximizing player): 12
Min evaluation at node C after considering child G: 9
Optimal value for node C (minimizing player): 9
Max evaluation at node A after considering child C: 9
Optimal value for node A (maximizing player): 9

--- Result ---
The optimal value for the starting node A is: 9
```

ALPHA BETA PRUNING

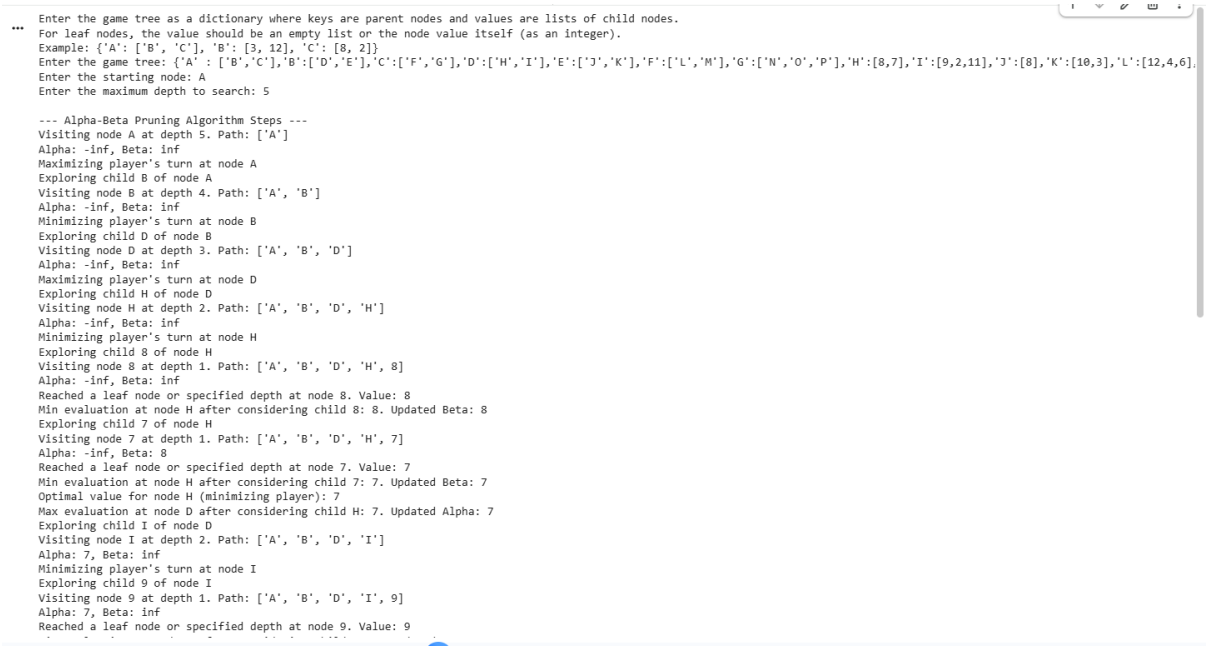
CODE:

```
import math

def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player,
tree, path):
    current_path = path + [node]
    print(f"Visiting node {node} at depth {depth}. Path:
{current_path}")
    print(f"Alpha: {alpha}, Beta: {beta}")

    if depth == 0 or node not in tree or not tree[node]:
        print(f"Reached a leaf node or specified depth at node {node}.
Value: {node}")
        return node

    if maximizing_player:
        print(f"Maximizing player's turn at node {node}")
        max_eval = -math.inf
        for child in tree[node]:
            print(f"Exploring child {child} of node {node}")
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta,
False, tree, current_path)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            print(f"Max evaluation at node {node} after considering
child {child}: {max_eval}. Updated Alpha: {alpha}")
```

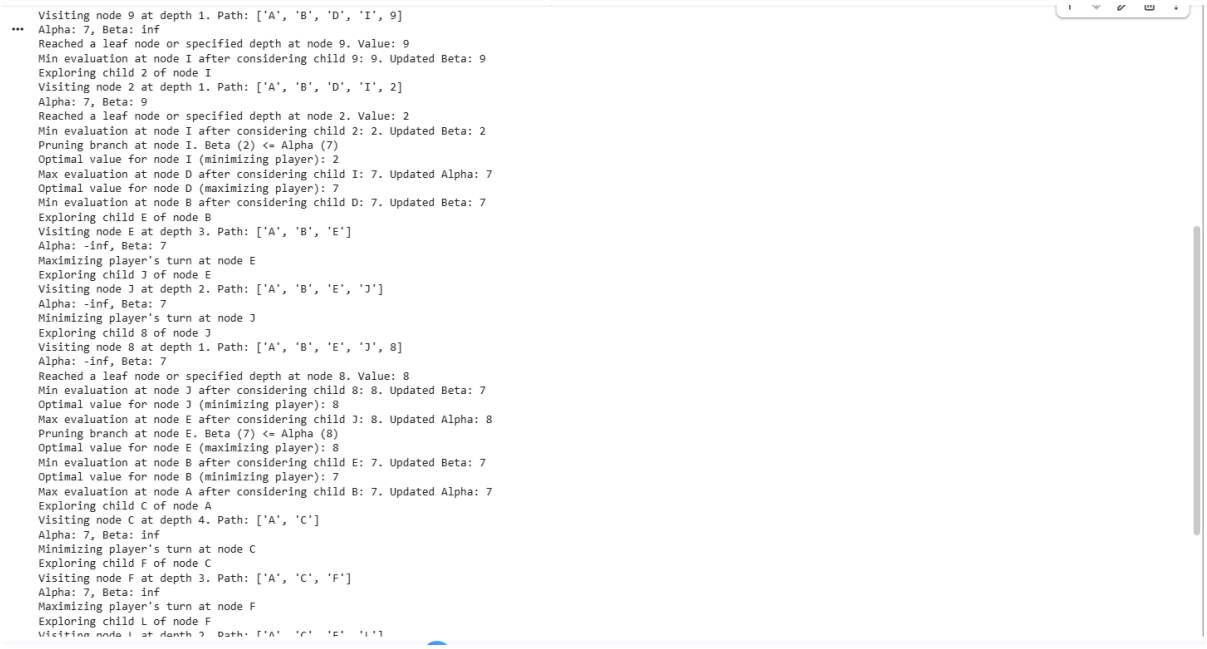
```
            if beta <= alpha:
                print(f"Pruning branch at node {node}. Beta ({beta}) <=
Alpha ({alpha})")
                break
        print(f"Optimal value for node {node} (maximizing player):
{max_eval}")
        return max_eval
    else:
        print(f"Minimizing player's turn at node {node}")
        min_eval = math.inf
        for child in tree[node]:
            print(f"Exploring child {child} of node {node}")
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta,
True, tree, current_path)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            print(f"Min evaluation at node {node} after considering
child {child}: {min_eval}. Updated Beta: {beta}")
            if beta <= alpha:
                print(f"Pruning branch at node {node}. Beta ({beta}) <=
Alpha ({alpha})")
                break
        print(f"Optimal value for node {node} (minimizing player):
{min_eval}")
        return min_eval

print("Enter the game tree as a dictionary where keys are parent nodes
and values are lists of child nodes.")
print("For leaf nodes, the value should be an empty list or the node
value itself (as an integer).")
print("Example: {'A': ['B', 'C'], 'B': [3, 12], 'C': [8, 2]}")

tree_input = input("Enter the game tree: ")
try:
    game_tree = eval(tree_input)
except Exception as e:
    print(f"Invalid input: {e}")
    game_tree = {}

for node, children in game_tree.items():
    if isinstance(children, list):
        game_tree[node] = [int(child) if isinstance(child, (str, int))
and child not in game_tree else child for child in children]

start_node = input("Enter the starting node: ")
depth_input = input("Enter the maximum depth to search: ")
try:
    max_depth = int(depth_input)
```

```
except ValueError:
    print("Invalid depth. Using default depth of 3.")
    max_depth = 3

print("\n--- Alpha-Beta Pruning Algorithm Steps ---")
optimal_value = alpha_beta_pruning(start_node, max_depth, -math.inf,
math.inf, True, game_tree, [])
print("\n--- Result ---")
print(f"The optimal value for the starting node {start_node} is:
{optimal_value}")
```

OUTPUT:



```
Enter the game tree as a dictionary where keys are parent nodes and values are lists of child nodes.
For leaf nodes, the value should be an empty list or the node value itself (as an integer).
Example: {'A': ['B', 'C'], 'B': [3, 12], 'C': [8, 2]}
Enter the game tree: {'A' : ['B','C'],'B':['D','E'],'C':['F','G'],'D':['H','I'],'E':['J','K'],'F':['L','M'],'G':['N','O','P'],'H':[8,7],'I':[9,2,11],'J':[8],'K':[10,3],'L':[12,4,6],
Enter the starting node: A
Enter the maximum depth to search: 5

--- Alpha-Beta Pruning Algorithm Steps ---
Visiting node A at depth 5. Path: ['A']
Alpha: -inf, Beta: inf
Maximizing player's turn at node A
Exploring child B of node A
Visiting node B at depth 4. Path: ['A', 'B']
Alpha: -inf, Beta: inf
Minimizing player's turn at node B
Exploring child D of node B
Visiting node D at depth 3. Path: ['A', 'B', 'D']
Alpha: -inf, Beta: inf
Maximizing player's turn at node D
Exploring child H of node D
Visiting node H at depth 2. Path: ['A', 'B', 'D', 'H']
Alpha: -inf, Beta: inf
Minimizing player's turn at node H
Exploring child 8 of node H
Visiting node 8 at depth 1. Path: ['A', 'B', 'D', 'H', 8]
Alpha: -inf, Beta: inf
Reached a leaf node or specified depth at node 8. Value: 8
Min evaluation at node H after considering child 8: 8. Updated Beta: 8
Exploring child 7 of node H
Visiting node 7 at depth 1. Path: ['A', 'B', 'D', 'H', 7]
Alpha: -inf, Beta: 8
Reached a leaf node or specified depth at node 7. Value: 7
Min evaluation at node H after considering child 7: 7. Updated Beta: 7
Optimal value for node H (minimizing player): 7
Max evaluation at node D after considering child H: 7. Updated Alpha: 7
Exploring child I of node D
Visiting node I at depth 2. Path: ['A', 'B', 'D', 'I']
Alpha: 7, Beta: inf
Minimizing player's turn at node I
Exploring child 9 of node I
Visiting node 9 at depth 1. Path: ['A', 'B', 'D', 'I', 9]
Alpha: 7, Beta: inf
Reached a leaf node or specified depth at node 9. Value: 9
```



```
Visiting node 9 at depth 1. Path: ['A', 'B', 'D', 'I', 9]
Alpha: 7, Beta: inf
Reached a leaf node or specified depth at node 9. Value: 9
Min evaluation at node I after considering child 9: 9. Updated Beta: 9
Exploring child 2 of node I
Visiting node 2 at depth 1. Path: ['A', 'B', 'D', 'I', 2]
Alpha: 7, Beta: 9
Reached a leaf node or specified depth at node 2. Value: 2
Min evaluation at node I after considering child 2: 2. Updated Beta: 2
Pruning branch at node I. Beta (2) <= Alpha (7)
Optimal value for node I (minimizing player): 2
Max evaluation at node D after considering child I: 7. Updated Alpha: 7
Optimal value for node D (maximizing player): 7
Min evaluation at node B after considering child D: 7. Updated Beta: 7
Exploring child E of node B
Visiting node E at depth 3. Path: ['A', 'B', 'E']
Alpha: -inf, Beta: 7
Maximizing player's turn at node E
Exploring child J of node E
Visiting node J at depth 2. Path: ['A', 'B', 'E', 'J']
Alpha: -inf, Beta: 7
Minimizing player's turn at node J
Exploring child 8 of node J
Visiting node 8 at depth 1. Path: ['A', 'B', 'E', 'J', 8]
Alpha: -inf, Beta: 7
Reached a leaf node or specified depth at node 8. Value: 8
Min evaluation at node J after considering child 8: 8. Updated Beta: 7
Optimal value for node J (minimizing player): 8
Max evaluation at node E after considering child J: 8. Updated Alpha: 8
Pruning branch at node E. Beta (7) <= Alpha (8)
Optimal value for node E (maximizing player): 8
Min evaluation at node B after considering child E: 7. Updated Beta: 7
Optimal value for node B (minimizing player): 7
Max evaluation at node A after considering child B: 7. Updated Alpha: 7
Exploring child C of node A
Visiting node C at depth 4. Path: ['A', 'C']
Alpha: 7, Beta: inf
Minimizing player's turn at node C
Exploring child F of node C
Visiting node F at depth 3. Path: ['A', 'C', 'F']
Alpha: 7, Beta: inf
Maximizing player's turn at node F
Exploring child L of node F
```

```
Alpha: 7, Beta: inf
Maximizing player's turn at node F
Exploring child L of node F
Visiting node L at depth 2. Path: ['A', 'C', 'F', 'L']
Alpha: 7, Beta: inf
Minimizing player's turn at node L
Exploring child 12 of node L
Visiting node 12 at depth 1. Path: ['A', 'C', 'F', 'L', 12]
Alpha: 7, Beta: inf
Reached a leaf node or specified depth at node 12. Value: 12
Min evaluation at node L after considering child 12: 12. Updated Beta: 12
Exploring child 4 of node L
Visiting node 4 at depth 1. Path: ['A', 'C', 'F', 'L', 4]
Alpha: 7, Beta: 12
Reached a leaf node or specified depth at node 4. Value: 4
Min evaluation at node L after considering child 4: 4. Updated Beta: 4
Pruning branch at node L. Beta (4) <= Alpha (7)
Optimal value for node L (minimizing player): 4
Max evaluation at node F after considering child L: 4. Updated Alpha: 7
Exploring child M of node F
Visiting node M at depth 2. Path: ['A', 'C', 'F', 'M']
Alpha: 7, Beta: inf
Minimizing player's turn at node M
Exploring child 9 of node M
Visiting node 9 at depth 1. Path: ['A', 'C', 'F', 'M', 9]
Alpha: 7, Beta: inf
Reached a leaf node or specified depth at node 9. Value: 9
Min evaluation at node M after considering child 9: 9. Updated Beta: 9
Optimal value for node M (minimizing player): 9
Max evaluation at node F after considering child M: 9. Updated Alpha: 9
Optimal value for node F (maximizing player): 9
Min evaluation at node C after considering child F: 9. Updated Beta: 9
Exploring child G of node C
Visiting node G at depth 3. Path: ['A', 'C', 'G']
Alpha: 7, Beta: 9
Maximizing player's turn at node G
Exploring child N of node G
Visiting node N at depth 2. Path: ['A', 'C', 'G', 'N']
Alpha: 7, Beta: 9
Minimizing player's turn at node N
Exploring child 6 of node N
Visiting node 6 at depth 1. Path: ['A', 'C', 'G', 'N', 6]
Alpha: 7, Beta: 9
Reached a leaf node or specified depth at node 6. Value: 6
```

```
Alpha: 7, Beta: 9
Minimizing player's turn at node N
Exploring child 6 of node N
Visiting node 6 at depth 1. Path: ['A', 'C', 'G', 'N', 6]
Alpha: 7, Beta: 9
Reached a leaf node or specified depth at node 6. Value: 6
Min evaluation at node N after considering child 6: 6. Updated Beta: 6
Pruning branch at node N. Beta (6) <= Alpha (7)
Optimal value for node N (minimizing player): 6
Max evaluation at node G after considering child N: 6. Updated Alpha: 7
Exploring child O of node G
Visiting node O at depth 2. Path: ['A', 'C', 'G', 'O']
Alpha: 7, Beta: 9
Minimizing player's turn at node O
Exploring child 14 of node O
Visiting node 14 at depth 1. Path: ['A', 'C', 'G', 'O', 14]
Alpha: 7, Beta: 9
Reached a leaf node or specified depth at node 14. Value: 14
Min evaluation at node O after considering child 14: 14. Updated Beta: 9
Exploring child 12 of node O
Visiting node 12 at depth 1. Path: ['A', 'C', 'G', 'O', 12]
Alpha: 7, Beta: 9
Reached a leaf node or specified depth at node 12. Value: 12
Min evaluation at node O after considering child 12: 12. Updated Beta: 9
Optimal value for node O (minimizing player): 12
Max evaluation at node G after considering child O: 12. Updated Alpha: 12
Pruning branch at node G. Beta (9) <= Alpha (12)
Optimal value for node G (maximizing player): 12
Min evaluation at node C after considering child G: 9. Updated Beta: 9
Optimal value for node C (minimizing player): 9
Max evaluation at node A after considering child C: 9. Updated Alpha: 9
Optimal value for node A (maximizing player): 9

--- Result ---
The optimal value for the starting node A is: 9
```