Implementation of Alpha-Beta Pruning

```
class MinMaxPruning:

def __init__(self):

self.tree = []

self.pruned_nodes = []


def take_input(self):

"""

Takes user input for the tree structure and its values.

Example Input:

Enter depth of the tree: 3

Enter leaf values at depth 3 (comma-separated): 3, 5, 6, 9, 1, 2, 0, -1

"""

self.depth = int(input("Enter depth of the tree: "))

print(f"Tree is assumed to be a full binary tree of depth {self.depth}.")


# Input leaf node values

leaf_values = list(map(int, input(f"Enter leaf values at depth {self.depth}
(comma-separated): ").split(",")))

self.tree = leaf_values

print("\nInput Tree (Leaf Nodes):", self.tree)


def min_max(self, depth, node_index, maximizing_player, alpha, beta):

"""

Performs Min-Max pruning and calculates the optimal value.
```

```
:param depth: Current depth in the tree
:param node_index: Index of the current node
:param maximizing_player: Boolean indicating whether it's Max's turn
:param alpha: Alpha value for pruning
:param beta: Beta value for pruning
:return: Optimal value for the current subtree
"""
if depth == self.depth:
return self.tree[node_index]

if maximizing_player:
best = float("-inf")
for i in range(2):
value = self.min_max(depth + 1, 2 * node_index + i, False, alpha, beta)
best = max(best, value)
alpha = max(alpha, best)
if beta <= alpha:
self.pruned_nodes.append((depth, node_index, "MAX"))
break  # Beta cutoff
return best
else:
best = float("inf")
for i in range(2):
value = self.min_max(depth + 1, 2 * node_index + i, True, alpha, beta)
best = min(best, value)
```

```
beta = min(beta, best)
if beta <= alpha:
self.pruned_nodes.append((depth, node_index, "MIN"))
break  # Alpha cutoff
return best

def display_tree(self):
"""
Display the pruned tree structure and the pruned nodes.
"""
print("\nPruned Nodes (Depth, Node Index, Type):", self.pruned_nodes)
print("Pruned nodes represent subtrees that were skipped due to pruning.")

def run(self):
"""
Driver function to run the Min-Max pruning.
"""
self.take_input()
print("\nRunning Min-Max Pruning...")
optimal_value = self.min_max(0, 0, True, float("-inf"), float("inf"))
print("\nOptimal Value (Root Node):", optimal_value)
self.display_tree()
if __name__ == "__main__":
pruning = MinMaxPruning()
pruning.run()
```

Output:

Enter depth of the tree: 3

Tree is assumed to be a full binary tree of depth 3.

Enter leaf values at depth 3 (comma-separated): -1,8,-3,-1,2,1,3,4

Input Tree (Leaf Nodes): [-1, 8, -3, -1, 2, 1, 3, 4]

Running Min-Max Pruning...

Optimal Value (Root Node): 2

Pruned Nodes (Depth, Node Index, Type): [(2, 3, 'MAX')]

Pruned nodes represent subtrees that were skipped due to pruning.