INDEX

Sr.no	DATE	Practical	Signature
1	4/01/2025	Design an expert system for responding to the patient query for identifying the FLU	
2	8/01/2025	Design an expert system using AIML for Restaurant Recommendation	
3	22/01/2025	Design an E-commerce chat bot using AIML	
4	12/02/2025	Design a game bot [rock, paper, scissor bot] using aim	
5 (A)	20/01/2025	Implement Depth-First Search and Breadth First Search algorithm	
5 (B)	12/02/2025	Implement Back-tracking algorithm	
6 (A)	3/03/2025	Implement Iterative deepening search	
6 (B)	3/03/2025	Implement Uniform Cost Search	
7 (A)	17/02/2025	Implement Greedy Algorithm	
7 (B)	17/02/2025	Implement Best First Search Algorithm	
8 (A)	24/02/2025	Implement Beam Search algorithm	
8 (B)	24/02/2025	Implement Branch and Bound algorithm	
9	24/02/2025	Implement A* algorithm	
10	20/01/2025	Write a program to implement rule based system (automatic sprinkler RBS)	

Practical 1

AIM: Design an expert system for responding TB patient queries for identifying the FLU

CODE:

```
def ask_question(question):
    response=input(question+"(Yes/No)").strip().lower()
    while response not in["yes","no"]:
        print("Enter yes/no only")
        response=input(question+"(Yes/No)").strip().lower()
    return response=="yes"


def collect_symptoms():
    print("Enter your symptoms in yes/no")
    symptoms={}
    symptoms["fever"]=ask_question("Do you have fever?")
    symptoms["cold"]=ask_question("Do you have cold?")
    symptoms["sore_thorat"]=ask_question("Do you have sore thorat?")
    symptoms["running_nose"]=ask_question("Do you have running nose?")
    symptoms["body_ache"]=ask_question("Do you have body ache?")
    symptoms["cough"]=ask_question("Do you have cough?")
    symptoms["fatigue"]=ask_question("Do you have fatigue?")
    symptoms["head_ache"]=ask_question("Do you have head ache?")
    return symptoms


def evaluate(symptoms):
    if symptoms["fever"] and symptoms["cough"] and symptoms["fatigue"]:
            if symptoms["head_ache"] and symptoms["cold"]:
```

```
            return "You have to visit doctor"
        return "Go to doctor"
    elif symptoms["running_nose"] and symptoms["sore_thorat"]:
        return "Take good amount of rest"
    elif symptoms["body_ache"] and not symptoms["fever"]:
        return "Rest is nesscary"
    else:
        return "your symptoms are unidentiable.You are well"


def play():
    print("welcome to Flu expert system. Let evaluate your symptoms")
    symptoms=collect_symptoms()
    diagonosis=evaluate(symptoms)
    print("\n Diagonosis")
    print(diagonosis)


play()
```

OUTPUT:

```
welcome to Flu expert system. Let evaluate your symptoms
Enter your symptoms in yes/no
Do you have fever?(Yes/No)yes
Do you have cold?(Yes/No)no
Do you have sore thorat?(Yes/No)no
Do you have running nose?(Yes/No)no
Do you have body ache?(Yes/No)no
Do you have cough?(Yes/No)yes
Do you have fatigue?(Yes/No)yes
Do you have head ache?(Yes/No)no

 Diagonosis
Go to doctor
```

Practical 2

AIM: Design an expert system using AIML for Restaurant Recommendation

CODE:

main.py

```python
import aiml

kernel= aiml.Kernel()

kernel.learn("rest.aiml")

print("recommendation is running")

print("type quit to exit")

while True:

    user_input = input("You:")

    if(user_input=="quit"):

        print("Bot: GoodBye!")

        break

    response=kernel.respond(user_input)

    print("bot:" , response)
```

rest.aiml

```xml
<aiml version="2.0">

<category>
<pattern>RECOMMEND RESTAURANT</pattern>
<template> What type are you looking(chinese , indian , italian , thai )
</template>
</category>

<category>
<pattern>i want * cuisine</pattern>
```

```
<template> Here are some popular <star/> restaurant
<think>
 <set name="cuisine" ><star/></set>
</think>
<condition name="cuisine">
<li value="italian"> spagetti , rissoto</li>
<li value="indian"> Puranpoli , Chicken curry </li>
<li value="thai"> Mango rice </li>
<li value="chinese"> Noodles ,Soup</li>
</condition>
</template>
</category>


<category>
<pattern>THANK YOU</pattern>
<template> ENJOY YOUR MEAL  </template>
</category>


</aiml>
```

OUTPUT:

```
Loading rest.aiml...done (0.12 seconds)
recommendation is running
type quit to exit
You:  Recommend Restaurant
bot: What type of cuisine are you looking for? (Chinese, Indian, Italian, Thai)
You: I want indian cuisine
bot: Here are some popular indian restaurants:  Puran Poli, Chicken Curry
You: I want chinese cuisine
bot: Here are some popular chinese restaurants:  Noodles, Soup
You: Thank You
bot: Enjoy your meal!
You: quit
Bot: GoodBye!
```

Practical 3

AIM: Design an Ecommerce chat bot using AIML

CODE:

main.py

```
import aiml
import os
def run():
    kernel = aiml.Kernel()
    kernel.learn("p3aiml.aiml")
    print("E-commerence chatbot is ready to run")


    while True:
        user_input = input("YOU:")
        if user_input.lower() == "exit":
            print("ChatBot: Thank you for visiting! Come back soon.")
            break
        response = kernel.respond(user_input)
        print("ChatBot:", response)


run()
```

p3aiml.aiml

```
<aiml version="2.0">
  <category>
    <pattern>HELLO</pattern>
    <template>Welcome to our e-commerce store.</template>
  </category>

  <category>
    <pattern>BUY *</pattern>
    <template>
      Great choice! We have a wide range of <star/>. Would you like some suggestions?
    </template>
  </category>

  <category>
    <pattern>YES</pattern>
    <template>
      Can you specify your preferences (color, brand, budget)?
    </template>
  </category>

  <category>
    <pattern>NO</pattern>
    <template>
```

```
            No problem! Let me know if there's anything else I can help with.
        </template>
    </category>

    <category>
        <pattern>WHAT ARE YOUR OFFERS</pattern>
        <template>
            We currently have discounts on electronics and fashion. Would you like to explore?
        </template>
    </category>

    <category>
        <pattern>EXIT</pattern>
        <template>
            Thank you for visiting.
        </template>
    </category>
</aiml>
```

OUTPUT:

```
Loading p3aiml.aiml...done (0.10 seconds)
E-commerence chatbot is ready to run
YOU:Hello
ChatBot: Welcome to our e-commerce store.
YOU:Buy Shoes
ChatBot: Great choice! We have a wide range of Shoes. Would you like some suggestions?
YOU:Yes
ChatBot: Can you specify your preferences (color, brand, budget)?
YOU:No
ChatBot: No problem! Let me know if there's anything else I can help with.
YOU:What are your offers
ChatBot: We currently have discounts on electronics and fashion. Would you like to explore?
YOU:Exit
ChatBot: Thank you for visiting! Come back soon.
```

Practical 4

AIM: Design a game bot [rock,paper.scissor] using aiml

CODE:

```
import aiml

def play():
    ker = aiml.Kernel()
    ker.learn("p4aiml.aiml")
    print("Game bot: type rock paper scissor tp play. Type quit to exit")
    while True:
        user_input = input("You: ").strip().upper()
        if user_input == "QUIT":
            print ("Good day")
            break
        elif user_input in ["ROCK","PAPER","SCISSOR"]:
            response = ker.respond(user_input)
            print("Game Bot: ",response)
        else:
            print("I dont understand the command")

play()
```

p4aim.aiml =

```
<aiml version="2.0">
        <category>
                <pattern>ROCK</pattern>
                <template>
                        <think><set name="userMove">rock</set></think>
                        <random>
                                <li>Paper. I win</li>
                                <li>Rock. It's a tie</li>
                                <li>Scissors. You won</li>
                        </random>
                </template>
        </category>

        <category>
                <pattern>PAPER</pattern>
                <template>
                        <think><set name="userMove">paper</set></think>
                        <random>
```

```
                        <li>Paper. It's a tie</li>
                        <li>Rock. You win</li>
                        <li>Scissors. I won</li>
                    </random>
                </template>
            </category>

            <category>
                <pattern>SCISSORS</pattern>
                <template>
                    <think><set name="userMove">scissors</set></think>
                    <random>
                        <li>Paper. You won</li>
                        <li>Rock. I win</li>
                        <li>Scissors. It's a tie</li>
                    </random>
                </template>
            </category>

            <category>
                <pattern>HELLO</pattern>
                <template>
                    Welcome to the game.
                </template>
            </category>

            <category>
                <pattern>EXIT</pattern>
                <template>
                    Exiting the game.
                </template>
            </category>
        </aiml>
```

OUTPUT:

```
Loading p4aiml.aiml...done (0.13 seconds)
Game bot: type rock paper scissors to play. Type quit to exit.
You: Paper
Game Bot: Paper. It's a tie
You: Rock
Game Bot: Scissors. You won
You: Scissors
Game Bot: Rock. I win
You: Quit
Good day
```

Practical 5

A] AIM: Implement DFS and BFS algorithm

CODE:

```
from collections import defaultdict, deque

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

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

    def dfs(self, start, visited=None):
        if visited is None:
            visited = set()
        visited.add(start)
        print(start)

        for neighbour in self.graph[start]:
            if neighbour not in visited:
                self.dfs(neighbour, visited)

    def bfs(self, start):
        visited = set()
        queue = deque([start])
        visited.add(start)

        while queue:
            vertex = queue.popleft()
            print(vertex)

            for neighbour in self.graph[vertex]:
                if neighbour not in visited:
                    visited.add(neighbour)
                    queue.append(neighbour)

g = Graph()
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 4)
```

```
g.add_edge(2, 3)

print("DFS:")
g.dfs(0

print("\nBFS:")
g.bfs(0)
```

OUTPUT:

```
DFS:
0
1
4
2
3

BFS:
0
1
2
4
3
```

B] AIM: Implement Back Tracking algorithm

CODE:

```
def print_solution(board):
    for row in board:
        print(" ".join("Q" if col else "." for col in row))
    print()

def is_safe(board,row,col,n):
    for i in range(row):
        if board[i][col]:
            return False

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

    return True

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

    success = False
```

```
    for col in range(n):
        #print(col)
        if is_safe(board,row,col,n):
            board[row][col]=1
            success=solve(board,row+1,n) or success
            board[row][col]=0 #backtrack
    return success

n=4
board=[[0]*n for _ in range(n)]
print("solution to ",n,"Queen problem")
solve(board,0,n)
```

OUTPUT :

```
solution to  4 Queen problem
Q . . .
. Q . .
. . Q .
. . . Q

Q . . .
. . Q .
. . . Q
. Q . .

Q . . .
. . . Q
. Q . .
. . Q .

. Q . .
. . Q .
Q . . .
. . . Q

. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. Q . .
. . . Q

. . Q .
Q . . .
. . . Q
. Q . .

True
```

Practical 6

A] AIM: Implement Iterative deepening search

CODE:

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

    def add_edge(self, node, neighbor):
        if node not in self.edges:
            self.edges[node] = []
        self.edges[node].append(neighbor)

    def get_neighbors(self, node):
        return self.edges.get(node, [])

def depth_limited_search(graph, start, goal, limit):
    if start == goal:
        return [start]
    if limit <= 0:
        return None
    for neighbor in graph.get_neighbors(start):
        path = depth_limited_search(graph, neighbor, goal, limit - 1)
        if path:
            return [start] + path
    return None
```

```
def iterative_deepening_search(graph, start, goal):
    depth = 0
    while True:
        path = depth_limited_search(graph, start, goal, depth)
        if path:
            return path
        depth += 1

# Example usage for IDS
graph = Graph()
graph.add_edge('A', 'B')
graph.add_edge('A', 'C')
graph.add_edge('B', 'D')
graph.add_edge('C', 'E')
graph.add_edge('D', 'F')

print("IDS Path:", iterative_deepening_search(graph, 'A', 'F'))
```

OUTPUT:

```
IDS Path: ['A', 'B', 'D', 'F']
```

B] AIM: Implement Uniform cost search

CODE:

```
import heapq


def uniform_cost_search(graph, start, goal):
    priority_queue = []
    heapq.heappush(priority_queue, (0, start, [start]))  # (cost, node, path)
    visited = set()


    while priority_queue:
        cost, current_node, path = heapq.heappop(priority_queue)


        if current_node in visited:
            continue
        visited.add(current_node)


        if current_node == goal:
            return path, cost


        for neighbor, edge_cost in graph.get_neighbors(current_node):
            if neighbor not in visited:
                heapq.heappush(priority_queue, (cost + edge_cost, neighbor, path + [neighbor]))


    return None, float('inf')  # Return None if no path is found


class WeightedGraph:
```

```
    def __init__(self):
        self.edges = {}

    def add_edge(self, node, neighbor, cost):
        if node not in self.edges:
            self.edges[node] = []
        self.edges[node].append((neighbor, cost))

    def get_neighbors(self, node):
        return self.edges.get(node, [])

# Example usage for UCS
weighted_graph = WeightedGraph()
weighted_graph.add_edge('A', 'B', 1)
weighted_graph.add_edge('A', 'C', 4)
weighted_graph.add_edge('B', 'D', 2)
weighted_graph.add_edge('C', 'E', 1)
weighted_graph.add_edge('D', 'F', 5)
weighted_graph.add_edge('E', 'F', 1)

path, cost = uniform_cost_search(weighted_graph, 'A', 'F')
print("UCS Path:", path, "with cost:", cost)
```

OUTPUT:

```
UCS Path: ['A', 'C', 'E', 'F'] with cost: 6
```

Practical 7

A] AIM: Implement Greedy Algorithm

CODE:

```
import heapq
class Graph:
    def __init__(self):
        self.graph = {}
    def add_Edge (self,start,end):
        if start not in self.graph:
            self.graph[start]=[]
        self.graph[start].append(end)
    def greedy(self,start,goal,heuristic):
        current = start
        path = [current]
        while current != goal:
            if current not in self.graph:
                print("No path found")
                return []
            current = min (self.graph[current],key=lambda node: heuristic[node])
            path.append(current)
        return path


g = Graph()
```

```
g.add_Edge("A","B")
g.add_Edge("A","C")
g.add_Edge("B","D")
g.add_Edge("C","D")
g.add_Edge("C","E")
g.add_Edge("D","E")

heuristic ={
    "A":7, "B":6, "C":2, "D":1, "E":0
    }

start = "A"
goal = "E"
print("Greedy algorithm OUTPUT")
print(g.greedy(start,goal,heuristic))
```

OUTPUT:

```
Greedy algorithm:
['A', 'C', 'E']
```

B] AIM : Implement BEST FIRST SEARCH Algorithm

Code:

```
import heapq
class Graph:
    def __init__(self):
        self.graph = {}
    def add_Edge (self,start,end):
        if start not in self.graph:
            self.graph[start]=[]
        self.graph[start].append(end)
    def greedy(self,start,goal,heuristic):
        current = start
        path = [current]
        while current != goal:
            if current not in self.graph:
                print("No path found")
                return []
            current = min (self.graph[current],key=lambda node: heuristic[node])
            path.append(current)
        return path

    def bfs(self,start,goal,heuristic):
            pq = []
            heapq.heappush(pq,(heuristic[start],start))
```

```
        visited = set()
        path = []

        while pq:
            _,current = heapq.heappop(pq)
            if current in visited:
                continue
            path.append(current)
            visited.add(current)

            if current == goal:
                return path

            if current in self.graph:
                for neighbour in self.graph[current]:
                    if neighbour not in visited:
                        heapq.heappush(pq,(heuristic[neighbour],neighbour))
        print("No path found")
        return []

g = Graph()
g.add_Edge("A","B")
g.add_Edge("A","C")
g.add_Edge("B","D")
```

```
g.add_Edge("C","D")

g.add_Edge("C","E")

g.add_Edge("D","E")


heuristic ={

    "A":7, "B":6, "C":2, "D":1, "E":0

    }


start = "A"

goal = "E"


print ("Best First Search Algorithm")

print (g.bfs(start,goal,heuristic))
```

OUTPUT:

```
Best First Search Algorithm
['A', 'C', 'E']
```

Practical 8

A] Aim: Implement beam search algorithn

CODE:

```
import heapq

def beam_search(start_state, goal_test, successors, beam_width):
    # Initialize the beam with the start state
    beam = [(start_state, 0)] # (state, cost)

    while beam:
        # List of nodes at the current level
        next_beam = []

        for state, cost in beam:
            if goal_test(state):
                return state

            # Get the successor states and their costs
            successors_list = successors(state)

            for successor, successor_cost in successors_list:
                total_cost = cost + successor_cost
                next_beam.append((successor, total_cost))

        # Keep the top `beam_width` nodes from the next level
        beam = heapq.nsmallest(beam_width, next_beam, key=lambda x: x[1])

    return None # Goal is not reachable

# Example usage
def successors(state):
    # Define how to get the next state from current state
    return [(state + 1, 1), (state * 2, 1)]

def goal_test(state):
    return state == 10

# Beam search with beam width 2
result = beam_search(1, goal_test, successors, beam_width=2)
print(f"Found goal state: {result}")
```

OUTPUT:

```
Found goal state: 10
```

B] AIM: Implement branch and bound algorithm

CODE =

```
import heapq

def branch_and_bound(start_state, goal_test, successors, heuristic):

    # Priority queue for states to be explored

    queue = [(0 + heuristic(start_state), 0, start_state)]# (cost + heuristic, cost,
state)

    visited = set()

    while queue:

        _, cost, state = heapq.heappop(queue)

        #print(cost, state)

        if goal_test(state):

            return state

        if state in visited:

            continue

        visited.add(state)

        #print(visited)

        # Explore successor states

        for successor, step_cost in successors(state):

            total_cost = cost + step_cost

            #print(total_cost)
```

```
            heapq.heappush(queue, (total_cost + heuristic(successor), total_cost,
successor))
            #print(queue)
    return None # Goal is not reachable


# Example usage
def successors(state):
    # Define how to get the next state from current state
    #print((state + 1, 1), (state * 2, 1))
    return [(state + 1, 1), (state * 2, 1)]


def goal_test(state):
    #print(state==10)
    return state == 10


def heuristic(state):
    # Heuristic: Simple difference to goal (straight-line distance)
    #print (abs(state - 10))
    return abs(state - 10)


# Branch and Bound search
result = branch_and_bound(1, goal_test, successors, heuristic)
```

```python
print(f"Found goal state: {result}")
```

OUTPUT:

```
Found goal state: 10
```

Practical 9

AIM: Implement A* algorithm

CODE:

```
import heapq

class Node:
    def __init__(self, name, parent=None, g=0, h=0):
        self.name = name
        self.parent = parent
        self.g = g # cost from start to node
        self.h = h # heuristic estimated cost from node to goal
        self.f = g + h # total cost (f = g + h)

    def __lt__(self, other):
        # This ensures that heapq can compare nodes based on their f value
        return self.f < other.f

def a_star_algorithm(start, goal, graph, heuristic):
    open_list = []
    closed_list = set()

    # Create the start node
    start_node = Node(start, None, 0, heuristic[start])
    heapq.heappush(open_list, start_node)
    i=0
    while open_list:
        current_node = heapq.heappop(open_list)
        #print(current_node.name)
        # If we reach the goal, reconstruct the path
        if current_node.name == goal:
            path = []
            while current_node:
                path.append(current_node.name)
                current_node = current_node.parent
            return path[::-1] # Reverse the path to get the correct order

        closed_list.add(current_node.name)

        # Explore the neighbors
        for neighbor, cost in graph[current_node.name]:
```

```
            #print(neighbor,cost)
            if neighbor in closed_list:
                continue
            #print(current_node.g," ",cost)
            g_cost = current_node.g + cost
            #print(g_cost)
            h_cost = heuristic[neighbor]
            #print(h_cost)
            neighbor_node = Node(neighbor, current_node, g_cost, h_cost)

            # Check if the neighbor is already in the open list
            if not any(open_node.name == neighbor and open_node.f <=
neighbor_node.f for open_node in open_list):
                heapq.heappush(open_list, neighbor_node)

    return None # No path found

# Example graph with costs (adjacency list representation)
graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 2)],
    'C': [('A', 3), ('D', 1)],
    'D': [('B', 2), ('C', 1)],
}

# Heuristic values (straight-line distance to goal)
heuristic = {
    'A': 4,
    'B': 2,
    'C': 1,
    'D': 0,
}

start = 'A'
goal = 'D'

# Find the path from start to goal
path = a_star_algorithm(start, goal, graph, heuristic)

if path:
    print("Path found:", path)
else:
    print("No path found")
```

OUTPUT:

```
Path found: ['A', 'B', 'D']
```

Practical 10

AIM: Write a program to implement rule based system (automatic sprinkler RBS)

CODE:

```
class AutomaticSprinkler:
    def __init__(self):
        self.soil_moister = 0
        self.wheather_condition = "sunny"  # sunny, rainy, cloudy
        self.time_of_day = "day"  # day, night

    def set_soil_moister(self, moister_level):
        self.soil_moister = moister_level

    def set_wheather_condition(self, condition):
        self.wheather_condition = condition

    def set_time_of_day(self, time_of_day):
        self.time_of_day = time_of_day

    def decide_sprinkler_action(self):
        if self.wheather_condition == "rainy":
            print("No need for water, it's already rainy.")
            return "No action needed"

        if self.wheather_condition == "cloudy":
            if self.soil_moister < 50:  # Fixed condition
                print("It is cloudy, but soil is dry. Sprinkler activated.")
                return "Sprinkler ON"
            else:
                print("It is cloudy, soil moisture is sufficient. No need to water.")
                return "No action needed"

        if self.wheather_condition == "sunny":
            if self.soil_moister < 30 and self.time_of_day == "day":
                print("It is sunny, soil is dry. Sprinkler activated.")
                return "Sprinkler ON"
            elif self.soil_moister > 30 and self.time_of_day == "day":
                print("Soil moisture is sufficient. No need to water.")
                return "No action needed"
```

```
        if self.time_of_day == "night":
            print("It's night, no need to water.")
            return "No action needed"

        return "No action needed"

# Example usage:
sprinkler = AutomaticSprinkler()
sprinkler.set_soil_moister(25)
sprinkler.set_wheather_condition("sunny")
sprinkler.set_time_of_day("day")

result = sprinkler.decide_sprinkler_action()
print(f"Sprinkler action: {result}")
```

OUTPUT:

```
It is sunny, soil is dry. Sprinkler activated.
Sprinkler action: Sprinkler ON
```