EX.NO:02 ITERATIVE DEEPENING AND

**DATE:10.8.2022 DEPTH LIMITED SEARCH**

# AIM:

To solve the given 4 queens problem using iterative deepening DFS and depth limited search.

**Iterative deepening DFS:**

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

# SOURCE CODE:

res = []

def totalNQueens(n):

def check(x, y, board):

i = x

j = y

# checking upper left diagonal

while i >= 0 and j >= 0:

if board[i][j] == 1:

return False

i -= 1

j -= 1

i = x

j = y

# checking lower left diagonal

while i < n and j >= 0:

if board[i][j] == 1:

return False

i += 1

j -= 1

i = x

j = y

# checking the column

while j >= 0:

if board[i][j] == 1:

return False

j -= 1

return True

def dfs(col, board, depth):

if col >= n:

res.append([])

for i in range(n):

res[-1].append("")

for j in range(n):

if board[i][j]:

res[-1][-1] += "Q"

else:

res[-1][-1] += "#"

return

if depth <= 0:

return False

for i in range(n):

if check(i, col, board):

board[i][col] = 1

dfs(col+1, board, depth-1)

board[i][col] = 0

board = [

[0]\*n for i in range(n)

]

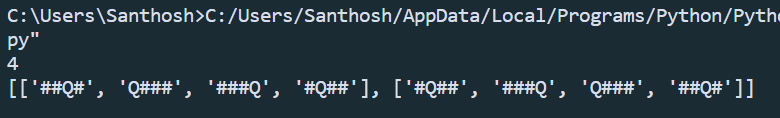
depth = int(input())

dfs(0, board, depth)

print(res)

totalNQueens(4)

# OUTPUT:



# Depth limited:

# A depth-limited search algorithm is similar to depth-first search with a predetermined limit.

# Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

# SOURCE CODE:

res = []

def totalNQueens(n):

def check(x, y, board):

i = x

j = y

# checking upper left diagonal

while i >= 0 and j >= 0:

if board[i][j] == 1:

return False

i -= 1

j -= 1

i = x

j = y

# checking lower left diagonal

while i < n and j >= 0:

if board[i][j] == 1:

return False

i += 1

j -= 1

i = x

j = y

# checking the column

while j >= 0:

if board[i][j] == 1:

return False

j -= 1

return True

def dfs(col, board, maxdepth):

if col >= n:

res.append([])

for i in range(n):

res[-1].append("")

for j in range(n):

if board[i][j]:

res[-1][-1] += "Q"

else:

res[-1][-1] += "#"

return

if maxdepth <= 0:

return False

for i in range(n):

if check(i, col, board):

board[i][col] = 1

dfs(col+1, board, maxdepth-1)

board[i][col] = 0

board = [

[0]\*n for i in range(n)

]

for i in range(int(input())):

res = []

dfs(0, board, i+1)

print(res)

totalNQueens(4)

# OUTPUT:

# 

**RESULT:**

The given 4-queens problem is solved using IDDFS and Depth limited search successfully.