```
import math

def minimax (curDepth, nodeIndex,
             maxTurn, scores,
             targetDepth):

    # base case : targetDepth reached
    if (curDepth == targetDepth):
        return scores[nodeIndex]

    if (maxTurn):
        return max(minimax(curDepth + 1, nodeIndex * 2,
                    False, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1,
                    False, scores, targetDepth))

    else:
        return min(minimax(curDepth + 1, nodeIndex * 2,
                     True, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1,
                     True, scores, targetDepth))

# Driver code
scores = [-1,4,2,6,-3,-5,0,7]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores, treeDepth))
```

```
The optimal value is : 4
```

```
tree = {
    'A': {'B': {'D': {'H': -1, 'I': 4}, 'E': {'J': 2, 'K': 6}},
          'C': {'F': {'L': -3, 'M': -5}, 'G': {'N': 0, 'O': 7}}
         }
}

def alpha_beta(node, alpha, beta, maximizing_player=True):
    if type(node) is not dict:
        return node
    if maximizing_player:
        max_val = float('-inf')
        for child in node.values():
            value = alpha_beta(child, alpha, beta, False)
            max_val = max(max_val, value)
            alpha = max(alpha, value)
            if beta <= alpha:
                break
        return max_val
```

```python
    else:
        min_val = float('inf')
        for child in node.values():
            value = alpha_beta(child, alpha, beta, True)
            min_val = min(min_val, value)
            beta = min(beta, value)
            if beta <= alpha:
                break
        return min_val

result = alpha_beta(tree['A'], float('-inf'), float('inf'))
print(f"The value of the root node A after applying Alpha-Beta pruning
is {result}")
```

```
The value of the root node A after applying Alpha-Beta pruning is 4
```