

**CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,

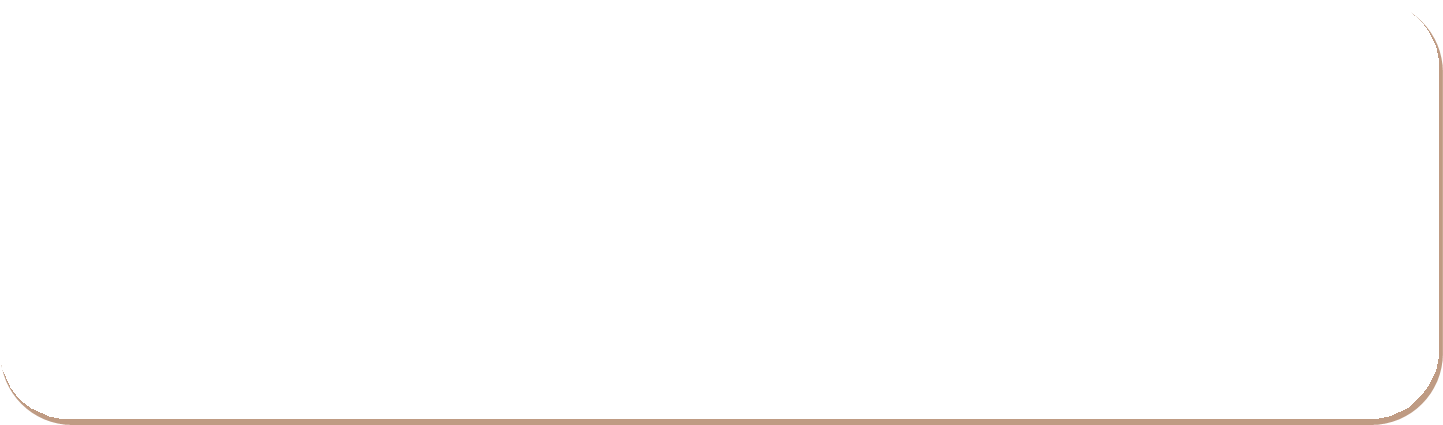
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

[www.cbit.ac.in](http://www.cbit.ac.in)

**Laboratory Manual**

**DEPARTMENT OF ARTIFICAL INTELLIGENCE AND MACHINE LEARNING (AI & ML)**



**PROGRAM - B.E**

**PRINCIPLES OF ARTIFICIAL INTELLIGENCE LAB(22AMC04)**

**(Common for CSE-AIML & AIML)**

**SEMESTER - IV**

**R-22 Regulation**

***Prepared by: S.Pallavi (Asst.Professor)* *Verified by:* ……………………………**



**CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,

Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in  
**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING (AI&ML)**

**Name of the Lab Course:**

PRINCIPLES OF ARTIFICIAL INTELLIGENCE LAB (22AMC04)

**INDEX**

|  |  |
| --- | --- |
| **Item** | **Page No** |
| Institute Vision, Institute Mission, Quality Policy, Department Vision and Department Mission | 2 |
| Program Educational Objectives (PEOs), Program Outcomes (POs) and Program Specific Outcomes (PSOs) | 3 |
| Syllabus | 7 |
| Course Introduction | 8 |
| Assessment Procedure of CIE and SEE | 9-10 |
| General Instructions for Laboratory Classes | 11 |
| Concluding Remarks / Gap Analysis | 16 |

**LABORATORY / PRACTICAL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp No** | **Laboratory / Practical** | **CO** | **BTL** | **Page No** |
| 1 | Design/construct the workflow of a general AI project using draw.io | 1 | L3 | 6 |
| 2 | Implement Water Jug Problem using A\* search | 1 | L1 | 8 |
| 3 | Implement an 8-puzzle solver using Heuristic search technique. | 2 | L1 | 12 |
| 4 | Implement the Constraint Satisfaction problem using backtracking | 3 | L1 | 15 |
| 5 | Implement a program for game search. | 3 | L1 | 17 |
| 6 | Implement a Bayesian network from a given data and infer the data from that Bayesian network. | 4 | L1 | 19 |
| 7 | Implement a MDP to run value and policy iteration in any environment. | 5 | L1 | 21 |
| 8 | Understanding of GitHub and conda environments. |  | L1 | 23 |
| 9 | Use the GitHub packages and libraries to frame a standard project and commit back to GitHub. |  | L1 | 24 |

**Note:**

1. Minimum 12 experiments should be included in every course other than experiments beyond syllabus.
2. Add project/ case studies/ student defined experiments for the courses having experiments less than 12 in syllabus.
3. Include Minimum 2 experiments beyond prescribed experiments meeting industry problems to challenge student learning.



**CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,

Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in

|  |
| --- |
| ***Vision of Institute***  To be the Centre of Excellence in Technical Education and Research. |
| ***Mission of Institute***  To address the Emerging needs through Quality Technical Education and Advanced Research. |
| ***Quality Policy***  CBIT imparts value based Technical Education and Training to meet the requirements of students, Industry, Trade/ Profession, Research and Development Organizations for Self-sustained growth of Society. |



**CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,

Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

[www.cbit.ac.in](http://www.cbit.ac.in)

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING (AI&ML)**

***Vision of the Department***

To produce professionals in artificial intelligence and machine learning through the best possible education, acquire international recognition as a destination, and advance society in exciting and creative ways.

***Mission of the Department***

* Impart rigorous training to generate knowledge through the state-of-the-art concepts and technologies in Artificial Intelligence and Machine Learning.
* Develop technical proficiency in students through creativity and leadership.
* Encourage lifelong learning, social responsibility, environmental conservation, and professional ethics.
* Establish centres of excellence in leading areas of computer and artificial intelligence disciplines.

***Program Educational Objectives (PEOs)***

PEO1: Using a solid foundation in mathematical, scientific, engineering, and current computing

Principles, formulate, analyse, and resolve engineering issues.

PEO2: Apply artificial intelligence theory and concepts to analyse the requirements, realise technical

Specifications, and design engineering solutions.

PEO3: Through cross-disciplinary projects and a variety of professional activities, demonstrate

technical proficiency, AI competency, and foster collaborative learning and a sense of teamwork.

PEO4: Provide graduates with solid knowledge, competence, and soft skills that will enable them to

ethically contribute to societal demands and achieve sustainable advancement in emerging

computer technologies through lifelong learning.



**CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,

Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

**AND MACHINE LEARNING (AI&ML)**

Program Outcomes (POs)

**Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization for the solution of complex engineering problems

**Problem analysis**: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**Design/development of solutions:**Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

**Conduct investigations of complex problems:**Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities, with an understanding of the limitations.

**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**Ethics:**Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**Individual and team work:**Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**Communication:**Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**Project management and finance:**Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**Life-long learning:**Recognise the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1. Apply the principal concepts of AI Engineering to design, develop, deploy and prototype AI Subsystems.

PSO2. Apply the knowledge gained pertaining to data storage, data analytics and AI concepts to solve real world business problems.

PSO3. Apply, analyse, design, develop, and test principles of AI concepts on Intelligent Systems.

**CBIT (A)** With effect from the academic year 2023-24

**22AMC04**

**PRINCIPLES OF ARTIFICIAL INTELLIGENCE LAB**

**Course Objectives:**

**The objectives of this course are,**

1. To design and analyse various computing algorithms and techniques using Python.

2. To apply different learning algorithms to solve real time problems.

3. To recognize the underlying mathematical models and logics behind various AI techniques**.**

**Course Outcomes:**

**On successful completion of the course, students will be able to,**

1. Understand the basic components of library environment and installations.

2. Analyse the design heuristics and apply various techniques to solve real world problems.

3. Apply variety of algorithms to solve problems.

4. Identify how to use GitHub and submit back genuine contributions. 5. Implement problems using game search algorithms.

**Lab Experiments:**

1. Design/construct the workflow of a general AI project using draw.io

2. Implement Water Jug Problem using A\* search

3. Implement an 8-puzzle solver using Heuristic search technique.

4. Implement the Constraint Satisfaction problem using backtracking.

5. Implement a program for game search.

6. Implement a Bayesian network from a given data and infer the data from that Bayesian network.

7. Implement a MDP to run value and policy iteration in any environment.

8. Understanding of GitHub and conda environments.

9. Use the GitHub packages and libraries to frame a standard project and commit back to GitHub.

**Text Books**:

1. Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, 3rd Edition, Prentice Hall, 2010.

2. Elaine Rich and Kevin Knight, “Artificial Intelligence”, Tata McGraw Hill, 3rd Edition, 2017.

**Suggested Reading:**

1. Trivedi, M.C., “A Classical Approach to Artificial Intelligence”, Khanna Publishing House, Delhi, 2018. 2. Saroj Kaushik, “Artificial Intelligence”, Cengage Learning India, 2011. Online Resources: 1. <https://nptel.ac.in/courses/106105077>

2. <https://nptel.ac.in/courses/106106126>

3. <https://aima.cs.berkeley.edu>

4. <https://ai.berkeley,edu/project_overview.html>

**Experiment-1**

**1.** **Design/construct the workflow of a general AI project using draw.io.**

The AI workflow typically involves several stages, from data collection and preprocessing to model training, evaluation, and deployment. Here's a simplified overview of the AI workflow with an example:

1. **Define the Problem:**
   * **Example:** Let's say we want to build a spam detection system for emails.
2. **Collect and Prepare Data:**
   * **Example:** Gather a dataset of labeled emails, where each email is labeled as either spam or not spam. This dataset will be used to train and evaluate the model.
3. **Data Preprocessing:**
   * **Example:** Clean and preprocess the text data by removing stop words, converting text to lowercase, and using techniques like tokenization.
4. **Feature Engineering:**
   * **Example:** Convert the text data into numerical features using methods like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings.
5. **Select a Model:**
   * **Example:** Choose a machine learning algorithm suitable for text classification, such as a Naive Bayes classifier or a neural network.
6. **Train the Model:**
   * **Example:** Split the dataset into training and testing sets. Train the selected model on the training set using an optimization algorithm, adjusting the model's parameters to minimize the prediction error.
7. **Evaluate the Model:**
   * **Example:** Use the testing set to evaluate the model's performance. Common evaluation metrics for classification tasks include accuracy, precision, recall, and F1 score.
8. **Fine-Tune the Model:**
   * **Example:** If the model's performance is not satisfactory, adjust hyperparameters or try different algorithms to improve results.
9. **Deploy the Model:**
   * **Example:** Once satisfied with the model's performance, deploy it to a production environment where it can be used to classify new, unseen emails as spam or not spam.
10. **Monitor and Update:**
    * **Example:** Regularly monitor the model's performance in the production environment. If necessary, update the model with new data or retrain it to adapt to changes in the data distribution.

**Experiment:2**

**Aim** : Implement Water Jug Problem using A\* search

**Explanation:** The water jug problem, also known as the "Die Hard 3" problem, is a classic puzzle that involves using two jugs to measure a specific volume of water. The puzzle is typically framed as follows:

You have two empty jugs, one with a capacity of *X* Liters and the other with a capacity of *Y* liters, where *X* and *Y* are positive integers. You are also given a water source, such as a tap or a river. The goal is to use these jugs to measure out a specific volume of water, typically denoted as *Z* Liters, where *Z* is a positive integer.

The puzzle typically comes with a set of rules:

1. You can fill a jug completely from the water source.
2. You can empty a jug completely.
3. You can pour water from one jug into another until the receiving jug is full or the source jug is empty.

The objective is to figure out a sequence of these actions to measure out exactly *Z* liters of water using the *X* and *Y* liter jugs.

Solving the water jug problem often requires creative thinking and problem-solving skills. There are different ways to approach the problem based on the values of *X*, *Y*, and *Z*. Some problems have unique solutions, while others may have multiple valid solutions.

Here are a few tips for solving the water jug problem:

1. **Understand the Constraints:** Start by understanding the capacities of the jugs (*X* and *Y*) and the target volume (*Z*). Ensure that *Z* is a possible result given the jug capacities.
2. **Experiment and Visualize:** Visualize the problem and experiment with different actions to move water between the jugs. Try to see if you can work backward from the desired *Z* litres.
3. **Use Math and Logic:** In some cases, you can use mathematical relationships to solve the problem more efficiently. For instance, you can calculate the greatest common divisor (GCD) of *X* and *Y* to determine if *Z* is a multiple of the GCD.
4. **Trial and Error:** If necessary, you may need to use trial and error to explore different sequences of actions until you find a solution.

The water jug problem is a classic example of a mathematical and logical puzzle, and it's often used to teach problem-solving skills and basic concepts in number theory. It's a fun challenge

to work on and can be adapted to various jug capacities and target volumes to make it more

or less difficult.

**Code:**

import heapq

# Define the state class to represent the current state of the jugs

class State:

def \_\_init\_\_(self, jug1, jug2, parent=None, action="Initial"):

self.jug1 = jug1

self.jug2 = jug2

self.parent = parent

self.action = action

self.cost = 0

def \_\_lt\_\_(self, other):

return self.cost < other.cost

def \_\_eq\_\_(self, other):

return self.jug1 == other.jug1 and self.jug2 == other.jug2

def \_\_hash\_\_(self):

return hash((self.jug1, self.jug2))

# A\* algorithm to solve the water jug problem

def solve\_water\_jug\_problem(capacity1, capacity2, target):

initial\_state = State(0, 0)

goal\_state = State(target, 0)

open\_list = []

closed\_set = set()

heapq.heappush(open\_list, initial\_state)

while open\_list:

current\_state = heapq.heappop(open\_list)

if current\_state == goal\_state:

path = []

while current\_state.parent:

path.append(current\_state.action)

current\_state = current\_state.parent

path.reverse()

return path

if current\_state in closed\_set:

continue

closed\_set.add(current\_state)

# Fill jug1

if current\_state.jug1 < capacity1:

fill\_jug1 = min(capacity1 - current\_state.jug1, capacity2 - current\_state.jug2)

new\_state = State(current\_state.jug1 + fill\_jug1, current\_state.jug2, current\_state, f"Fill jug1")

new\_state.cost = new\_state.parent.cost + 1

heapq.heappush(open\_list, new\_state)

# Fill jug2

if current\_state.jug2 < capacity2:

fill\_jug2 = min(capacity2 - current\_state.jug2, capacity1 - current\_state.jug1)

new\_state = State(current\_state.jug1, current\_state.jug2 + fill\_jug2, current\_state, f"Fill jug2")

new\_state.cost = new\_state.parent.cost + 1

heapq.heappush(open\_list, new\_state)

# Empty jug1

if current\_state.jug1 > 0:

new\_state = State(0, current\_state.jug2, current\_state, f"Empty jug1")

new\_state.cost = new\_state.parent.cost + 1

heapq.heappush(open\_list, new\_state)

# Empty jug2

if current\_state.jug2 > 0:

new\_state = State(current\_state.jug1, 0, current\_state, f"Empty jug2")

new\_state.cost = new\_state.parent.cost + 1

heapq.heappush(open\_list, new\_state)

# Pour water from jug1 to jug2

if current\_state.jug1 > 0 and current\_state.jug2 < capacity2:

pour\_jug1\_to\_jug2 = min(current\_state.jug1, capacity2 - current\_state.jug2)

new\_state = State(current\_state.jug1 - pour\_jug1\_to\_jug2, current\_state.jug2 + pour\_jug1\_to\_jug2, current\_state, f"Pour jug1 to jug2")

new\_state.cost = new\_state.parent.cost + 1

heapq.heappush(open\_list, new\_state)

# Pour water from jug2 to jug1

if current\_state.jug2 > 0 and current\_state.jug1 < capacity1:

pour\_jug2\_to\_jug1 = min(current\_state.jug2, capacity1 - current\_state.jug1)

new\_state = State(current\_state.jug1 + pour\_jug2\_to\_jug1, current\_state.jug2 - pour\_jug2\_to\_jug1, current\_state, f"Pour jug2 to jug1")

new\_state.cost = new\_state.parent.cost + 1

heapq.heappush(open\_list, new\_state)

return None

# Main function to solve the water jug problem

def main():

capacity1 = 4 # Capacity of jug 1

capacity2 = 3 # Capacity of jug 2

target = 2 # Target amount of water to measure

solution = solve\_water\_jug\_problem(capacity1, capacity2, target)

if solution:

print("Solution:")

for step, action in enumerate(solution):

print(f"Step {step + 1}: {action}")

else:

print("No solution found.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Experiment:3**

**Aim:** Implement an 8-puzzle solver using Heuristic search technique

**Explanation:** The 8-puzzle problem, also known as the "Sliding Puzzle" or "15-Puzzle," is a popular puzzle or sliding block puzzle that consists of a 3x3 grid (containing 9 tiles) and one blank tile. The goal of the puzzle is to arrange the tiles in ascending numerical order (from 1 to 8) in rows from left to right, with the blank tile in the bottom-right corner. The tiles can be moved horizontally or vertically, but only one tile at a time.

Here's a more detailed explanation of the 8-puzzle problem:

**Setup:**

You start with a 3x3 grid that contains eight numbered tiles (usually 1 through 8) and one empty space.

The initial configuration is random, and the objective is to transform it into the desired target configuration.

**Goal State**:

The goal is to reach a specific, predefined configuration where the tiles are ordered as follows:

1 2 3

4 5 6

7 8

**Legal Moves**:

You can move a tile into the adjacent empty space, either vertically or horizontally.

Only one tile can be moved at a time, and you cannot jump over tiles.

Some configurations may not be solvable. In such cases, reaching the goal state is impossible from that configuration.

Complexity:

The 8-puzzle problem is a classic example in the field of artificial intelligence and combinatorial optimization.

It is part of a larger family of sliding puzzle problems, some of which are more complex, such as the 15-puzzle.

**Solving the Puzzle**:

There are various algorithms and techniques for solving the 8-puzzle problem. These include:

Breadth-First Search (BFS): A simple and exhaustive approach that explores all possible moves and their consequences. It can be quite slow for large state spaces.

**A Search\***: A more efficient search algorithm that uses heuristics to prioritize promising moves.

Informed Search: Different heuristics can be used to estimate the cost of reaching the goal state from a given state, e.g., Manhattan distance or number of misplaced tiles.

**Inversions and Solvability:**

Whether a given configuration of the 8-puzzle is solvable depends on the number of inversions in the initial state. An inversion occurs when a larger number appears before a smaller number.

If the initial state has an even number of inversions, it is solvable. If it has an odd number of inversions, it is not solvable.

The 8-puzzle problem serves as an illustrative example in computer science and artificial intelligence, particularly in the context of search algorithms, pathfinding, and problem-solving. It can be used to demonstrate the efficiency of different search algorithms and heuristics in finding a solution from a given initial state to the goal state.

**Code:**

import heapq

import copy

# Define the 8-puzzle state class

class PuzzleState:

def \_\_init\_\_(self, board, parent=None, move="Initial"):

self.board = board

self.parent = parent

self.move = move

self.g = 0

self.h = 0

def \_\_lt\_\_(self, other):

return (self.g + self.h) < (other.g + other.h)

def \_\_eq\_\_(self, other):

return self.board == other.board

def \_\_hash\_\_(self):

return hash(tuple(map(tuple, self.board)))

# Define the goal state

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Define possible moves

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

move\_names = ["Right", "Down", "Left", "Up"]

# Define a function to calculate the Manhattan distance heuristic

def calculate\_manhattan\_distance(state):

distance = 0

for i in range(3):

for j in range(3):

if state.board[i][j] != 0:

x, y = divmod(state.board[i][j] - 1, 3)

distance += abs(x - i) + abs(y - j)

return distance

# A\* search algorithm to solve the 8-puzzle

def solve\_8\_puzzle(initial\_state):

open\_list = []

closed\_set = set()

heapq.heappush(open\_list, initial\_state)

while open\_list:

current\_state = heapq.heappop(open\_list)

if current\_state.board == goal\_state:

# Solution found

path = []

while current\_state.parent:

path.append(current\_state.move)

current\_state = current\_state.parent

path.reverse()

return path

if current\_state in closed\_set:

continue

closed\_set.add(current\_state)

for move, move\_name in zip(moves, move\_names):

new\_i, new\_j = current\_state.board.index([0]) + move[0],

current\_state.board[current\_state.board.index([0])].index(0) + move[1]

if 0 <= new\_i < 3 and 0 <= new\_j < 3:

new\_board = copy.deepcopy(current\_state.board)

new\_board[new\_i][new\_j], new\_board[current\_state.board.index([0])][current\_state.board[current\_state.board.index([0])].index(0)] = 0, new\_board[new\_i][new\_j]

new\_state = PuzzleState(new\_board, current\_state, move\_name)

new\_state.g = current\_state.g + 1

new\_state.h = calculate\_manhattan\_distance(new\_state)

heapq.heappush(open\_list, new\_state)

return None

# Main function to solve the 8-puzzle

def main():

initial\_board = [[1, 2, 3], [4, 5, 0], [7, 8, 6]] # Change this to the initial configuration

initial\_state = PuzzleState(initial\_board)

solution = solve\_8\_puzzle(initial\_state)

if solution:

print("Solution:")

for step, action in enumerate(solution):

print(f"Step {step + 1}: {action}")

else:

print("No solution found.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Experiment:4**

**Aim:** Implement the Constraint Satisfaction problem using backtracking.

**Code:**

class ConstraintSatisfactionProblem:

def \_\_init\_\_(self, variables, domains, constraints):

self.variables = variables

self.domains = domains

self.constraints = constraints

def is\_consistent(self, variable, value, assignment):

return all(constraint(variable, value, assignment) for constraint in self.constraints.get(variable, []))

def backtrack(self, assignment):

if len(assignment) == len(self.variables):

return assignment # All variables are assigned, solution

found

var = self.select\_unassigned\_variable(assignment)

for value in self.order\_domain\_values(var, assignment):

if self.is\_consistent(var, value, assignment):

assignment[var] = value

result = self.backtrack(assignment)

if result is not None:

return result

assignment.pop(var) # Backtrack if no solution found

return None

def select\_unassigned\_variable(self, assignment):

return next(variable for variable in self.variables if variable not in assignment)

def order\_domain\_values(self, variable, assignment):

return self.domains[variable]

# Example usage:

variables = ['A', 'B', 'C']

domains = {'A': [1, 2, 3], 'B': [1, 2, 3], 'C': [1, 2, 3]}

constraints = {

'A': [('B', lambda a, b, assignment: a != b)],

'B': [('A', lambda b, a, assignment: b != a)],

'C': [('A', lambda c, a, assignment: c != a)]

}

csp = ConstraintSatisfactionProblem(variables, domains, constraints)

solution = csp.backtrack({})

if solution:

print("Solution found:")

print("\n".join(f"{variable}: {value}" for variable, value in solution.items()))

else:

print("No solution found.")

**Experiment:5**

**Aim:** Implement a program for game search.

**Explanation** : Game search algorithms, like Minimax, are often used in game-playing scenarios. Below is a simple implementation of the Minimax algorithm for a two-player game, specifically Tic-Tac-Toe, in Python.

**Code**:

class TicTacToe:

def \_\_init\_\_(self):

self.board = [' '] \* 9

self.current\_player = 'X'

def print\_board(self):

for i in range(0, 9, 3):

print("|".join(self.board[i:i + 3]))

if i < 6:

print("-----")

def is\_winner(self, player):

# Check rows, columns, and diagonals

for i in range(0, 9, 3):

if all(self.board[j] == player for j in range(i, i + 3)):

return True

if all(self.board[j] == player for j in range(i % 3, 9, 3)):

return True

if all(self.board[i] == player for i in [0, 4, 8]) or all(self.board[i] == player for i in [2, 4, 6]):

return True

return False

def is\_full(self):

return ' ' not in self.board

def is\_game\_over(self):

return self.is\_winner('X') or self.is\_winner('O') or self.is\_full()

def get\_available\_moves(self):

return [i for i, v in enumerate(self.board) if v == ' ']

def make\_move(self, move):

self.board[move] = self.current\_player

self.current\_player = 'O' if self.current\_player == 'X' else 'X'

def undo\_move(self, move):

self.board[move] = ' '

self.current\_player = 'O' if self.current\_player == 'X' else 'X'

def minimax(board, maximizing\_player):

if board.is\_game\_over():

if board.is\_winner('X'):

return -1

elif board.is\_winner('O'):

return 1

else:

return 0

if maximizing\_player:

max\_eval = float('-inf')

for move in board.get\_available\_moves():

board.make\_move(move)

eval = minimax(board, False)

board.undo\_move(move)

max\_eval = max(max\_eval, eval)

return max\_eval

else:

min\_eval = float('inf')

for move in board.get\_available\_moves():

board.make\_move(move)

eval = minimax(board, True)

board.undo\_move(move)

min\_eval = min(min\_eval, eval)

return min\_eval

def get\_best\_move(board):

best\_move = None

best\_eval = float('-inf')

for move in board.get\_available\_moves():

board.make\_move(move)

eval = minimax(board, False)

board.undo\_move(move)

if eval > best\_eval:

best\_eval = eval

best\_move = move

return best\_move

# Example Usage:

game = TicTacToe()

while not game.is\_game\_over():

game.print\_board()

if game.current\_player == 'X':

move = int(input("Enter your move (0-8): "))

else:

move = get\_best\_move(game)

game.make\_move(move)

game.print\_board()

if game.is\_winner('X'):

print("You win!")

elif game.is\_winner('O'):

print("You lose!")

else:

print("It's a draw!")

**Experiment : 6**

**Aim:** Implement a Bayesian network from a given data and infer the data from that Bayesian network

**Code:**

from pgmpy.models import BayesianModel

from pgmpy.estimators import ParameterEstimator

from pgmpy.inference import VariableElimination

import pandas as pd

# Sample data

data = pd.DataFrame(data={'Rain': ['No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No'],

'TrafficJam': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No'],

'ArriveLate': ['Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No']})

# Define the Bayesian network structure

model = BayesianModel([('Rain', 'TrafficJam'), ('TrafficJam', 'ArriveLate')])

# Fit the model to the data using Maximum Likelihood Estimation

model.fit(data, estimator=ParameterEstimator)

# Print conditional probability distributions

print(model.get\_cpds())

# Perform inference

inference = VariableElimination(model)

query\_result = inference.query(variables=['ArriveLate'], evidence={'Rain': 'Yes'})

print(query\_result)

**Explanation:** This example creates a Bayesian network with three variables: Rain, TrafficJam, and ArriveLate. The relationships between these variables are defined in the structure of the network. The code then fits the model to the given data and prints the conditional probability distributions. Finally, it performs inference to query the probability of ArriveLate given evidence of Rain being 'Yes'.

Please note that this is a simplified example, and Bayesian networks can become much more complex in real-world scenarios. The pgmpy library provides tools for working with Bayesian networks, including parameter estimation, structure learning, and inference. Make sure to adapt the code and network structure to fit your specific data and use case.

**Experiment: 7**

**Aim:** Implement a MDP to run value and policy iteration in any environment.

Code:

import numpy as np

class GridWorldMDP:

def \_\_init\_\_(self, size, goal, trap):

self.size = size

self.goal = goal

self.trap = trap

self.state\_space = [(i, j) for i in range(size) for j in range(size)]

self.action\_space = ['UP', 'DOWN', 'LEFT', 'RIGHT']

self.transitions = self.build\_transitions()

self.rewards = self.build\_rewards()

def build\_transitions(self):

transitions = {}

for state in self.state\_space:

transitions[state] = {}

for action in self.action\_space:

transitions[state][action] = self.calculate\_transitions(state, action)

return transitions

def calculate\_transitions(self, state, action):

i, j = state

if action == 'UP':

return self.validate\_state(i - 1, j)

elif action == 'DOWN':

return self.validate\_state(i + 1, j)

elif action == 'LEFT':

return self.validate\_state(i, j - 1)

elif action == 'RIGHT':

return self.validate\_state(i, j + 1)

def validate\_state(self, i, j):

i = max(0, min(i, self.size - 1))

j = max(0, min(j, self.size - 1))

if (i, j) == self.goal:

return [(0.0, self.goal)]

elif (i, j) == self.trap:

return [(0.0, self.trap)]

else:

return [(1.0, (i, j))]

def build\_rewards(self):

rewards = {}

for state in self.state\_space:

rewards[state] = -1.0

rewards[self.goal] = 0.0

rewards[self.trap] = -10.0

return rewards

def value\_iteration(mdp, gamma=0.9, epsilon=0.01):

state\_values = {state: 0.0 for state in mdp.state\_space}

while True:

delta = 0

for state in mdp.state\_space:

if state == mdp.goal or state == mdp.trap:

continue

v = state\_values[state]

state\_values[state] = max([sum([p \* (mdp.rewards[next\_state] + gamma \* state\_values[next\_state])

for p, next\_state in mdp.transitions[state][action]])

for action in mdp.action\_space])

delta = max(delta, abs(v - state\_values[state]))

if delta < epsilon:

break

return state\_values

def policy\_iteration(mdp, gamma=0.9):

policy = {state: np.random.choice(mdp.action\_space) for state in mdp.state\_space}

state\_values = {state: 0.0 for state in mdp.state\_space}

while True:

# Policy Evaluation

while True:

delta = 0

for state in mdp.state\_space:

if state == mdp.goal or state == mdp.trap:

continue

v = state\_values[state]

action = policy[state]

state\_values[state] = sum([p \* (mdp.rewards[next\_state] + gamma \* state\_values[next\_state])

for p, next\_state in mdp.transitions[state][action]])

delta = max(delta, abs(v - state\_values[state]))

if delta < 0.01:

break

# Policy Improvement

policy\_stable = True

for state in mdp.state\_space:

if state == mdp.goal or state == mdp.trap:

continue

old\_action = policy[state]

policy[state] = max(mdp.action\_space, key=lambda a: sum([p \* (mdp.rewards[next\_state] + gamma \* state\_values[next\_state])

for p, next\_state in mdp.transitions[state][a]]))

if old\_action != policy[state]:

policy\_stable = False

if policy\_stable:

break

return policy, state\_values

# Example Usage:

size = 3

goal = (2, 2)

trap = (1, 1)

mdp = GridWorldMDP(size, goal, trap)

# Value Iteration

value\_iteration\_result = value\_iteration(mdp)

print("Value Iteration Results:")

for state, value in value\_iteration\_result.items():

print(f"State: {state}, Value: {value}")

# Policy Iteration

policy\_iteration\_result, policy\_iteration\_state\_values = policy\_iteration(mdp)

print("\nPolicy Iteration Results:")

for state, action in policy\_iteration\_result.items():

print(f"State: {state}, Action: {action}, Value: {policy\_iteration\_state\_values[state]}")

# This code defines a simple grid world MDP with a goal and a trap state. It then runs both value iteration and policy iteration algorithms to find the optimal state values and policies. The results are printed for each state. Feel free to modify the MDP parameters or the algorithms to fit your specific needs.

**ASSESSMENT PROCEDURE AND AWARD OF CIE MARKS**

Following is the subdivision for the internal marks (50) of the Lab:

1. 20 marks for the lab internal tests

Two tests are to be conducted i.e one test after 1st cycle of experiments and second test after the second cycle. Average of two tests marks put together should be consider (20 maximum)

1. 30 marks for CIE

For the CIE 30 marks will be awarded based on the rubrics provided below.

This Rubrics are general guideline. Based on the lab type (programming or hardware) and complexity of the course Rubrics can be customized by the department in tune to program and course offered. Performance Indicators of the Rubrics also can be changed by the departments/Program based on the need.

**RUBRICS**

| **S. No** | **Parameter** | **Descriptors and Score** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Outstanding** | **Good** | **Fair** | **Poor** | **Very Poor** |
| 1 | Pre-Experiment Preparation work  5 M | Well prepared for the experimentation with a clear specifications, plan/design and additional information (5M) | prepared for the experimentation with clear specifications and plan/design (4M) | Adequately prepared for the experimentation without clear specifications and plan/design (3M) | Minimal preparation and without clear specifications and plan/design (2M) | Lacks preparation and without clear specifications and plan/design (1 M) |
| 2 | Experimentation (Problem Solving, Methodology of Conduction)  10 M | Student conducts experiment with all possible test cases in an optimized fashion.(10M) | Student solves the problem and conducts experiments with all possible test cases (8M) | Student solves the problem and conducts experiment with few possible test cases with complexity (6M) | Student solves the problem with few test cases with complexity (4M) | Student fails to solve the problem (2M) |
| 3 | Post Experiment Analysis [Viva, Inference]  5M | Demonstrates the simulation/ findings /Hardware results Infers and answer all the Questions posed by Instructor  (5M) | Demonstrates results and inference;  Able to answer Few Questions posed by Instructor (4M) | Demonstrates results and inference;  Unable to answer the Questions posed by Instructor. (3M) | Demonstrates Partial results and inference;  Unable to answer the Questions posed by Instructor (2M) | Failed to Demonstrates results and inference;  Unable to answer the Questions posed by Instructor (1M) |
| 4 | Report Writing  5M | Report meets all requirements and it is prepared in original and creative way to engage readers (5M) | Report with well-organized content, visuals, graphics, citations and references (4M) | Report is complete and adequate with poor grammar. (3M) | Report is complete, poor grammar and inadequate and failed to organize thoughts (2M) | Report is incomplete, unclear, poor grammar and inadequate (1M) |
| 5 | Conduct (Ethics, Safety, Team Work) 5M | Excellent team spirit, strictly follows ethics and safety precautions with good team work (5M) | Follows the safety precautions, practices ethics and poor team work (4M) | Follows safety precautions and ethical practices and failed to exhibit teamwork.  (3M) | Followed minimum safety precautions and ethical practices and failed to exhibit team work. (2M) | Does not Follows safety precautions and ethical practices and failed to exhibit team work. (1M) |
| TOTAL SCORE | |  | | | | |

**ASSESSMENT PROCEDURE AND AWARD OF SEE MARKS**

| **S. No.** | **Parameter** | **Descriptors and Score** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Outstanding** | **Good** | **Fair** | **Poor** | **Very Poor** |
| 1 | Record  5 M | The content of all the experiments is well-organized, recorded the tables and neatly drawn graphs. Results and discussions are well presented. (5M) | The content of most of the experiments are well-organized, recorded the tables, neatly drawn graphs. Results and discussion are presented. (4M) | The content of the most of experiments are well-organized, recorded the tables, neatly drawn graphs. Results and discussion are not presented (3M) | The content of the few experiments is not well-organized, recorded the tables, neatly drawn graphs. Results and discussion are not presented (2M) | The content of the all the experiments is not well-organized, recorded the tables, neatly drawn graphs. Results and discussion are not presented (1M) |
| 2 | Write up about the experiment  10M | Presentation of the given experiment/program is very well organized with the required content, specifications, plan/procedure of the conduct and all the required additional information. (10M) | Presentation of the given experiment/program is organized with the required content, specifications, plan/procedure of the conduct.  (8M) | Presentation of the given experiment/program is organized and is without clear specifications and plan/design.  (6M) | Presentation of the given experiment/program is minimal with clear specifications and plan/design.  (4M) | Presentation of the given experiment/program is minimum without clear specifications and plan/design.  (2M) |
| 3 | Conduction of the experiment and observations  15M | Conducts experiment / Simulate the problem with proper connections / all possible test cases. All the possible observations are noted. (15M) | Conducts experiment / Simulate the problem with connections / with most of the test cases. Failed to note all observations (12M) | Conducts experiment / Simulate the problem with proper connections / less test cases.  (9M) | Conducts experiment / Simulate the problem with improper connections / less test cases. (6M) | Failed to Conducts experiment / Simulate the problem with improper connections / less test cases. (3M) |
| 4 | Results & Analysis  10M | Demonstrates the experimental results with adequate analysis / simulation/ findings /obtained results /plotting the graphs. (10M) | Demonstrates the experimental results with required analysis / simulation/ findings /obtained results /plotting the graphs. (8M) | Demonstrates the experimental results, failed to maximum analysis / simulation/ findings.  (6M) | Demonstrates the partial experimental results with least analysis / simulation/ findings.  (4M) | Failed to Demonstrate the results and least analysis / simulation/ findings.  (2M) |
| 5 | Viva-Voce  10M | Answers most of the questions with good analytical explanation. (10M) | Answers most of the questions with good explanation. (8M) | Answers only few of the questions with good explanation. (6M) | Answers only few of the questions with nominal explanation. (4M) | Failed to answer questions. (2M) |
| TOTAL SCORE | | | |  |  |  |

**GENERAL INSTRUCTIONS FOR LABORATORY CLASSES**

**DO‘S**

1. Without Prior permission do not enter into the Laboratory.
2. While entering into the LAB students should wear their ID cards.
3. The Students should come with proper uniform.
4. Students should sign in the LOGIN REGISTER before entering into the laboratory.
5. Students should come with observation and record note book to the laboratory.
6. Students should maintain silence inside the laboratory.
7. After completing the laboratory exercise, make sure to shutdown the system properly

**DONT‘S**

1. Students bringing the bags inside the laboratory..
2. Students wearing slippers/shoes insides the laboratory.
3. Students using the computers in an improper way.
4. Students scribbling on the desk and mishandling the chairs.
5. Students using mobile phones inside the laboratory.
6. Students making noise inside the laboratory.

**SAMPLE VIVA VOCE QUESTIONS:**

Here are some viva voce questions:

1. How would you design and construct the workflow of a general AI project using draw.io?
2. Can you explain the A\* search algorithm and how you implemented it to solve the Water Jug Problem?
3. Describe the heuristic search technique you employed to solve the 8-puzzle. How does the choice of heuristic impact the efficiency of the solution?
4. Walk me through the implementation of the Constraint Satisfaction Problem using backtracking.
5. Explain the algorithm you used for game search. What considerations are important when implementing a search algorithm for a game?
6. How did you construct a Bayesian network from given data, and how does it facilitate data inference?
7. Discuss the implementation of a Markov Decision Process (MDP) and the application of value and policy iteration in a given environment.
8. Describe the process of building a game bot using easy AI libraries. What challenges did you encounter?
9. Explain the role of GitHub in version control and collaboration in software development. How does Anaconda contribute to creating a robust development environment for AI projects?
10. Walk me through the steps of using GitHub packages and libraries to frame a standard project. How do you ensure effective collaboration and version control? How would you commit changes back to GitHub?

**GAP ANALYSIS BY THE COURSE FACULTY / COORDINATOR**

**…………………………………………………………………..**

**…………………………………………………………………..**

**…………………………………………………………………..**

**…………………………………………………………………..**

**…………………………………………………………………..**

**…………………………………………………………………..**

**…………………………………………………………………..**

**…………………………………………………………………..**

Faculty name & Signature with date

HoD/BoS Chairman remarks:…………………………………………………………………………………………………………………………………………………………………………………………………………………………………….…

HoD/BoS chairman Signature with date