**DFS-BFS**

from collections import deque

from typing import Deque, List

class Graph:

def \_\_init\_\_(self) -> None:

self.n = int(input("Enter number of nodes"))

self.nodes = {i+1 : [] for i in range(self.n)}

def addEdge(self,start:int,end:int):

self.nodes[start].append(end)

self.nodes[end].append(start)

def printGraph(self):

for node, list in self.nodes.items():

print(f"{node} -> ",end="")

for vertex in list:

print(vertex,end=" ")

print()

def dfsHelper(self,current : int,visited : List[bool]):

print(current,end= " ")

visited[current] = True

for adjacentNode in self.nodes[current]:

if visited[adjacentNode] == False:

self.dfsHelper(adjacentNode,visited)

def dfs(self,start : int):

visited = [False] \* (self.n + 1)

print(f"DFS from {start}: ",end="")

self.dfsHelper(start,visited)

print()

def bfs(self,start:int):

queue = deque()

visited = [False] \* (self.n + 1)

print(f"BFS from {start}: ",end="")

queue.append(start)

visited[start] = True

self.bfsHelper(queue,visited)

print()

def bfsHelper(self,queue : Deque,visited: List[bool]):

if len(queue) == 0:

return

current = queue.popleft()

print(current,end=" ")

for adjacentNode in self.nodes[current]:

if visited[adjacentNode] == False:

queue.append(adjacentNode)

visited[adjacentNode] = True

self.bfsHelper(queue,visited)

g = Graph()

g.addEdge(1,2)

g.addEdge(1,3)

g.addEdge(1,4)

g.addEdge(5,2)

g.addEdge(5,3)

g.addEdge(5,4)

g.printGraph()

g.dfs(3)

g.bfs(1)

**N-queens**

def solveNQueens(n : int):

col = set()

posDiag = set() # determined by r+c

negDiag = set() # determined by r-c

res = []

board = [['.' for \_ in range(n)] for \_ in range(n)]

def backtrack(r):

if (r == n):

res.append([''.join(row) for row in board])

return

for c in range(n):

if (c in col or r+c in posDiag or r-c in negDiag):

continue

col.add(c)

posDiag.add(r + c)

negDiag.add(r - c)

board[r][c] = 'Q'

backtrack(r + 1)

col.remove(c)

posDiag.remove(r + c)

negDiag.remove(r - c)

board[r][c] = '.'

backtrack(0)

return res

def printSolutions(boards):

for board in enumerate(boards):

print(f"Solution: {board[0]+1}")

for row in board[1]:

for col in row:

print(col, end=' ')

print()

print()

if \_\_name\_\_ == "\_\_main\_\_":

boards = solveNQueens(8)

printSolutions(boards)

**Chatbot**

import random

# Dictionary containing available rooms and their availability

rooms = {

"Single Room": 10,

"Double Room": 10,

"Suite": 5

}

def display\_available\_rooms():

"""

Display the available rooms and their availability.

"""

print("Available Rooms:")

for room, availability in rooms.items():

print(f"- {room}: {availability} available")

print()

def make\_reservation(room\_type, num\_rooms):

"""

Attempt to make a reservation for the given room type and number of rooms.

"""

if room\_type in rooms and rooms[room\_type] >= num\_rooms:

rooms[room\_type] -= num\_rooms

print(f"Reservation confirmed for {num\_rooms} {room\_type}(s). Enjoy your stay!")

else:

print("Sorry, there are not enough available rooms of that type. Please choose another.")

def display\_hotel\_info():

"""

Display basic information about the hotel.

"""

print("Hotel Information:")

print("Name: Chatbot Hotel")

print("Location: 123 Main Street")

print("Check-in Time: 3:00 PM")

print("Check-out Time: 12:00 PM")

def main():

print("Welcome to Chatbot Hotel! How can I assist you today?")

display\_hotel\_info()

while True:

print("What would you like to do?")

print("1. Check room availability")

print("2. Make a reservation")

print("3. Get hotel information")

print("4. Exit")

choice = input("> ")

if choice == "1":

display\_available\_rooms()

elif choice == "2":

print("Please select a room type (Single Room, Double Room, Suite):")

room\_type = input("> ")

print("How many rooms would you like to reserve?")

num\_rooms = int(input("> "))

make\_reservation(room\_type, num\_rooms)

elif choice == "3":

display\_hotel\_info()

elif choice == "4":

print("Thank you for choosing Chatbot Hotel. Have a pleasant day!")

break

else:

print("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Expert system**

class HospitalExpertSystem:

def \_\_init\_\_(self):

self.departments = {}

self.staff = {}

def add\_department(self, department\_name, facilities):

self.departments[department\_name] = facilities

def add\_staff(self, staff\_name, department):

if department in self.staff:

self.staff[department].append(staff\_name)

else:

self.staff[department] = [staff\_name]

def get\_departments(self):

return self.departments.keys()

def get\_facilities(self, department):

return self.departments.get(department, "Department not found")

def get\_staff(self, department):

return self.staff.get(department, [])

def main():

hospital\_system = HospitalExpertSystem()

# Adding hospital departments and facilities

hospital\_system.add\_department("Cardiology", ["ECG", "Echocardiogram", "Cardiac Catheterization"])

hospital\_system.add\_department("Orthopedics", ["X-ray", "MRI", "Physical Therapy"])

hospital\_system.add\_department("Oncology", ["Chemotherapy", "Radiation Therapy", "Surgery"])

# Adding staff members

hospital\_system.add\_staff("Dr. Smith", "Cardiology")

hospital\_system.add\_staff("Dr. Johnson", "Cardiology")

hospital\_system.add\_staff("Dr. Brown", "Orthopedics")

hospital\_system.add\_staff("Dr. Lee", "Oncology")

# Retrieving hospital departments

print("Hospital Departments:")

for department in hospital\_system.get\_departments():

print(f"- {department}: {hospital\_system.get\_facilities(department)}")

# Retrieving staff members for a department

department = "Cardiology"

print(f"\nStaff in {department}: {hospital\_system.get\_staff(department)}")

if \_\_name\_\_ == "\_\_main\_\_":

main()