Artificial Intelligence Practical File

Prepared by: V

Table of Contents

DFS Algorithm

```
def dfs(graph, start, visited=None):

    if visited is None:

        visited = set()

    visited.add(start)

    print(start, end=' ')

    for neighbor in graph[start]:

        if neighbor not in visited:

            dfs(graph, neighbor, visited)


graph = {

    'A': ['B', 'C'],

    'B': ['D', 'E'],

    'C': ['F'],

    'D': [],

    'E': ['F'],

    'F': []

}


print("DFS Traversal:")

dfs(graph, 'A')
```

BFS Algorithm

```
from collections import deque


def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            print(vertex, end=' ')
            visited.add(vertex)
            queue.extend(graph[vertex])


graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}


print("\nBFS Traversal:")
bfs(graph, 'A')
```

A* Algorithm Algorithm

```
from queue import PriorityQueue


def heuristic(a, b):

    return abs(a[0] - b[0]) + abs(a[1] - b[1])


def a_star(start, goal):

    frontier = PriorityQueue()

    frontier.put((0, start))

    came_from = {}

    cost_so_far = {start: 0}


    while not frontier.empty():

        _, current = frontier.get()


        if current == goal:

            break


        for dx, dy in [(1,0), (0,1), (-1,0), (0,-1)]:

            neighbor = (current[0] + dx, current[1] + dy)

            if 0 <= neighbor[0] < 5 and 0 <= neighbor[1] < 5:

                new_cost = cost_so_far[current] + 1

                if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:

                    cost_so_far[neighbor] = new_cost

                    priority = new_cost + heuristic(goal, neighbor)

                    frontier.put((priority, neighbor))
```

```
                    came_from[neighbor] = current

    return came_from

start = (0, 0)
goal = (4, 4)
path = a_star(start, goal)
print("\nPath traced using A* (some output):", path)
```

Selection Sort Algorithm

```
def selection_sort(arr):

    for i in range(len(arr)):

        min_idx = i

        for j in range(i+1, len(arr)):

            if arr[min_idx] > arr[j]:

                min_idx = j

        arr[i], arr[min_idx] = arr[min_idx], arr[i]


arr = [64, 25, 12, 22, 11]

selection_sort(arr)

print("\nSorted array:", arr)
```

Prim's MST Algorithm

```
import sys


def prim(graph):
    selected = [False] * len(graph)
    no_edge = 0
    selected[0] = True

    print("Edge : Weight")
    while no_edge < len(graph) - 1:
        minimum = sys.maxsize
        x = 0
        y = 0
        for i in range(len(graph)):
            if selected[i]:
                for j in range(len(graph)):
                    if (not selected[j]) and graph[i][j]:
                        if minimum > graph[i][j]:
                            minimum = graph[i][j]
                            x = i
                            y = j
        print(f"{x} - {y}: {graph[x][y]}")
        selected[y] = True
        no_edge += 1


graph = [
```

```
    [0, 2, 0, 6, 0],

    [2, 0, 3, 8, 5],

    [0, 3, 0, 0, 7],

    [6, 8, 0, 0, 9],

    [0, 5, 7, 9, 0]

]


prim(graph)
```

Kruskal's MST Algorithm

```
class DisjointSet:

    def __init__(self, n):

        self.parent = list(range(n))

        self.rank = [0] * n



    def find(self, u):

        if self.parent[u] != u:

            self.parent[u] = self.find(self.parent[u])

        return self.parent[u]



    def union(self, u, v):

        root_u = self.find(u)

        root_v = self.find(v)

        if root_u != root_v:

            if self.rank[root_u] > self.rank[root_v]:

                self.parent[root_v] = root_u

            else:

                self.parent[root_u] = root_v

                if self.rank[root_u] == self.rank[root_v]:

                    self.rank[root_v] += 1



def kruskal(n, edges):

    ds = DisjointSet(n)

    mst = []

    edges.sort(key=lambda edge: edge[2])
```

```
    for u, v, w in edges:
        if ds.find(u) != ds.find(v):
            ds.union(u, v)
            mst.append((u, v, w))

    return mst


edges = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]
mst = kruskal(4, edges)
print("\nKruskal's MST:")
for u, v, w in mst:
    print(f"{u} - {v}: {w}")
```

Dijkstra's Algorithm Algorithm

```
import heapq


def dijkstra(graph, start):
    queue = [(0, start)]
    dist = {start: 0}
    while queue:
        (cost, node) = heapq.heappop(queue)

        if node in dist and dist[node] < cost:
            continue

        for neighbor, weight in graph[node]:
            new_cost = cost + weight
            if neighbor not in dist or new_cost < dist[neighbor]:
                dist[neighbor] = new_cost
                heapq.heappush(queue, (new_cost, neighbor))

    return dist


graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('A', 1), ('C', 2), ('D', 5)],
    'C': [('A', 4), ('B', 2), ('D', 1)],
    'D': [('B', 5), ('C', 1)]
}
```

```
print("\nShortest path from A:", dijkstra(graph, 'A'))
```

N-Queens Backtracking Algorithm

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

    for i in range(col):

        if board[row][i] == 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, n, 1), range(col, -1, -1)):

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

            return False

    return True


def solve(board, col, n):

    if col >= n:

        return True

    for i in range(n):

        if is_safe(board, i, col, n):

            board[i][col] = 1

            if solve(board, col + 1, n):

                return True

            board[i][col] = 0

    return False


def print_solution(board):

    for row in board:
```

```python
        print(' '.join(['Q' if x else '.' for x in row]))


n = 4

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

if solve(board, 0, n):

    print("\nSolution for N-Queens:")

    print_solution(board)

else:

    print("\nSolution does not exist")
```

Graph Coloring Backtracking Algorithm

```
def is_safe(graph, color, c, v):

    for i in graph[v]:

        if color[i] == c:

            return False

    return True


def graph_coloring(graph, m, color, v):

    if v == len(graph):

        return True

    for c in range(1, m + 1):

        if is_safe(graph, color, c, v):

            color[v] = c

            if graph_coloring(graph, m, color, v + 1):

                return True

            color[v] = 0

    return False


def print_solution(color):

    print("Graph coloring solution:", color)


graph = [

    [1, 2, 3],

    [0, 2],

    [0, 1],

    [0]
```

```
]

m = 3

color = [0] * len(graph)

if graph_coloring(graph, m, color, 0):

    print_solution(color)

else:

    print("Solution does not exist")
```

Chatbot Algorithm

```
import random


def chatbot_response(user_input):
    responses = {
        'hello': 'Hi there! How can I assist you today?',
        'bye': 'Goodbye! Have a great day!',
        'help': 'Sure, what can I help you with?',
        'default': 'Sorry, I did not understand that.'
    }
    return responses.get(user_input.lower(), responses['default'])


while True:
    user_input = input("You: ")
    if user_input.lower() == 'bye':
        print("Chatbot:", chatbot_response(user_input))
        break
    print("Chatbot:", chatbot_response(user_input))
```

Salesforce Application (Apex) Algorithm

```
// Example Apex Class in Salesforce

public class HelloWorld {

    public static String greet(String name) {

        return 'Hello, ' + name + '!';

    }

}


// Call the method from Apex class

String result = HelloWorld.greet('V');

System.debug(result);
```

Salesforce Cloud App Algorithm

```
// Simple Salesforce Cloud Application (Mini Project) Example

public class AccountManager {

    public void createAccount(String name, String type) {

        Account acc = new Account();

        acc.Name = name;

        acc.Type = type;

        insert acc;

    }

}


// Instantiate the class and create an account

AccountManager manager = new AccountManager();

manager.createAccount('Acme Corp', 'Customer');
```

Reflex Agent Algorithm

```
# Simple Reflex Agent (AI)

def reflex_agent(state):

    if state == 'hungry':

        return 'eat'

    elif state == 'tired':

        return 'sleep'

    else:

        return 'do nothing'


state = 'hungry'

action = reflex_agent(state)

print("Reflex Agent action:", action)
```

Alpha-Beta Pruning Algorithm

```
# Simple Alpha-Beta Pruning Example

def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player):

    if depth == 0 or not node:

        return node



    if maximizing_player:

        max_eval = -float('inf')

        for child in node:

            eval = alpha_beta_pruning(child, depth-1, alpha, beta, False)

            max_eval = max(max_eval, eval)

            alpha = max(alpha, eval)

            if beta <= alpha:

                break

        return max_eval

    else:

        min_eval = float('inf')

        for child in node:

            eval = alpha_beta_pruning(child, depth-1, alpha, beta, True)

            min_eval = min(min_eval, eval)

            beta = min(beta, eval)

            if beta <= alpha:

                break

        return min_eval



# Example of alpha-beta pruning on a tree
```

```python
node_tree = [3, 12, 8, 2]

result = alpha_beta_pruning(node_tree, 3, -float('inf'), float('inf'), True)

print("\nAlpha-Beta Pruning Result:", result)
```

Forward Chaining Algorithm

```
# Forward Chaining Example

knowledge_base = {'A': True, 'B': False, 'C': None}



def forward_chaining(kb):

    if kb['A'] and kb['B'] is False:

        kb['C'] = True



    return kb



result = forward_chaining(knowledge_base)

print("\nForward Chaining Result:", result)
```