AI Practical Codes

Practical 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.

Code:-

```
# BFS function
def bfs(graph, start_node):
    visited = []  # List to keep track of visited nodes
    queue = [start_node]  # Start with the first node in the queue

    while queue:
        node = queue.pop(0)
        if node not in visited:
            print(node, end=" ")  # Print the node
            visited.append(node)
            queue.extend(graph[node])  # Add neighbors to the queue

# DFS function
def dfs(visited, graph, node):
    if node not in visited:
        print(node, end=" ")  # Print the node
        visited.add(node)  # Mark node as visited
        for neighbor in graph[node]:  # Visit all neighbors
            dfs(visited, graph, neighbor)

# Get user input for the graph
graph = {}
n = int(input("Enter number of nodes: "))  # Number of nodes

# Get the graph structure from the user
for _ in range(n):
    node = input("Enter node: ")
    neighbors = input(f"Enter neighbors of {node} (space-separated): ").split()
    graph[node] = neighbors

# Get the starting node for BFS
start_node_bfs = input("Enter starting node for BFS: ")

# Perform BFS
print("Breadth-First Search:")
```

```
bfs(graph, start_node_bfs)

# Get the starting node for DFS
start_node_dfs = input("\nEnter starting node for DFS: ")

# Initialize the visited set for DFS
visited = set()

# Perform DFS
print("Depth-First Search:")
dfs(visited, graph, start_node_dfs)
```

Input and Output:-

Enter number of nodes: 4

Enter node: A

Enter neighbors of A (space-separated): B C

Enter node: B

Enter neighbors of B (space-separated): D

Enter node: C

Enter neighbors of C (space-separated): D

Enter node: D

Enter neighbors of D (space-separated):

Enter starting node for BFS: A

Breadth-First Search:

A B C D

Enter starting node for DFS: A

Depth-First Search:

A B D C

Practical 2:- Implement A star (A*) Algorithm for any game search problem.

Code:-

```python
import copy

# Define the goal state
goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
goal_pos = {goal_state[i][j]: (i, j) for i in range(3) for j in range(3)}


def find_blank(state):
    """Find the blank tile's (0) position in the state."""
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j


def heuristic(state):
    """Calculate the Manhattan distance as the heuristic."""
    dist = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                x, y = goal_pos[state[i][j]]
                dist += abs(i - x) + abs(j - y)
    return dist


def is_goal(state):
    """Check if the current state is the goal state."""
    return state == goal_state


def get_neighbors(state):
    """Generate neighboring states by moving the blank tile."""
    neighbors = []
    i, j = find_blank(state)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for di, dj in directions:
        new_i, new_j = i + di, j + dj
        if 0 <= new_i < 3 and 0 <= new_j < 3:
            new_state = copy.deepcopy(state)
            new_state[i][j], new_state[new_i][new_j] = new_state[new_i][new_j],
new_state[i][j]
            neighbors.append(new_state)

    return neighbors
```

```
def a_star(start_state):
    """Perform the A* search algorithm to solve the puzzle."""
    open_list = [(start_state, 0)]
    closed_list = set()

    while open_list:
        # Sort by f = g + h
        open_list.sort(key=lambda x: x[1] + heuristic(x[0]))
        current_state, g_value = open_list.pop(0)

        if is_goal(current_state):
            return current_state, g_value  # Solution found

        closed_list.add(tuple(tuple(row) for row in current_state))

        for neighbor in get_neighbors(current_state):
            neighbor_tuple = tuple(tuple(row) for row in neighbor)
            if neighbor_tuple not in closed_list:
                open_list.append((neighbor, g_value + 1))

    return None  # No solution found


def get_user_input():
    """Get the initial puzzle state from the user."""
    print("Enter the initial state of the puzzle (3x3 grid, use 0 for the blank tile):")
    initial_state = []
    for i in range(3):
        row = input(f"Enter row {i + 1} (space-separated integers): ").split()
        initial_state.append([int(x) for x in row])
    return initial_state


def main():
    choice = input("Do you want to provide an initial state? (y/n): ").strip().lower()

    if choice == 'y':
        initial_state = get_user_input()
    else:
        print("Using default example: [[1, 2, 3], [4, 0, 6], [7, 5, 8]]")
        initial_state = [[1, 2, 3], [4, 0, 6], [7, 5, 8]]

    solution, moves = a_star(initial_state)
    if solution:
```

```
        print("Solution found in", moves, "moves:")
        for row in solution:
            print(row)
    else:
        print("No solution found.")
```

Input and Output:-

```
C:\Users\champ\bluej\Python37\python.exe C:/Users/champ/PycharmProjects/Python_Projects/Astar.py
Do you want to provide an initial state? (y/n): y
Enter the initial state of the puzzle (3x3 grid, use 0 for the blank tile):
Enter row 1 (space-separated integers): 1 2 3
Enter row 2 (space-separated integers): 4 0 6
Enter row 3 (space-separated integers): 7 5 8
Solution found in 2 moves:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

```
C:\Users\champ\bluej\Python37\python.exe C:/Users/champ/PycharmProjects/Python_Projects/Astar.py
Do you want to provide an initial state? (y/n): n
Using default example: [[1, 2, 3], [4, 0, 6], [7, 5, 8]]
Solution found in 2 moves:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

Practical 3:- Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

Code:-

```
# Taking the number of queens as input from the user
print("Enter the number of queens:")
N = int(input())

# Creating an NxN chessboard initialized with 0s
board = [[0] * N for _ in range(N)]

# Function to check if a queen can be placed on board[i][j]
def is_safe(i, j):
    # Check this row on the left side
    for k in range(j):
        if board[i][k] == 1:
            return False
```

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

    # Check lower diagonal on the left side
    for k, l in zip(range(i, N), range(j, -1, -1)):
        if board[k][l] == 1:
            return False

    return True

# Function to solve the N-Queens problem using backtracking
def solve_n_queens(col):
    # If all queens are placed, return True
    if col >= N:
        return True

    # Try placing a queen in each row of the current column
    for i in range(N):
        if is_safe(i, col):
            # Place the queen
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solve_n_queens(col + 1):
                return True

            # If placing queen in board[i][col] doesn't lead to a solution,
            # then remove the queen (backtrack)
            board[i][col] = 0

    return False

# Start solving the problem from the first column
if solve_n_queens(0):
    # Print the solution
    for row in board:
        print(row)
else:
    print("No solution exists for", N, "queens.")
```

Input and Output:-

```
C:\Users\champ\bluej\Python37\python.exe C:/Users/champ/PycharmProjects/Python_Projects/CSP.py
Enter the number of queens:
4
[0, 0, 1, 0]
[1, 0, 0, 0]
[0, 0, 0, 1]
[0, 1, 0, 0]
```

Practical 4:- Implement Greedy search algorithm for any of the following application:

a)Prim's Minimal Spanning Tree Algorithm

b)Kruskal's Minimal Spanning Tree Algorithm

C)Dijkstra's Minimal Spanning Tree Algorithm

Code(Prims Algorithm):-

```
import sys

class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for _ in range(vertices)] for _ in range(vertices)]

    def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(f"{parent[i]} - {i} \t {self.graph[i][parent[i]]}")

    def minKey(self, key, mstSet):
        min_value = sys.maxsize
        min_index = -1

        for v in range(self.V):
            if key[v] < min_value and not mstSet[v]:
                min_value = key[v]
                min_index = v
        return min_index

    def primMST(self):
        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        key[0] = 0
        mstSet = [False] * self.V
        parent[0] = -1
```

```
        for _ in range(self.V):
            u = self.minKey(key, mstSet)
            mstSet[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and not mstSet[v] and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

        self.printMST(parent)

# Input graph from the user
def inputGraph():
    vertices = int(input("Enter number of vertices: "))
    g = Graph(vertices)

    print("Enter adjacency matrix (0 for no connection):")
    for i in range(vertices):
        row = list(map(int, input(f"Row {i + 1}: ").split()))
        g.graph[i] = row

    return g

g = inputGraph()
g.primMST()
```

Input and Output(Prims Algorithm):-

Enter number of vertices: 5

Enter adjacency matrix (0 for no connection):

Row 1: 0 2 0 6 0

Row 2: 2 0 3 8 5

Row 3: 0 3 0 0 7

Row 4: 6 8 0 0 9

Row 5: 0 5 7 9 0

Code(Kruskal's Algorithm):-

```
class Graph:
    def __init__(self, vertices):
```

```
    self.V = vertices
    self.graph = []

def addEdge(self, u, v, w):
    self.graph.append([u, v, w])

def find(self, parent, i):
    if parent[i] != i:
        parent[i] = self.find(parent, parent[i])
    return parent[i]

def union(self, parent, rank, x, y):
    if rank[x] < rank[y]:
        parent[x] = y
    elif rank[x] > rank[y]:
        parent[y] = x
    else:
        parent[y] = x
        rank[x] += 1

def KruskalMST(self):
    result = []  # Store the final MST
    i, e = 0, 0  # i is index for sorted edges, e is index for result
    self.graph = sorted(self.graph, key=lambda item: item[2])

    parent, rank = [], []
    for node in range(self.V):
        parent.append(node)
        rank.append(0)

    while e < self.V - 1:
        u, v, w = self.graph[i]
        i += 1
        x, y = self.find(parent, u), self.find(parent, v)

        if x != y:
            e += 1
            result.append([u, v, w])
            self.union(parent, rank, x, y)

    # Printing the MST edges and calculating the total cost
    minimumCost = 0
    print("Edges in MST:")
    for u, v, weight in result:
        minimumCost += weight
        print(f"{u} -- {v} == {weight}")
    print("Minimum Spanning Tree Cost:", minimumCost)
```

```
# Input graph from the user
def inputGraph():
    vertices = int(input("Enter number of vertices: "))
    edges = int(input("Enter number of edges: "))
    g = Graph(vertices)

    print("Enter each edge (u v w):")
    for _ in range(edges):
        u, v, w = map(int, input().split())
        g.addEdge(u, v, w)

    return g

# Driver code
g = inputGraph()
g.KruskalMST()
```

Input and Output(Kruskal's Algorithm):-

Enter number of vertices: 4

Enter number of edges: 5

Enter each edge (u v w):

0 1 10

0 2 6

0 3 5

1 3 15

2 3 4

Edges in MST:

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Spanning Tree Cost: 19

Code(Dijakstra's Algorithm):-

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for _ in range(vertices)] for _ in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
            print(f"{node} \t\t {dist[node]}")

    def minDistance(self, dist, sptSet):
        min_value = float('inf')
        min_index = -1

        for v in range(self.V):
            if dist[v] < min_value and not sptSet[v]:
                min_value = dist[v]
                min_index = v
        return min_index

    def dijkstra(self, src):
        dist = [float('inf')] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for _ in range(self.V):
            u = self.minDistance(dist, sptSet)
            sptSet[u] = True

            for v in range(self.V):
                if (self.graph[u][v] > 0 and not sptSet[v] and
                        dist[v] > dist[u] + self.graph[u][v]):
                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

# Input graph from the user
def inputGraph():
    vertices = int(input("Enter number of vertices: "))
    g = Graph(vertices)

    edges = int(input("Enter number of edges: "))
    print("Enter each edge (u v w):")
    for _ in range(edges):
        u, v, w = map(int, input().split())
```

```
        g.graph[u][v] = w
        g.graph[v][u] = w

    return g

g = inputGraph()
src = int(input("Enter the source vertex: "))
g.dijkstra(src)
```

Input and Output(Dijakstra Algorithm):-

Enter number of vertices: 5

Enter number of edges: 6

Enter each edge (u v w):

0 1 10

0 2 3

1 2 1

1 3 2

2 3 8

3 4 7

Enter the source vertex: 0

Vertex Distance from Source

0	0
1	4
2	3
3	6
4	13

Practical 5:- Develop an elementary chatbot for any suitable customer interaction application.

Code:-

```
def chatbot():
    print("Hi there! I am a simple shopping chatbot. Type 'bye' to exit.")
    responses = {
        "hi": "Hello there! How can I assist you today?",
        "hello": "Hello there! How can I assist you today?",
        "hey": "Hello there! How can I assist you today?",
        "bye": "Goodbye! Have a great day!",
        "i need": "What kind of item do you need?",
        "what products do you have?": "We have laptops, smartphones, and headphones. What
are you interested in?",
        "can you recommend": "Sure! What kind of recommendation are you looking for?",
        "i am": "Hello! How are you feeling today?",
        "feeling": "Good to hear that!"
    }

    while True:
        user_input = input("> ").strip().lower()
        if user_input in responses:
            print(responses[user_input])
            if user_input == "bye":
                break
        else:
            print("I'm sorry, I didn't understand that. Can you rephrase?")



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

Input and Output:-

```
C:\Users\champ\bluej\Python37\python.exe C:/Users/champ/PycharmProjects/Python_Projects/Chatbot.py
Hi there! I am a simple shopping chatbot. Type 'bye' to exit.
> hi
Hello there! How can I assist you today?
> what products do you have?
We have laptops, smartphones, and headphones. What are you interested in?
> can you recommend
Sure! What kind of recommendation are you looking for?
> bye
Goodbye! Have a great day!
```