EXPERIMENT 3

Implement Backtracking algorithms for CSP

AIM:

To implement backtracking algorithm MINIMAX in MIN MAX playing game using Alpha beta pruning.

ALGORITHM

1. Define two constants, `MAX` and `MIN`, to represent the maximum and minimum values for initialization.

2. Create the `minimax` function, which takes the following parameters:

- `depth`: The current depth in the search tree.

- `nodeIndex`: The index of the current node.

- `maximizingPlayer`: A boolean indicating whether the current player is maximizing (True) or minimizing (False).

- `values`: A list of values representing the nodes in the tree.

- `alpha`: The best value found for the maximizing player.

- `beta`: The best value found for the minimizing player.

3. The function uses a recursive approach to explore the search tree. The termination condition is when the maximum depth (`depth == 4`) is reached. In this case, the function returns the value of the current node.

4. If it's a maximizing player's turn, the function initializes `best` as the minimum value (`MIN`). It then iterates through the left and right children of the current node, calling `minimax` recursively for each child. It updates `best` with the maximum value found and also updates `alpha` with the maximum value. Alpha-beta pruning is applied by checking if `beta` is less than or equal to `alpha`. If this condition is met, the loop breaks early. The best value for the maximizing player is returned.

5. If it's a minimizing player's turn, the function initializes `best` as the maximum value (`MAX`). It follows a similar process to the maximizing player, but this time it seeks the minimum value. It updates `best` and `beta` accordingly and applies alpha-beta pruning if `beta` is less than or equal to `alpha`.

6. The driver code initializes the `values` list, representing the values in the search tree, and calls the `minimax` function with the initial parameters. The optimal value is printed as the result.

PROGRAM

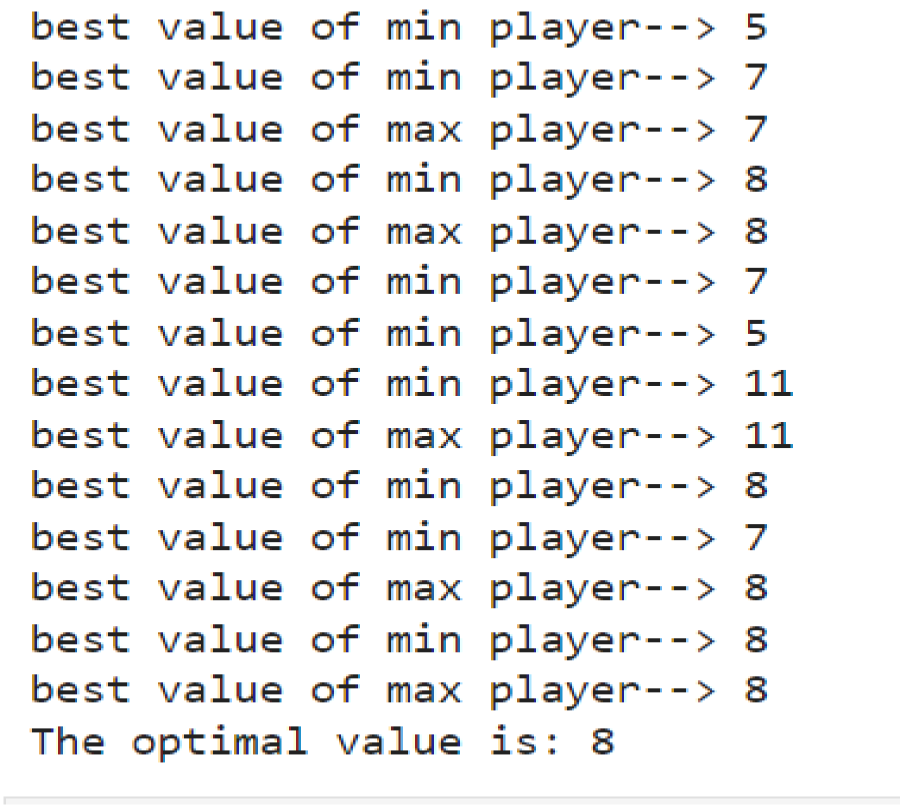
```
# Define constants for maximum and minimum values
MAX, MIN = 1000, -1000

# Minimax function for finding the optimal value in a search tree
def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    # Terminating condition: If the maximum depth is reached, return the value of the current node
    if depth == 4:
        return values[nodeIndex]

    if maximizingPlayer:
        best = MIN
        # For maximizing player, initialize best as the minimum value
        for i in range(0, 2):
            # Recur for left and right children
            val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)  # Update best with the maximum value
            alpha = max(alpha, best)  # Update alpha with the maximum value
            if beta <= alpha:
                break  # Alpha-Beta Pruning: If beta is less than or equal to alpha, break the loop
        print("best value of max player-->",best)
        return best  # Return the best value found so far for the maximizing player
    else:
        best = MAX
        # For minimizing player, initialize best as the maximum value
        for i in range(0, 2):
            # Recur for left and right children
            val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)  # Update best with the minimum value
            beta = min(beta, best)  # Update beta with the minimum value
            if beta <= alpha:
                break  # Alpha-Beta Pruning: If beta is less than or equal to alpha, break the loop
        print("best value of min player-->",best)
        return best  # Return the best value found so far for the minimizing player


# Driver code
if __name__ == "__main__":
    #values = [3, 5, 6, 9, 1, 2, 0, -1]
    values= [10,5,7,11,12,8,9,8,5,12,11,12,9,8,7,10]
    # Call the minimax function with initial parameters and print the result
    print("The optimal value is:", minimax(0, 0, True, values, MIN, MAX))
```

OUTPUT



```
best value of min player--> 5
best value of min player--> 7
best value of max player--> 7
best value of min player--> 8
best value of max player--> 8
best value of min player--> 7
best value of min player--> 5
best value of min player--> 11
best value of max player--> 11
best value of min player--> 8
best value of min player--> 7
best value of max player--> 8
best value of min player--> 8
best value of max player--> 8
The optimal value is: 8
```

RESULT:

The backtracking algorithm MINIMAX in MIN MAX playing game using Alpha beta pruning is implemented and the output is verified.