1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem

Program

```
def dfs(jug1, jug2, visited):
    if (jug1, jug2) in visited:
        return False
    visited.add((jug1, jug2))

    print(f"({jug1}, {jug2})")

    if jug1 == 2 or jug2 == 2:
        print("Solution found!")
        return True

    return any([
        dfs(4, jug2, visited),
        dfs(jug1, 3, visited),
        dfs(0, jug2, visited),
        dfs(jug1, 0, visited),
        dfs(jug1 - min(jug1, 3 - jug2), jug2 + min(jug1, 3 - jug2), visited),
        dfs(jug1 + min(jug2, 4 - jug1), jug2 - min(jug2, 4 - jug1), visited)
    ])

dfs(0, 0, set())
```

Output

```
(0, 0)
(4, 0)
(4, 3)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
Solution found!
(1, 3)
(1, 0)
(0, 1)
(4, 1)
(2, 3)
Solution found!
```

2. Implement and demonstrate the best-first search algorithm on the Missionaries and Cannibals

Problem

```
from queue import PriorityQueue

def best_first_search(start_state):
    frontier = PriorityQueue()
    frontier.put((0, start_state))
    explored = set()

    while not frontier.empty():
        _, current_state = frontier.get()
        explored.add(current_state)

        if current_state == (0, 0, 0):
            return "Solution found"

        m, c, b = current_state
        successors = [(m-1, c, 1-b), (m-2, c, 1-b), (m, c-1, 1-b), (m, c-2, 1-b), (m-1, c-1, 1-b)]
        for successor in successors:
            if 0 <= successor[0] <= 3 and 0 <= successor[1] <= 3 and (successor[0] >= successor[1] or
successor[0] == 0) and (3-successor[0] >= 3-successor[1] or successor[0] == 3):
                if successor not in explored:
                    frontier.put((0, successor))
                    explored.add(successor)

    return "No solution found"

start_state = (3, 3, 1)
print(best_first_search(start_state))
```

Output

Solution found!

3. Implement A* search Algorithm.

Program

```
import heapq

def astar(start, goal, heuristic, get_neighbors):
    open_set = [(0, start)]
    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            return path[::-1]

        for neighbor in get_neighbors(current):
            tentative_g_score = g_score[current] + 1
            if tentative_g_score < g_score.get(neighbor, float('inf')):
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                heapq.heappush(open_set, (tentative_g_score + heuristic(neighbor, goal), neighbor))

    return None

# Example usage:
def heuristic(state, goal):
    return abs(state[0] - goal[0]) + abs(state[1] - goal[1])

def get_neighbors(state):
    x, y = state
    return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]  # Assuming 4-connected grid

start = (0, 0)
goal = (5, 5)
path = astar(start, goal, heuristic, get_neighbors)
print(path)
```

Output

```
[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5)]
```

4. Implement of the AO* search algorithm

Program

```
import heapq

def ao_star(start, goal, heuristic, get_neighbors):
    open_set = [(0, start)]
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            return path[::-1]

        for neighbor in get_neighbors(current):
            tentative_g_score = g_score[current] + 1
            if tentative_g_score < g_score.get(neighbor, float('inf')):
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic(neighbor, goal)
                heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

# Example usage:
def heuristic(state, goal):
    return abs(state[0] - goal[0]) + abs(state[1] - goal[1])

def get_neighbors(state):
    x, y = state
    return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]  # Assuming 4-connected grid

start = (0, 0)
goal = (5, 5)
path = ao_star(start, goal, heuristic, get_neighbors)
print(path)
```

Output

```
[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5)]
```

5. Solve 8-queens problem with suitable assumptions

Program

```
def is_safe(board, row, col):
    # Check if there's a queen in the same column
    for i in range(row):
        if board[i] == col:
            return False
        # Check diagonals
        if abs(board[i] - col) == row - i:
            return False
    return True

def solve_queens(board, row):
    if row == 8:  # All queens are placed
        return True
    for col in range(8):
        if is_safe(board, row, col):
            board[row] = col
            if solve_queens(board, row + 1):
                return True
            board[row] = -1  # Backtrack
    return False

board = [-1] * 8  # Initialize empty board
if solve_queens(board, 0):
    for row in board:
        print(row)
else:
    print("No solution found.")
```

Output

```
[0, 4, 7, 5, 2, 6, 1, 3]
```

6. Implementation of TSP using heuristic approach

Program

```
import numpy as np

def tsp_heuristic(distance_matrix):
    num_cities = distance_matrix.shape[0]
    unvisited_cities = set(range(num_cities))
    current_city = np.random.randint(num_cities)  # Start from a random city
    tour = [current_city]

    while unvisited_cities:
        nearest_city = min(unvisited_cities, key=lambda city: distance_matrix[current_city][city])
        tour.append(nearest_city)
        unvisited_cities.remove(nearest_city)
        current_city = nearest_city
```

```
    return tour

# Example usage:
# Distance matrix representing the distances between cities
distance_matrix = np.array([[0, 10, 15, 20],
                            [10, 0, 35, 25],
                            [15, 35, 0, 30],
                            [20, 25, 30, 0]])

tour = tsp_heuristic(distance_matrix)
print("Tour:", tour)
```

Output

Tour: [0, 1, 3, 2]

7. Implementation of the problem solving strategies, either forward chaining or backward chaining

Program

```
def forward_chaining(goal, rules, known_facts):
    agenda = known_facts.copy()
    while agenda:
        fact = agenda.pop(0)
        if fact == goal:
            return True
        for rule in rules:
            if rule[0] == fact:
                if all(subgoal in known_facts for subgoal in rule[1]):
                    agenda.append(rule[2])
                    known_facts.append(rule[2])
    return False

def backward_chaining(goal, rules, known_facts):
    if goal in known_facts:
        return True
    for rule in rules:
        if rule[2] == goal:
            if all(backward_chaining(subgoal, rules, known_facts) for subgoal in rule[1]):
                known_facts.append(goal)
                return True
    return False

# Example usage:
# Rules: (premise, [list of conditions], conclusion)
rules = [("A", [], "B"), ("B", ["A"], "C"), ("D", ["B"], "E"), ("E", [], "F")]
known_facts = ["A"]
goal = "F"

# Forward Chaining
result_forward = forward_chaining(goal, rules, known_facts)
print("Forward Chaining - Can goal be inferred?", result_forward)
```

```
# Backward Chaining
result_backward = backward_chaining(goal, rules, known_facts)
print("Backward Chaining - Can goal be inferred?", result_backward)
```

Output

Forward Chaining - Can goal be inferred? True
Backward Chaining - Can goal be inferred? True

8.Implement resolution principle of FOPL related problems

Problem

```
def resolve(clause1, clause2):
    resolvents = set()
    for literal1 in clause1:
        for literal2 in clause2:
            if literal1[1:] == literal2 or literal2[1:] == literal1:
                resolvent = tuple(set(clause1) | set(clause2) - {literal1, literal2})
                resolvents.add(resolvent)
    return resolvents

def resolution(agenda):
    while True:
        new_clauses = set()
        for clause1 in agenda:
            for clause2 in agenda:
                if clause1 != clause2:
                    resolvents = resolve(clause1, clause2)
                    if () in resolvents:  # Empty clause found, contradiction
                        return True
                    new_clauses.update(resolvents)
        if new_clauses.issubset(agenda):  # No new clauses added
            return False
        agenda.update(new_clauses)

# Example usage:
agenda = {("A", "~B"), ("~A", "B")}
result = resolution(agenda)
print("Contradiction found?" if result else "No contradiction found")
```

Output

Contradiction found?

9. Implementation of a tic-tac-toe game in Python

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board):
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
```

```
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != ' ' or board[0][2] == board[1][1] ==
board[2][0] != ' ':
        return board[1][1]
    return None

def tic_tac_toe():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    current_player = 'X'
    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    for _ in range(9):
        row, col = map(int, input(f"Player {current_player}, enter row and column (0-2): ").split())
        if board[row][col] == ' ':
            board[row][col] = current_player
            print_board(board)
            winner = check_winner(board)
            if winner:
                print(f"Player {winner} wins!")
                return
            current_player = 'O' if current_player == 'X' else 'X'
        else:
            print("That cell is already taken!")

    print("It's a draw!")

tic_tac_toe()
```

Output

```
Welcome to Tic-Tac-Toe!
 | |
-----
 | |
-----
 | |
-----
Player X, enter row and column (0-2): 1 1
 | |
-----
 |X|
-----
 | |
-----
Player O, enter row and column (0-2): 0 0
O| |
-----
 |X|
-----
```

```
 | |
-----
Player X, enter row and column (0-2): 0 2
O| |X
-----
 |X|
-----
 | |
-----
Player O, enter row and column (0-2): 2 0
O| |X
-----
 |X|
-----
O| |
-----
Player X, enter row and column (0-2): 2 2
O| |X
-----
 |X|
-----
O| |X
-----
Player X wins!
```

10. Build a bot which provides all the information related to text in search box

Program

```
from flask import Flask, request
import wikipedia

app = Flask(__name__)

@app.route('/')
def index():
    return '''
        <form action="/search" method="post">
            <input type="text" name="query" placeholder="Enter your search query">
            <input type="submit" value="Search">
        </form>
    '''

@app.route('/search', methods=['POST'])
def search():
    query = request.form['query']
    try:
        result = wikipedia.summary(query)
        return result
    except wikipedia.exceptions.DisambiguationError as e:
        # If the query is ambiguous, let the user know
        return f"Your query '{query}' is ambiguous. Please be more specific."
    except wikipedia.exceptions.PageError as e:
        # If the query doesn't match any page, let the user know
        return f"No information found for '{query}'."

if __name__ == '__main__':
    app.run(debug=True)
```

Output

```
* Serving Flask app "bot" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

11. Implement any game and demonstrate the game playing strategies
import random

Program

```
def guess_the_number():
    low, high = 1, 100
    correct_number = random.randint(low, high)
    attempts = 0

    print(f"Think of a number between {low} and {high}.")

    while True:
        guess = (low + high) // 2
        attempts += 1
        print(f"Is it {guess}?")

        if guess < correct_number:
            print("Too low.")
to implement any game and demonstrate the game playing strategies            low = guess + 1
        elif guess > correct_number:
            print("Too high.")
            high = guess - 1
        else:
            print(f"Correct! The number is {guess}. It took {attempts} attempts.")
            break

guess_the_number()
```

Output

```
Think of a number between 1 and 100.
Is it 50?
Too low.
Is it 75?
Too low.
Is it 88?
Too high.
Is it 81?
Too low.
Is it 84?
Too high.
Is it 82?
Too high.
Is it 81?
Correct! The number is 81. It took 7 attempts.
```