j-weight) }

nto the knap

op.

ed label

e knapsack.

O(1) for

ack prob

knapsa

Assignment No. 05

Title of the Assignment :
Design n-queens matrix having first queen placed. Use backtracking to place remaining queens to generate the final n-queen's matrix.

Objective of the Assignment :
Students should be able to understand and solve n-queen problem and understand basics of backtracking

Prerequisite :

1. Basic of Python or Java Programming
2. Concept of backtracking method
3. N-Queen Problem

Theory :

Introduction to Backtracking :

- Many problems are difficult to solve algorithmically. Backtracking makes it possible to solve at least some large instances of difficult combinatorial problems.

What is backtracking ?

Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken.

TSSM
Page No. :

In backtracking, we first take a step & then we see if step taken is correct or not i.e., whether it will give a correct answer or not.

Thus, the general steps of backtracking are:

- Start with a sub-solution
- Check if this sub-solution will lead to the solution or
- If not, then come back and change the sub-solution Continue again.

- Applications of Backtracking:

- N Queens Problem
- Sum of subsets Problem
- Graph coloring
- Hamiltonian cycles.

N queens on NXN chessboard

N-queens problem:

A classic combinational problem is to place n queens on a n*n chess board so that no two attack. i.e, no two queens are on the same row, column or dizgonal.

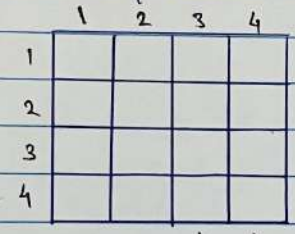
```
# Python3 program to solve
# Problem using backtrackin
global N
N = 4
def printSolution(board):
for i in range(N):
for j in range(N):
print(board[i][j], end = "")
print()
# A utility function to chec
# be placed on board[row][
# function is called when "
# already placed in column
# So we need to check only
# attacking queens
def isSafe(board, row, col)
# Check this row on left si
for i in range(col):
if board[row][i] == 1:
return False
# Check upper diagonal o
for i, j in zip(range(row, -
range(col, -1, -1)):
if board[i][j] == 1:
return False
# Check lower diagonal o
for i, j in zip(range(row, N
range(col, -1, -1)):
if board[i][j] == 1:
return False
return True
def solveNQUtil(board, c
# base case: If all queens
# then return true
if col >= N:
return True
# Consider this column a
# this queen in all rows
for i in range(N):
if isSafe(board, i, col):
# Place this queen in bo
board[i][col] = 1
# recur to place rest of t
if solveNQUtil(board, c
return True
# If placing queen in bo
# doesn't lead to a solut
# queen from board[i][c
board[i][col] = 0
# if the queen can not b
# this column col then
return False
```

Algorithm :

1. Start in the leftmost column.
2. If all queens are place return true.
3. Try all rows in the current column.
4. If all rows have been tried & nothing worked return false to trigger backtracking.

4-Queen Problem :

Problem : Given 4 x 4 chessboard, arrange four queens in a way, such that no two queens attack each other.



4x4 chessboard

All possible solutions for 4-queen are shown in fig.

	1	2	3	4
1		Q_1		
2				Q_2
3	Q_3			
4			Q_4	

fig. (a): solution-1

	1	2	3	4
1			Q_1	
2	Q_2			
3				Q_3
4		Q_4		

fig. (b): solution-2

Conclusion :

In this way we have explored Concept of Backtracking and solve n-queen problem using backtracking method.

PPadhavar