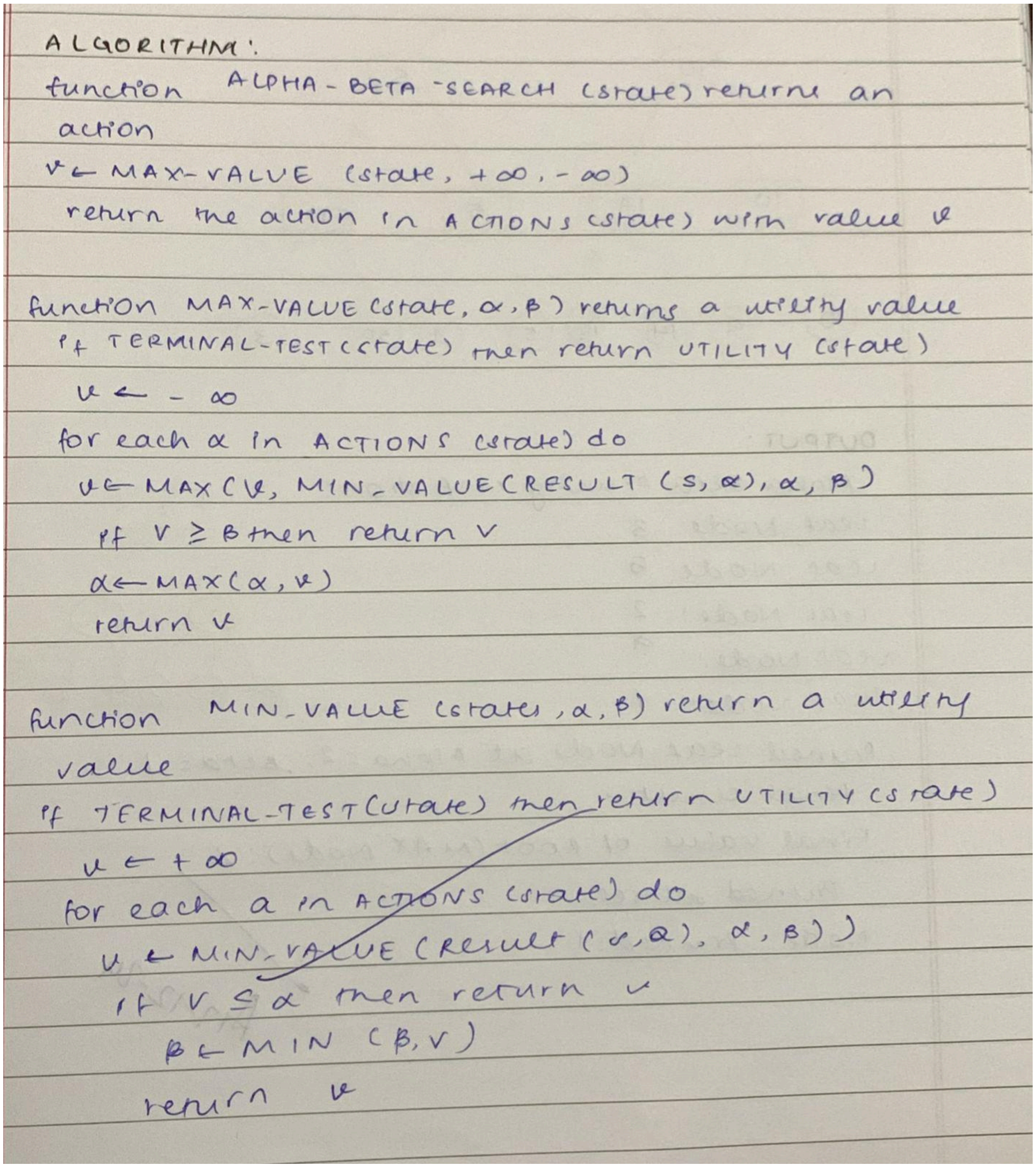
17-12-2024 WEEK - 9

ALPHA BETA PRUNING

ALGORITHM-



```
ALGORITHM:
function ALPHA-BETA-SEARCH (state) returns an
 action
v ← MAX-VALUE (state, +∞, -∞)
  return the action in ACTIONS (state) with value v

function MAX-VALUE (state, α, β) returns a utility value
  if TERMINAL-TEST (state) then return UTILITY (state)
   v ← - ∞
  for each α in ACTIONS (state) do
   v ← MAX (v, MIN-VALUE (RESULT (s, α), α, β)
     if v ≥ β then return v
    α ← MAX (α, v)
    return v

function MIN-VALUE (states, α, β) return a utility
 value
if TERMINAL-TEST (state) then return UTILITY (state)
  v ← +∞
  for each a in ACTIONS (state) do
   v ← MIN-VALUE (Result (s, a), α, β) )
     if v ≤ α then return v
       β ← MIN (β, v)
     return v
```

CODE-

```
class Node:
    def __init__(self, value=None, children=None):
        self.value = value  # Utility value at terminal nodes (leaves)
        self.children = children or []  # Children nodes (branches)
        self.is_pruned = False  # Flag to identify if a node was pruned

def alpha_beta_pruning(node, depth, alpha, beta, is_maximizing, path, pruned_paths):
       if not node.children or depth == 0:
        print(f"Leaf Node: Value = {node.value}")
        return node.value

    if is_maximizing:
        max_eval = float('-inf')
        for child in node.children:
            path.append(child)
            eval_value = alpha_beta_pruning(child, depth - 1, alpha, beta, False, path, pruned_paths)
            max_eval = max(max_eval, eval_value)
            alpha = max(alpha, eval_value)
            path.pop()

            # Pruning condition
            if beta <= alpha:
                print(f"Pruned Node at Alpha = {alpha}, Beta = {beta}")
                pruned_paths.append(child)
                child.is_pruned = True
                break
        node.value = max_eval  # Update value of node
        return max_eval

    # MIN node
    else:
        min_eval = float('inf')
        for child in node.children:
            path.append(child)
            eval_value = alpha_beta_pruning(child, depth - 1, alpha, beta, True, path, pruned_paths)
            min_eval = min(min_eval, eval_value)
            beta = min(beta, eval_value)
            path.pop()
            # Pruning condition
            if beta <= alpha:
                print(f"Pruned Node at Alpha = {alpha}, Beta = {beta}")
                pruned_paths.append(child)
                child.is_pruned = True
                break
        node.value = min_eval
        return min_eval

def display_results(root):
    print("\nFinal Results:")
    print(f"Final Value of Root (MAX Node): {root.value}")
    print("Pruned Subtrees:")
    def print_pruned(node):
        if node.is_pruned:
            print(f"Node Pruned: {node.value if node.value else 'Subtree'}")
        for child in node.children:
            print_pruned(child)
    print_pruned(root)

root = Node(children=[
    Node(children=[Node(value=3), Node(value=5), Node(value=2)]),
    Node(children=[Node(value=9), Node(value=1), Node(value=7)])
])

# Run Alpha-Beta Pruning
depth = 2  # Define maximum depth for tree search
alpha = float('-inf')
beta = float('inf')
pruned_paths = []

print("Alpha-Beta Pruning in Action:\n")
final_value = alpha_beta_pruning(root, depth, alpha, beta, True, [], pruned_paths)
display_results(root)
```

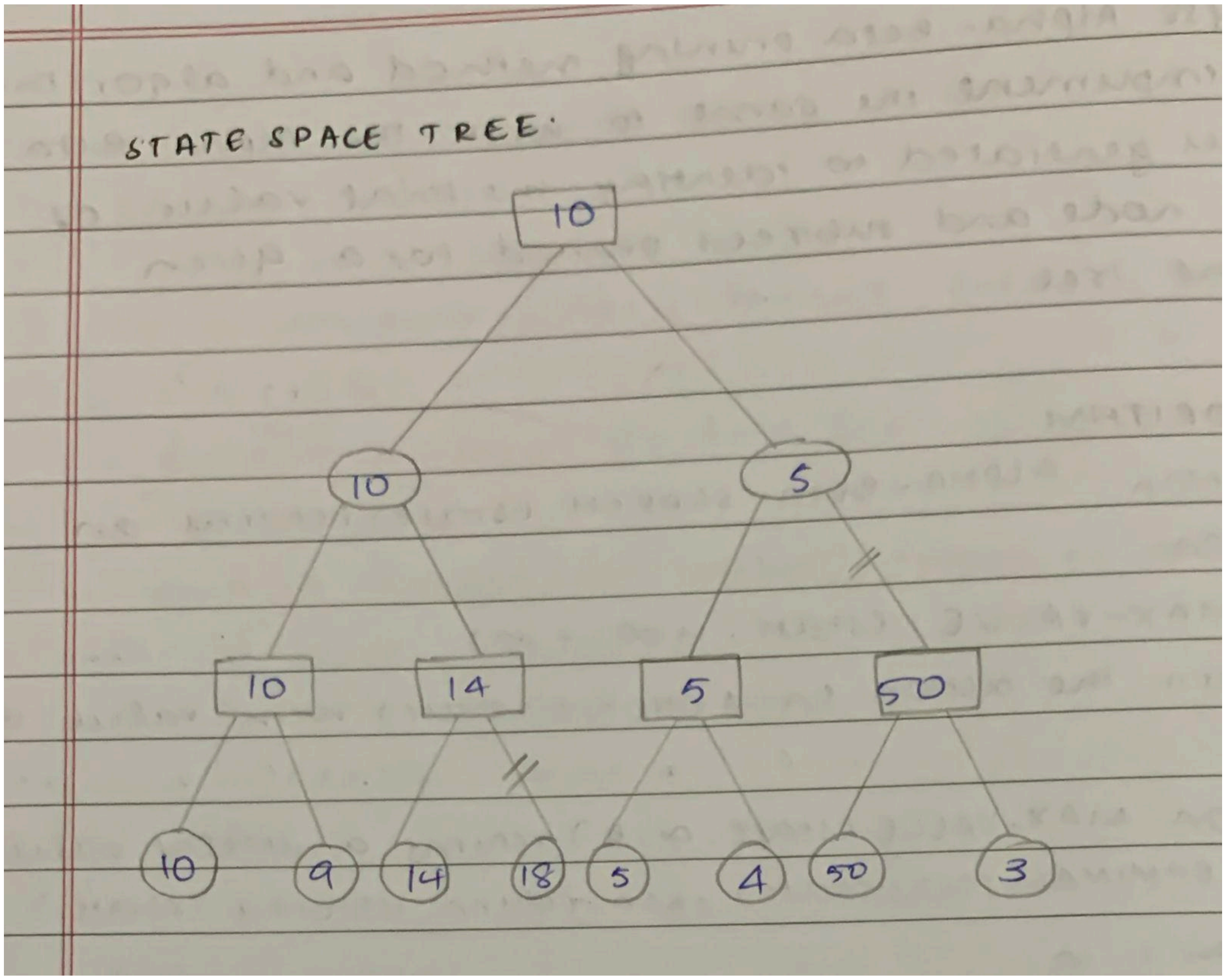
OUTPUT-

```
Alpha-Beta Pruning in Action:

Leaf Node: Value = 3
Leaf Node: Value = 5
Leaf Node: Value = 2
Leaf Node: Value = 9
Leaf Node: Value = 1
Pruned Node at Alpha = 2, Beta = 1

Final Results:
Final Value of Root (MAX Node): 2
Pruned Subtrees:
Node Pruned: 1
```

STATE SPACE TREE-


STATE SPACE TREE.
10
10
5
10
14
5
50
10
9
14
18
5
4
50
3