In [1]:

```
import math

# Define a simple game tree for demonstration purposes
class GameState:
    def __init__(self, value=None, children=None):
        self.value = value
        self.children = children or []

# Alpha-beta pruning implementation
def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or not node.children:
        return node.value

    if maximizing_player:
        max_eval = -math.inf
        for child in node.children:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # Beta cutoff
        return max_eval
    else:
        min_eval = math.inf
        for child in node.children:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # Alpha cutoff
        return min_eval

# Example usage with a simple game tree
if __name__ == "__main__":
    # Create a sample game tree
    leaf_nodes = [GameState(value) for value in [3, 5, 6, 9, 1, 2, 0, -1]]
    level_2_nodes = [GameState(None, [leaf_nodes[i], leaf_nodes[i+1]]) for
    level_1_nodes = [GameState(None, [level_2_nodes[i], level_2_nodes[i+1]]
    root = GameState(None, level_1_nodes)

    # Perform alpha-beta pruning
    result = alpha_beta_pruning(root, 3, -math.inf, math.inf, True)
    print(f"The result of alpha-beta pruning is: {result}")
```

```
The result of alpha-beta pruning is: 5
```

In [ ]: