**Exercise - 8**

**Backtracking Algorithm**

**Aim:** To implement a backtracking algorithm in Python to solve the n-Queen problem and the Hamiltonian Circuit problem.

**Algorithm:**

**n-Queen Problem:**

1. Start with an empty chessboard of size n x n.

2. Define a function is\_safe(board, row, col, n) that checks whether it is safe to place a queen at the given position (row, col) on the board. This function should check if there is any queen already placed in the same column, row, or diagonal as the given position.

3. Define a recursive function solve\_n\_queens(board, col, n) that tries to place a queen in the given column col of the board. This function will return True if a solution is found and False otherwise. The base case for this function is when col reaches n, indicating that all queens have been successfully placed.

4. Inside the solve\_n\_queens function, iterate through each row in the column col.

5. For each row, check if it is safe to place a queen at (row, col) using the is\_safe function. If it is safe, mark that position as a queen on the board.

6. Recursively call solve\_n\_queens with the next column (col + 1). If this recursive call returns True, it means a solution has been found, so return True.

7. If the recursive call in step 6 returns False, it means that placing a queen at (row, col) didn't lead to a solution. Therefore, backtrack by removing the queen from the board (i.e., mark that position as empty) and continue with the next row in the current column.

8. If all rows have been tried without success, return False to backtrack further.

9. Finally, create a main function n\_queen(n) that initializes the chessboard and calls solve\_n\_queens with the initial column as 0. If solve\_n\_queens returns True, print the board as the solution; otherwise, print that no solution exists.

**Hamiltonian Circuit:**

1. Start at an arbitrary vertex (let's call it "start").

2. Initialize an empty path list and an empty visited set.

3. Add the "start" vertex to the path list and mark it as visited.

4. If the path list contains all the vertices of the graph and there is an edge from the last vertex in the path list to the "start" vertex, return true (a Hamiltonian circuit is found).

5. Otherwise, recursively explore all the adjacent vertices of the last vertex in the path list that have not been visited yet.

* For each unvisited adjacent vertex:
  + Add the vertex to the path list and mark it as visited.
  + Recursively call the algorithm on the updated path list.
  + If the recursive call returns true, return true (a Hamiltonian circuit is found).
  + If the recursive call returns false, remove the vertex from the path list and mark it as unvisited.

6. If no Hamiltonian circuit is found after exploring all possibilities, return false.

**Source Code:**

**n-Queen Problem:**

N = int(input("Enter the number of queens: "))

#chessboard

#NxN matrix with all elements 0

board = [[0]\*N for \_ in range(N)]

def is\_attack(i, j):

    for k in range(0,N):

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

            return True

    for k in range(0,N):

        for l in range(0,N):

            if (k+l==i+j) or (k-l==i-j):

                if board[k][l]==1:

                    return True

    return False

def N\_queen(n):

    if n==0:

        return True

    for i in range(0,N):

        for j in range(0,N):

            if (not(is\_attack(i,j))) and (board[i][j]!=1):

                board[i][j] = 1

                if N\_queen(n-1)==True:

                    return True

                board[i][j] = 0

    return False

N\_queen(N)

for i in board:

    print (i)

**Hamiltonian Circuit:**

def hamiltonian\_circuit(graph, start, path, visited):

 path.append(start)

 visited.add(start)

 if len(path) == len(graph) and start in graph[path[0]]:

     return True

 for neighbor in graph[start]:

     if neighbor not in visited:

         if hamiltonian\_circuit(graph, neighbor, path, visited):

             return True

 path.pop()

 visited.remove(start)

 return False

def solve\_hamiltonian\_circuit(graph):

 start = list(graph.keys())[0]

 path = []

 visited = set()

 if hamiltonian\_circuit(graph, start, path, visited):

     return path

 else:

     return None

def display\_hamiltonian\_circuit(graph, circuit):

 if circuit is None:

     print("No Hamiltonian circuit found.")

 else:

     print("Hamiltonian circuit:")

 for vertex in circuit:

     print(vertex, end=' ')

 print(circuit[0])

graph = {

 'A': ['B', 'C', 'D'],

 'B': ['A', 'C', 'D'],

 'C': ['A', 'B', 'D'],

 'D': ['A', 'B', 'C']

}

circuit = solve\_hamiltonian\_circuit(graph)

display\_hamiltonian\_circuit(graph, circuit)

**Sample Input and Output:**

**n-Queen Problem:**

Input: 5

A picture containing text, font, screenshot, number

Description automatically generated

Input: 7

A picture containing text, screenshot, font, white

Description automatically generated

**Hamiltonian Circuit:**

Input: A -> B, C, D

B -> A, C, D

C -> A, B, D

D -> A, B, C

Output:

A black text on a white background

Description automatically generated with low confidence

**Result:**

Thus, backtracking algorithm in Python to solve the n-Queen problem and the Hamiltonian Circuit problem has been successfully implemented and the output is verified.