1. Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

```
class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v):
        if u in self.graph:
            self.graph[u].append(v)
        else:
            self.graph[u] = [v]
        if v in self.graph:
            self.graph[v].append(u)
        else:
            self.graph[v] = [u]

    def dfs_util(self, v, visited):
        visited[v] = True
        print(v, end=' ')

        for i in self.graph[v]:
            if not visited[i]:
                self.dfs_util(i, visited)

    def dfs(self, start):
        visited = {node: False for node in self.graph}
        self.dfs_util(start, visited)

    def bfs(self, start):
        visited = {node: False for node in self.graph}
        queue = [start]
        visited[start] = True

        while queue:
            vertex = queue.pop(0)
            print(vertex, end=' ')

            for i in self.graph[vertex]:
                if not visited[i]:
                    queue.append(i)
                    visited[i] = True

# Main program
graph = Graph()

n = int(input("Enter the number of edges: "))
```

```
print("Enter the edges (e.g., '1 2' for an edge between vertices 1 and 2):")
for _ in range(n):
    u, v = map(int, input().split())
    graph.add_edge(u, v)

start_vertex = int(input("Enter the starting vertex: "))

print("Depth First Search (DFS):")
graph.dfs(start_vertex)
print()

print("Breadth First Search (BFS):")
graph.bfs(start_vertex)
```

2.Implement Greedy search algorithm for Job Scheduling Problem

```
# Define a function to perform Greedy job scheduling
def greedy_job_scheduling(jobs):
    # Sort jobs by their deadlines
    sorted_jobs = sorted(jobs, key=lambda x: x[1])

    # Initialize variables
    schedule = []
    max_deadline = max(job[1] for job in sorted_jobs)
    time_slots = [False] * (max_deadline + 1)

    # Greedy scheduling
    for job in sorted_jobs:
        deadline = job[1]
        while deadline > 0:
            if not time_slots[deadline]:
                schedule.append(job)
                time_slots[deadline] = True
                break
            deadline -= 1

    return schedule

# Main function
def main():
    # Take user input for job details
    jobs = []
    n = int(input("Enter the number of jobs: "))
    for i in range(n):
        print(f"Enter details for job {i + 1}:")
        name = input("Name: ")
        deadline = int(input("Deadline: "))
        profit = int(input("Profit: "))
        jobs.append((name, deadline, profit))

    # Call greedy job scheduling function
    schedule = greedy_job_scheduling(jobs)

    # Print the schedule
    print("\nJob Schedule:")
    for job in schedule:
        print(f"{job[0]} -> Deadline: {job[1]}, Profit: {job[2]}")

# Run the main function
if __name__ == "__main__":
    main()
```

3. Implement a solution for a Constraint Satisfaction Problem using Backtracking for n-queens problem.

```
def is_safe(board, row, col, n):

    for i in range(row):
        if board[i][col] == 1:
            return False

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False



    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] == 1:
            return False

    return True

def solve_n_queens(board, row, n):
    if row >= n:
        return True

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1

            if solve_n_queens(board, row + 1, n):
                return True

            board[row][col] = 0

    return False

def print_board(board):
    for row in board:
        print(' '.join(map(str, row)))

def main():
    n = int(input("Enter the number of queens: "))
    board = [[0] * n for _ in range(n)]

    if solve_n_queens(board, 0, n):
        print("Solution exists. Placing queens on the board:")
        print_board(board)
```

```
    else:
        print("No solution exists for this configuration.")

if __name__ == "__main__":
    main()
```

5. Implement a solution for a Constraint Satisfaction Problem using Branch and Bound for n-queens problem

```
def is_safe(board, row, col, N):
    # Check if there is a queen in the same column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check upper diagonal on right side
    for i, j in zip(range(row, -1, -1), range(col, N)):
        if board[i][j] == 1:
            return False

    return True

def solve_n_queens_util(board, row, N):
    # If all queens are placed, return True
    if row >= N:
        return True

    # Try placing this queen in all columns one by one
    for col in range(N):
        if is_safe(board, row, col, N):
            # Place this queen in board[row][col]
            board[row][col] = 1

            # Recur to place rest of the queens
            if solve_n_queens_util(board, row + 1, N):
                return True

            # If placing queen in board[row][col] doesn't lead to a solution,
            # then backtrack and remove the queen from board[row][col]
            board[row][col] = 0

    # If queen can't be placed in any column in this row, return False
    return False
```

```
def solve_n_queens(N):
    board = [[0 for _ in range(N)] for _ in range(N)]

    if not solve_n_queens_util(board, 0, N):
        print("Solution does not exist")
        return

    print("Solution:")
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print()

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

if __name__ == "__main__":
    main()
```

6. Develop an elementary chatbot for any suitable customer interaction Application

```
import streamlit as st
bot_name = "College Buddy"

knowledge_base = {

    "what is your name?" : [
        f"My name is {bot_name}! \n Happy to help you out with your College enquiries!"
    ],

    "hello": [
        f"Hello my name is {bot_name}! \n Happy to help you out with your College enquiries!"
    ],

    "what are the best colleges from pune?": [
        "COEP",
        "PICT",
        "VIT",
        "CUMMINS",
        "PCCOE"
    ],

    "which are the best engineering branches?" : [
        "Computer Engineering",
        "IT Engineering",
        "ENTC Engineering"
    ],

    "what are the top branch cut-offs for coep?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2 percentile",
    ],

    "what are the top branch cut-offs for pict?" : [
        "Computer Engineering : 99.4 percentile",
        "IT Engineering : 98.6 percentile",
        "ENTC Engineering: 97.2 percentile",
    ],

    "what are the top branch cut-offs for vit?" : [
        "Computer Engineering : 99.8 percentile",
        "IT Engineering: 97.1 percentile",
        "ENTC Engineering: 96.2 percentile",
    ],
```

```
    "what are the top branch cut-offs for cummins?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2",
    ],

    "what are the top branch cut-offs for pccoe?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2",
    ],

    "When do college admissions start?": [
        "Admissions generally start around August",
    ],

}

st.header("College Enquiry Rule Based Chatbot")

def respond(input: str):
    if (input in knowledge_base):
        print(input)
        values = knowledge_base[input]
        for value in values:
            st.write(value)
    else:
        print(input)
        key = input
        st.write("Question is not present in the knowledge base!\nCould you please enter the
appropriate answer for the question below-")
        answer = st.text_input("Answer")
        add = st.button("Add answer")
        if (add):
            knowledge_base[key] = [answer]

if __name__ == "__main__":
    input = st.text_input("Enter a query here-")
    input = input.lower()
    col1, col2 = st.columns([1,0.1])
    with col1:
        ask = st.button("Ask")
    with col2:
        quit = st.button("Quit")
    if (ask):
        respond(input)
    if (quit):
        st.write("Thank you for using the Chatbot")
```

Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a non-recursive algorithm for searching all the vertices of a graph or tree data structure.

```
class Graph:
    def __init__(self):
        self.adjacency_list = {}

    def add_edge(self, u, v):
        if u not in self.adjacency_list:
            self.adjacency_list[u] = []
        if v not in self.adjacency_list:
            self.adjacency_list[v] = []
        self.adjacency_list[u].append(v)
        self.adjacency_list[v].append(u)

    def dfs(self, start):
        stack = [start]
        visited = set()
        while stack:
            vertex = stack.pop()
            if vertex not in visited:
                print(vertex, end=' ')
                visited.add(vertex)
                for neighbor in reversed(self.adjacency_list[vertex]):
                    if neighbor not in visited:
                        stack.append(neighbor)

    def bfs(self, start):
        queue = [start]
        visited = set()
        while queue:
            vertex = queue.pop(0)
            if vertex not in visited:
                print(vertex, end=' ')
                visited.add(vertex)
                for neighbor in self.adjacency_list[vertex]:
                    if neighbor not in visited:
                        queue.append(neighbor)
```

```
# Main code
graph = Graph()

# User input for adding edges
while True:
    edge = input("Enter an edge (or 'done' to finish): ").strip().split()
    if edge[0].lower() == 'done':
        break
    u, v = map(int, edge)
    graph.add_edge(u, v)

start_vertex = int(input("Enter the start vertex: "))

print("Depth First Search:")
graph.dfs(start_vertex)
print("\nBreadth First Search:")
graph.bfs(start_vertex)
```