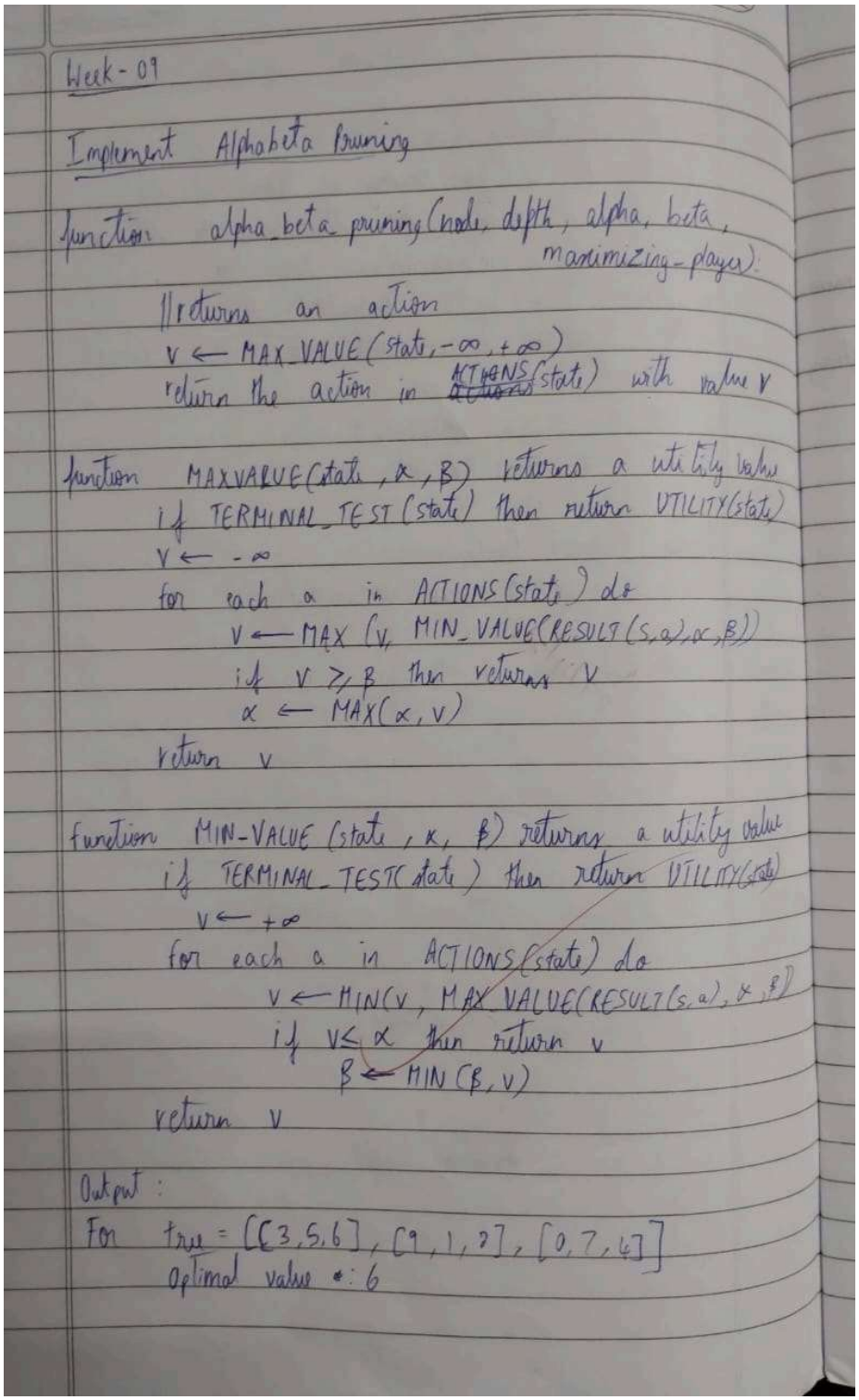
AlphaBeta Pruning Algorithm:



Week - 09

Implement Alphabeta Pruning

function alpha_beta_pruning(node, depth, alpha, beta, maximizing_player):

    //returns an action
    v ← MAX_VALUE(state, -∞, +∞)
    return the action in ACTIONS(state) with value v

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

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL_TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX_VALUE(RESULT(s,a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v

Output:

For tree = [[3,5,6], [9,1,2], [0,7,6]]
Optimal value : 6

Code:

```
MAX, MIN = 1000, -1000


def minimax(depth, nodeIndex, maximizingPlayer,
      values, alpha, beta):


  if depth == 3:
    return values[nodeIndex]


  if maximizingPlayer:


    best = MIN


    # Recur for left and right children
    for i in range(0, 2):


      val = minimax(depth + 1, nodeIndex * 2 + i,
            False, values, alpha, beta)
      best = max(best, val)
      alpha = max(alpha, best)


      # Alpha Beta Pruning
      if beta <= alpha:
        break


    return best
```

```
  else:
    best = MAX


    # Recur for left and
    # right children
    for i in range(0, 2):


      val = minimax(depth + 1, nodeIndex * 2 + i,
              True, values, alpha, beta)
      best = min(best, val)
      beta = min(beta, best)


      # Alpha Beta Pruning
      if beta <= alpha:
        break


    return best


# Driver Code
if __name__ == "__main__":


  values = [3, 5, 6, 9, 1, 2, 0, -1]
  print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
```

Output:

The optimal value is : 5