Kartik Padave

A022

70362019039

28/02/2022

**AI Practical 7**

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

**Code:**

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:**

Graphical user interface, text

Description automatically generated

**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)

print("Final Board:\n")

for i in board:

    print(f"{i}\n")

**Output:**

Text

Description automatically generated with medium confidence