Kartik Padave

A022

70362019039

28/02/2022

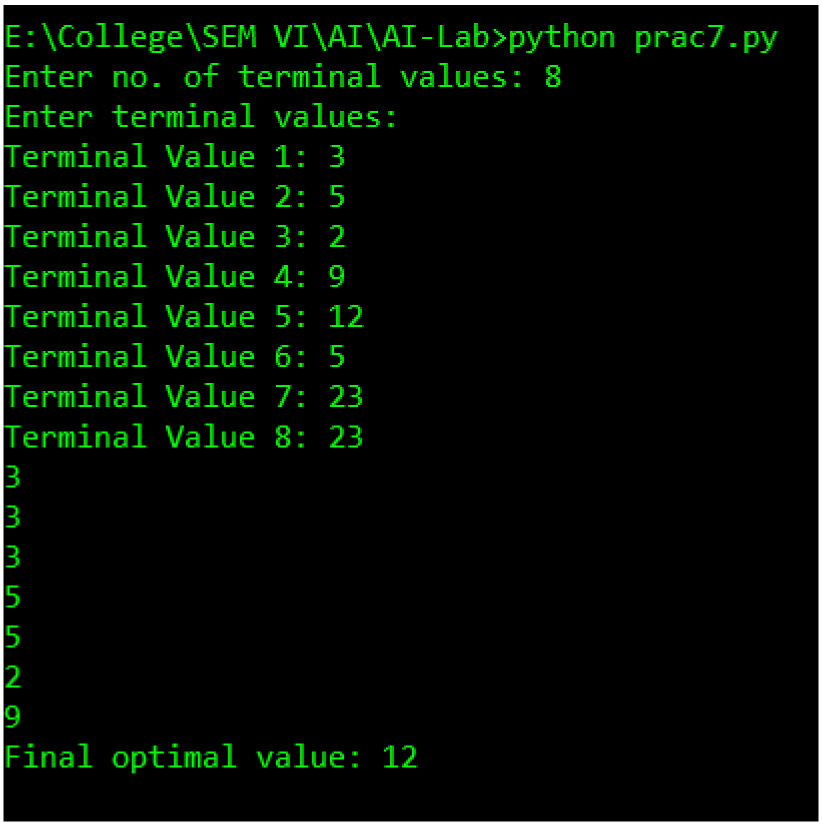
AI Practical 7

Aim: Minimax program with traversing and optimal value as output.

Code:

```python
import math

def minimax (curDepth, nodeIndex, maxTurn, scores, targetDepth):
    if (curDepth == targetDepth):
        return scores[nodeIndex]

    if (maxTurn):
        print(scores[nodeIndex])
        return max(minimax(curDepth + 1, nodeIndex * 2, False, scores,
targetDepth), minimax(curDepth + 1, nodeIndex * 2 + 1, False, scores,
targetDepth))
    else:
        print(scores[nodeIndex])
        return min(minimax(curDepth + 1, nodeIndex * 2, True, scores,
targetDepth), minimax(curDepth + 1, nodeIndex * 2 + 1, True, scores,
targetDepth))

# 3 5 2 9 12 5 23 23
n = int(input("Enter no. of terminal values: "))
scores = []
print("Enter terminal values: ")
for i in range(n):
    scores.append(int(input(f"Terminal Value {i+1}: ")))
treeDepth = math.log(len(scores), 2)

# print("The optimal value is : \n", end = "")
final = minimax(0, 0, True, scores, treeDepth)
print(f"Final optimal value: {final}")
```

Output:


```
E:\College\SEM VI\AI\AI-Lab>python prac7.py
Enter no. of terminal values: 8
Enter terminal values:
Terminal Value 1: 3
Terminal Value 2: 5
Terminal Value 3: 2
Terminal Value 4: 9
Terminal Value 5: 12
Terminal Value 6: 5
Terminal Value 7: 23
Terminal Value 8: 23
3
3
3
5
5
2
9
Final optimal value: 12
```

AI Practical 8

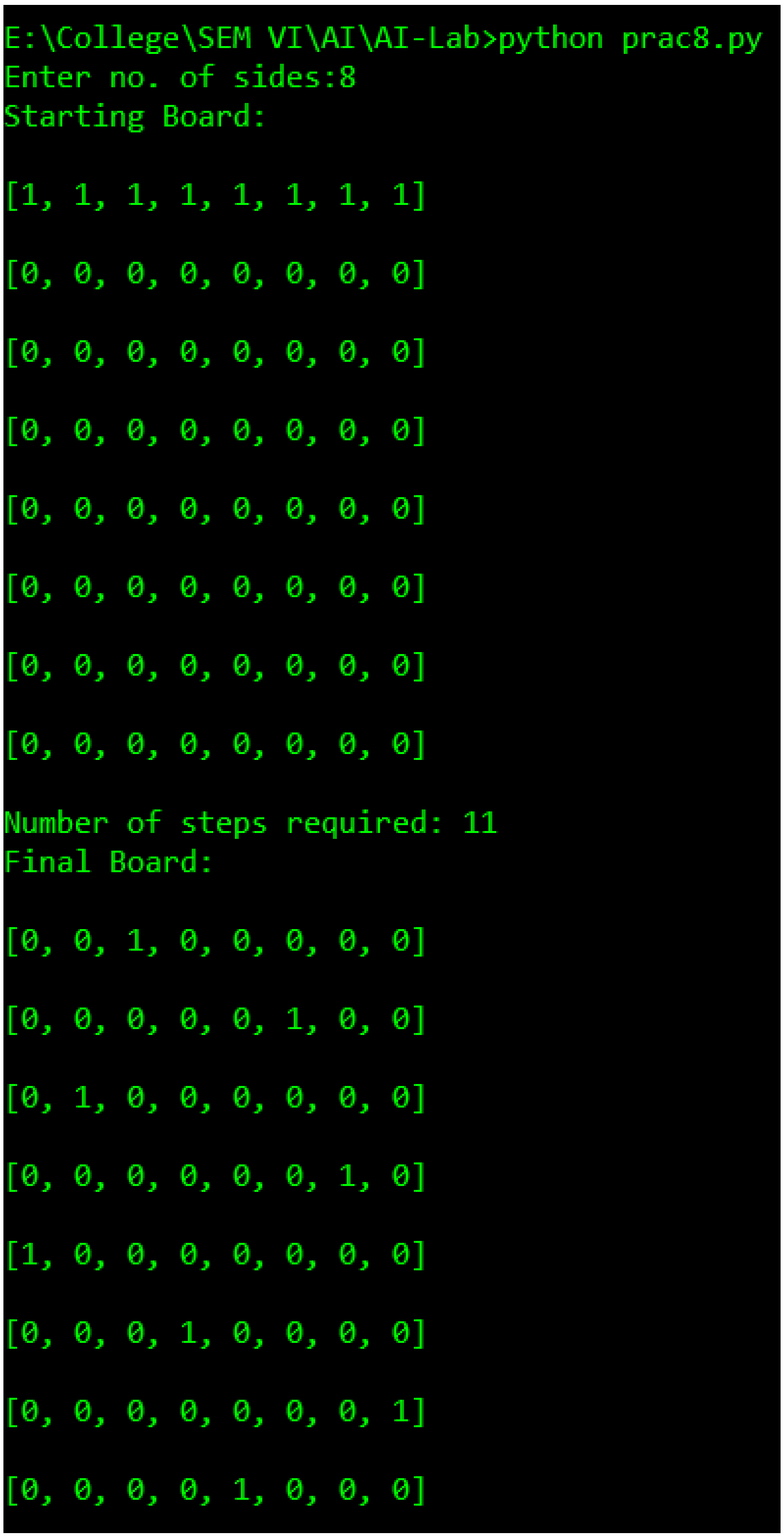
Aim: N-Queens problem.

Code:

```
import copy

def exists(i, j):
    return (i >= 0 and i < n and j >= 0 and j < n)


def contains(i, j, l, m, queen_pairs):
    if ((i, j, l, m) in queen_pairs) or ((l, m, i, j) in queen_pairs):
        return True
    return False

def position_queens_row_wise(board):
    for row in board:
        while row.count(1) > 1:
            for i in range(n):
                if board[i].count(1) == 0:
                    j = row.index(1)
                    board[i][j] = 1
                    row[j] = 0
                    break

    return board


def heuristic_value(board):
    h = 0
    queen_pairs = []
    for i in range(n):
        for j in range(n):

            if board[i][j]:
                for k in range(n):
                    if board[i][k] == 1 and k != j and not contains(i, j, i,
k, queen_pairs):
                        queen_pairs.append((i, j, i, k))
                        h += 1

                for k in range(n):
                    if board[k][j] == 1 and i != k and not contains(i, j, k,
j, queen_pairs):
                        queen_pairs.append((i, j, k, j))
                        h += 1
                l, m = i-1, j+1
                while exists(l, m):
                    if board[l][m] == 1 and not contains(i, j, l, m,
queen_pairs):
                        queen_pairs.append((i, j, l, m))
                        h += 1
                    l, m = l-1, m+1

                l, m = i+1, j-1
                while exists(l, m):
                    if board[l][m] == 1 and not contains(i, j, l, m,
queen_pairs):
                        queen_pairs.append((i, j, l, m))
                        h += 1
                    l, m = l+1, m-1

                l, m = i-1, j-1
                while exists(l, m):
                    if board[l][m] == 1 and not contains(i, j, l, m,
queen_pairs):
                        queen_pairs.append((i, j, l, m))
                        h += 1
                    l, m = l-1, m-1

                l, m = i+1, j+1
                while exists(l, m):
                    if board[l][m] == 1 and not contains(i, j, l, m,
queen_pairs):
                        queen_pairs.append((i, j, l, m))
                        h += 1
                    l, m = l+1, m+1

    return h


def hill_climbing(board):
    min_board = board
    min_h = 999999
    global n_side_moves, n_steps

    n_steps += 1

    if n_side_moves == 100:
        return -1

    sideway_move = False

    for i in range(n):
        queen = board[i].index(1)
        board[i][queen] = 0

        for k in range(n):

            if k != queen:
                board[i][k] = 1

                h = heuristic_value(board)

                if h < min_h:
                    min_h = h
                    min_board = copy.deepcopy(board)
                if h == min_h:
                    min_h = h
                    min_board = copy.deepcopy(board)
                    sideway_move = True

                board[i][k] = 0

        board[i][queen] = 1

    if sideway_move:
        n_side_moves += 1

    if min_h == 0:
        print("Number of steps required: {}".format(n_steps))
        return min_board

    return hill_climbing(min_board)

n_side_moves = 0
n_steps = 0

n = int(input("Enter no. of sides:"))
board = [
    [1] * n
]

for i in range(n-1):
    board.append([0] * n)

print("Starting Board:\n")
for i in board:
    print(f"{i}\n")

board = position_queens_row_wise(board)
board = hill_climbing(board)
```

```python
print("Final Board:\n")
for i in board:
    print(f"{i}\n")
```

Output:



```
E:\College\SEM VI\AI\AI-Lab>python prac8.py
Enter no. of sides:8
Starting Board:

[1, 1, 1, 1, 1, 1, 1, 1]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

Number of steps required: 11
Final Board:

[0, 0, 1, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 1, 0, 0]

[0, 1, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 1, 0]

[1, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 1, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 1]

[0, 0, 0, 0, 1, 0, 0, 0]
```