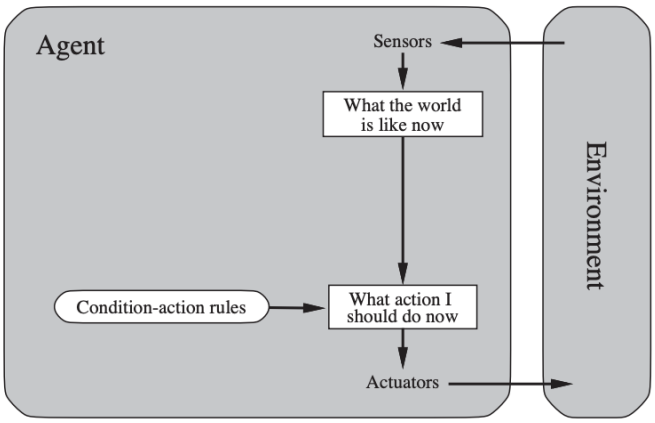
Unit – I

Q.No	Questions	CO's	Bloom's Level
	PART A		
1.	Define Artificial Intelligence. Artificial Intelligence (AI) is the study of how to make machines intelligent—capable of thinking, learning, and acting like humans or rational agents.	CO1	K1
2.	What is an intelligent agent? An intelligent agent is an autonomous AI-powered system that perceives its environment using sensors, processes information, and uses actuators to take actions that achieve specific goals or objectives.	CO1	K1
3.	What is the difference between informed and uninformed search? Uninformed search does not use any extra knowledge beyond the problem definition, while informed search uses heuristics to guide the search. Uninformed search is generally less efficient and may explore many unnecessary paths, whereas informed search reduces exploration by focusing on promising paths. Examples of uninformed search include BFS and DFS, while informed search includes A* and Greedy Best-First Search.	CO1	K2
4.	What is meant by the environment in AI? The environment is the external world in which an AI agent operates, perceives, and takes actions to achieve its goals. For example: ● In a chess game, the board and opponent are the environment.	CO1	K1
5.	What is a search algorithm? A search algorithm in AI is a step-by-step method used by an agent to find a sequence of actions that leads from the initial state to the goal state. Eg: BFS, DFS	CO1	K1

6.	State the drawback of depth-first search. DFS may go into an infinite path and never find a solution if the search space is very deep. It does not always find the shortest or optimal solution. Performance can be poor in large search spaces because it may waste time exploring unnecessary deep nodes.	CO1	K2
7.	What is the purpose of goal testing? Goal testing is used to check whether the current state matches the goal state. It tells the search algorithm if a solution has been reached or if the search should continue.	CO1	K1
8.	Define A* search. A* search is an informed search algorithm that uses both the cost to reach a node (g(n)) and the estimated cost from that node to the goal (h(n)) to find the most efficient path. It is both complete and optimal if the heuristic is admissible.	CO1	K1
9.	What is a heuristic? A *heuristic* is a function that estimates the cost of the cheapest path from a given state to the goal state. It provides additional domain-specific knowledge to guide informed search algorithms toward the goal more efficiently. Heuristics do not guarantee accuracy but help reduce the search effort.	CO1	K1
10.	Name two uninformed search strategies. ● Breadth-First Search (BFS) ● Depth-First Search (DFS) ● Uniform Cost Search (UCS) ● Depth-Limited Search (DLS) ● Iterative Deepening Depth-First Search (IDDFS) ● Bidirectional Search	CO1	K2
11.	Give an example of a real-world problem solvable by AI. Self-driving cars use AI to sense the environment (roads, traffic	CO1	K2

	signals, pedestrians) and make decisions like braking, turning, or changing lanes safely.		
12.	Define constraint satisfaction problem. A Constraint Satisfaction Problem (CSP) is a problem defined by a set of variables, each with a domain of possible values, and a set of constraints that specify allowable combinations of values. The goal is to assign values to all variables such that all constraints are satisfied.	CO1	K1
13.	Define problem formulation. Defining a problem by specifying the initial state, the set of possible actions, the transition model, the goal test, and the path cost. It provides the framework that a search algorithm uses to find a solution.	CO1	K1
14.	List the components of a rational agent. Performance Measure – the criteria for evaluating how successful the agent is. Percepts – the inputs the agent receives from the environment. Percept Sequence – the history of all percepts received so far. Agent Function – the mapping from percept sequences to actions. Agent Program – the implementation of the agent function that runs on the physical architecture.	CO1	K1
15.	What is state space in AI? State space in AI is the set of all possible states that can be reached in a problem, starting from the initial state and moving through actions until the goal state. It represents the entire search area of the problem.	CO1	K1

Q.No	Questions	CO's	Bloom'

			s Level
Part – B			
1.	Analyze the structure and various types of intelligent agents with suitable examples. How would you classify an agent operating in a self-driving car scenario? **Simple Reflex Agents :** ● Simple reflex agents select actions based solely on the current percept, ignoring the percept history. They operate using condition-action rules (if-then rules) that directly map perceived conditions to actions. ● <u>Example</u>: The classic vacuum-cleaner agent that cleans if the current square is dirty and moves otherwise is a simple reflex agent. Another example is a driver braking immediately when the brake lights of the car in front turn on. ● <u>Working Mechanism</u>: ● The agent perceives the current state via sensors. ● <u>The percept is</u> **function** TABLE-DRIVEN-AGENT(*percept*) **returns** an action **persistent**: *percepts*, a sequence, initially empty *table*, a table of actions, indexed by percept sequences, initial append *percept* to the end of *percepts* *action* ← LOOKUP(*percepts*, *table*) **return** *action* interpreted into an abstract state description. ● The agent matches this state against a set of condition-action rules. ● The first matching rule determines the action to execute. ● This process is simple and fast, often implementable by Boolean logic circuits.  	CO1	K4

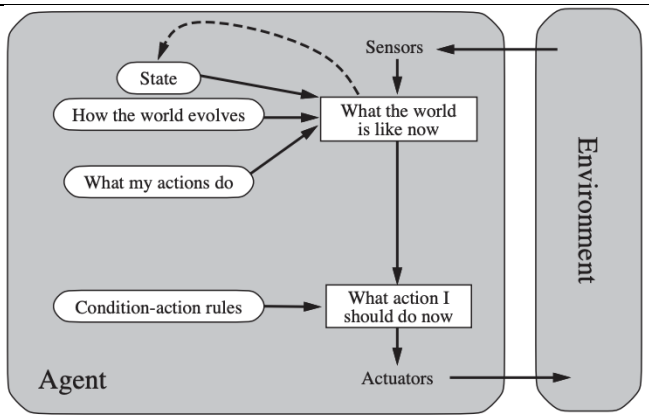
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

- Advantages:
  - Very simple and efficient to implement.
  - Works well in fully observable environments where the current percept contains all necessary information.
- Limitations:
  - Cannot handle partially observable environments because it ignores history or internal state.
  - May get stuck in infinite loops or make poor decisions if the percept is ambiguous or incomplete.
  - Lacks learning and adaptation capabilities.
- Example of Problem:
  A vacuum agent with only a dirt sensor but no location sensor might endlessly move left or right without cleaning effectively.

Model Based Reflex Agents

- Model-based reflex agents maintain an internal state that represents unobserved aspects of the environment, updated based on the percept history and a model of how the world evolves and how the agent's actions affect it.
- Internal State:
  Keeps track of information not directly observable in the current percept. For example, a driving agent remembers positions of other cars not currently visible or the agent's destination.
- World Model:
  Encodes knowledge about:
  - How the environment changes independently of the agent (e.g., a car behind getting closer).
  - How the agent's actions change the environment (e.g., turning the steering wheel turns the car).



function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
persistent: *state*, the agent's current conception of the world state
model, a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially none

state ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
rule ← RULE-MATCH(*state*, *rules*)
action ← *rule*.ACTION
return *action*

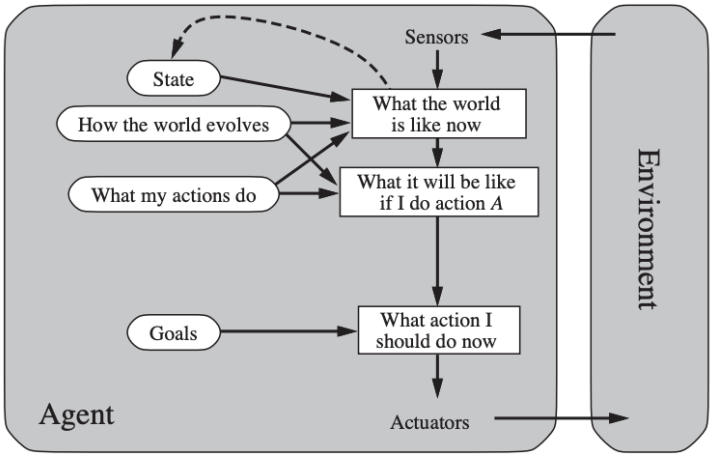
- Decision Process:
  - Update State: Combine previous internal state, recent action, and current percept to update the internal model.
  - Rule Matching: Use condition-action rules on the updated state to select the next action.
- Advantages:
  - Can operate in partially observable environments by maintaining and updating internal state.
  - More flexible and intelligent than simple reflex agents.
  - Can handle uncertainty by making best guesses about the current state.
- Limitations:
  - Building and maintaining accurate models can be computationally expensive.
  - Models may not capture all real-world complexities and require frequent updates.
  - Still limited by the predefined set of condition-action rules.
- Example:
  An automated taxi uses sensors to perceive the environment, updates its internal model of traffic and road conditions, and decides actions like braking or lane changes based on this model.

Goal-based agents

- Goal-based agents select actions based on both the current state and a goal that describes desirable situations (e.g.,

reaching a destination). They combine a model of the world with goal information to choose actions that achieve the goal.

- Key Features:
  - Use search and planning to find sequences of actions leading to the goal.
  - Consider future consequences of actions ("What will happen if I do this?").
  - More flexible than reflex agents because goals and models are explicit and modifiable.
  - Can adapt behavior easily by changing the goal without rewriting rules.

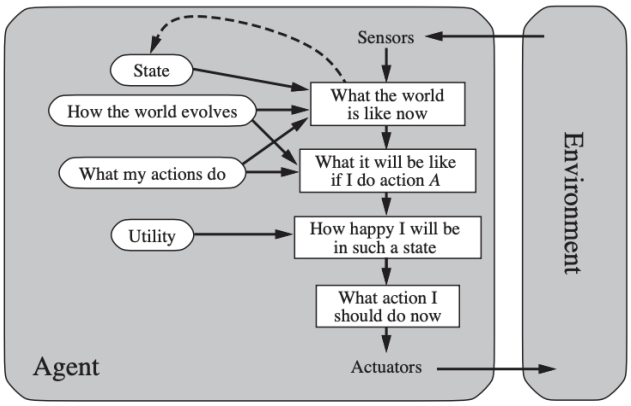


- Example:
  A taxi deciding whether to turn left, right, or go straight based on the destination goal rather than just reacting to immediate percepts.
- Applications:
  Robotics, autonomous vehicles, game AI, and any domain requiring planning and goal achievement.
- Advantages:
  - Clear direction for decision-making.
  - Adaptable to different goals and environments.
  - Can handle complex tasks requiring multi-step planning.
- Limitations:
  - Requires knowledge of goals and environment model.
  - Planning can be computationally expensive.

Utility-based agents

- Utility-based agents extend goal-based agents by using a utility function that assigns a numerical value to each possible state, representing the agent's preferences. They choose actions to maximize expected utility, balancing trade-offs among conflicting goals.
- Key Features:

- Handle conflicting goals (e.g., speed vs. safety) by weighing utilities.
- Make decisions under uncertainty and partial observability by maximizing expected utility.
- Internalize the performance measure as a utility function guiding rational behavior.

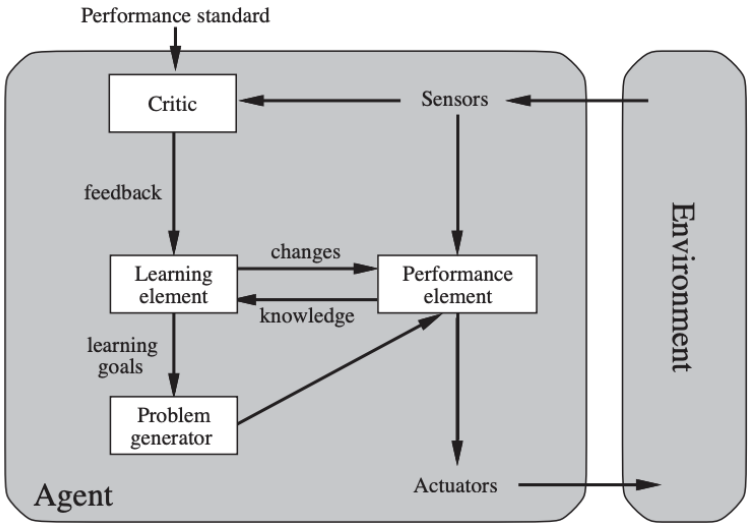


- Expected Utility:
  Agents compute the expected utility of possible outcomes, averaging over uncertainties weighted by their probabilities, and select the action with the highest expected utility.
- Example:
  A taxi choosing a route that balances shortest time, safety, and fuel efficiency rather than simply reaching the destination.
- Advantages:
  - More nuanced decision-making than binary goal satisfaction.
  - Can adapt to stochastic and uncertain environments.
  - Supports principled trade-offs among multiple objectives.
- Limitations:
  - Requires defining a suitable utility function.
  - Computationally complex to calculate expected utilities and optimal actions.

Learning agents

- Programming intelligent agents by hand is labor-intensive and limited. Turing (1950) proposed building machines that can learn and improve themselves, which is now the preferred approach in AI.
- Definition:
  A learning agent improves its performance over time by learning from experience and feedback, allowing it to operate effectively in initially unknown or changing environments.
- Four Key Components:

- Performance Element:
  - Responsible for selecting and executing actions based on current knowledge.
  - Essentially the agent itself as previously described (perceiving and acting).
- Learning Element:
  - Responsible for making improvements to the performance element by learning from feedback.
  - Modifies the agent's knowledge and behavior to perform better.
- Critic:
  - Evaluates the agent's actions against a fixed performance standard (objective measure of success).
  - Provides feedback to the learning element on how well the agent is doing.
- Problem Generator:
  - Suggests exploratory or novel actions to gather new information.
  - Encourages the agent to try suboptimal actions that may lead to better long-term performance.

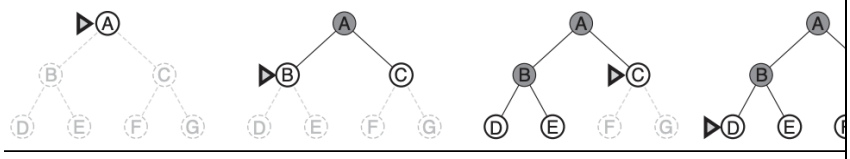


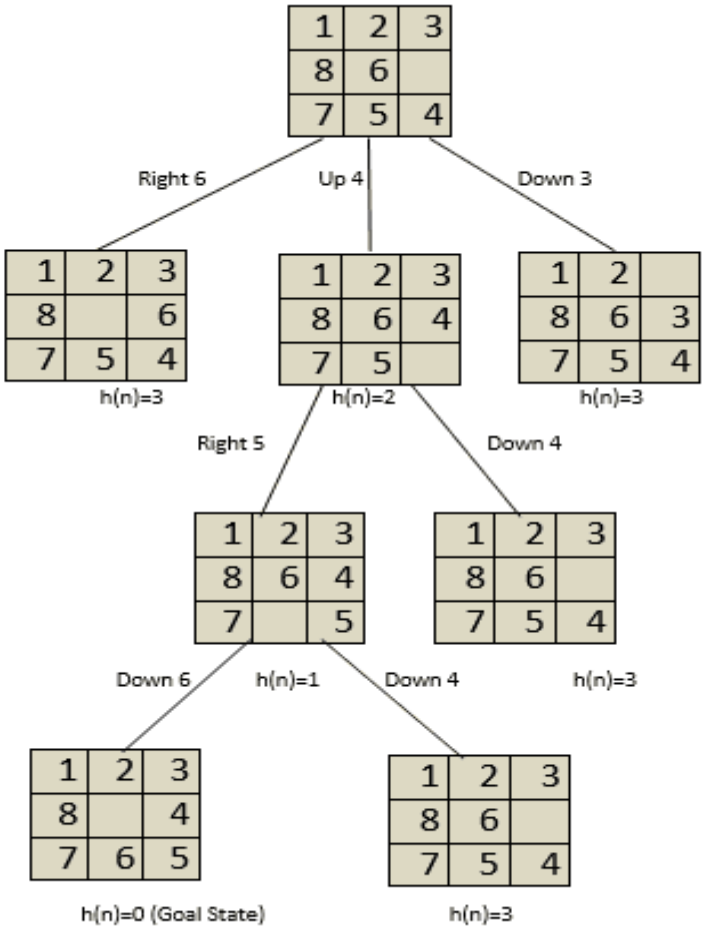
- Learning Process:
  - The agent interacts with the environment via the performance element.
  - The critic observes outcomes and provides feedback.
  - The learning element updates the performance element based on feedback.
  - The problem generator encourages exploration to improve learning.
- Example:
  An automated taxi learns from experiences such as

	aggressive maneuvers causing negative feedback (e.g., angry drivers), leading it to modify its driving rules for safer behavior. ● Types of Learning: ● Learning how the world evolves (environment dynamics). ● Learning the effects of the agent's actions. ● Learning utility or reward information from feedback (e.g., passenger satisfaction).		
2.	Demonstrate how a problem-solving agent operates in a real-world scenario such as a route-finding GPS system. Explain the key steps involved. Real-world scenario: Route-Finding GPS Imagine you are in Chennai Central Railway Station and want to go to Marina Beach. The GPS system acts as a problem-solving agent to find the best route. Key Steps Involved 1. Problem Formulation The agent needs to define the problem in terms of: ● Initial state → Current location (Chennai Central) ● Goal state → Destination (Marina Beach) ● Possible actions → Move via different roads, highways, or shortcuts ● Path cost function → Time taken, distance covered, or toll fees Example: "Take Mount Road", "Take Kamarajar Salai", etc. 2. Search Space Representation The city's roads are represented as a graph: ● Nodes (vertices) → Locations or junctions ● Edges → Roads connecting the locations ● Weights → Distance, travel time, or traffic conditions	CO1	K4

	The map becomes the state space. 3. Goal Test The agent checks whether the current location = Marina Beach. If yes → Stop. If not → Continue searching. 4. Search Algorithm Application The GPS applies a search strategy to explore possible routes: ● Uninformed Search (if no traffic data): BFS (shortest path in steps), Uniform Cost Search (least distance). ● Informed Search (with traffic & distance): A* Search (uses both distance + heuristic like estimated time). Example: A* uses ● g(n) = actual distance/time so far ● h(n) = estimated time remaining based on map data ● f(n) = g(n) + h(n) to pick the best next step. 5. Plan / Path Generation Once the best path is found, the GPS generates a step-by-step plan: ● Turn left after 500m ● Take the flyover ● Continue for 2 km ● Reach Marina Beach 6. Execution & Monitoring The agent monitors whether the user follows the path: ● If the user misses a turn, the GPS re-plans (restarts search with new initial state = current position). ● If traffic updates come in, the GPS recalculates the best		

	route dynamically.		
3.	Explain the formulation of a problem in AI with an example. In Artificial Intelligence, problem formulation is the process of defining a problem in such a way that it can be solved by an intelligent agent. It is the first step in problem-solving, where the real-world situation is converted into a well-defined model that an AI system can understand and work on. A well-formulated problem specifies the starting point, the desired goal, the set of possible actions, and the criteria for evaluating solutions. The major components of problem formulation are: 1. Initial State – the starting point of the problem. 2. Goal State – the condition that defines the success of the problem. 3. Actions – the possible operations or moves available to the agent. 4. Transition Model – the description of what result each action will produce. 5. Path Cost – a function that assigns a numeric cost to each path, used to identify the most efficient solution. Example: Route-Finding Problem Consider the example of a GPS navigation system. Here, the initial state is the current location of the user, and the goal state is the desired destination. The actions are the possible movements through roads, such as turning left, moving straight, or taking a flyover. The transition model defines the outcome of each action, for instance, if the agent turns left at a junction, it will reach a specific new location. The path cost can be defined in terms of distance, travel time, or fuel consumption. By clearly formulating the problem in this structured manner, the AI system can apply suitable search algorithms such as Uniform Cost Search or A* to find the most optimal route. Thus, problem formulation ensures that an AI agent can convert a real-world problem into a structured search problem and solve it effectively.	CO1	K4
4.	Apply the Breadth-First Search algorithm to solve the 8-puzzle		K4

	problem (or any suitable example). Illustrate the state space and explain the traversal.
	
	● Basic Idea:
	BFS explores the search tree level by level. It starts at the root (initial state), expands all nodes at depth 1, then all nodes at depth 2, and so forth.
	● Data Structure Used:
	A FIFO queue is used to keep track of nodes to be expanded, ensuring nodes are expanded in order of increasing depth.
	● Process:
	1. Start by enqueueing the initial node and marking it as visited.
	2. Dequeue the front node, expand it by generating all its successors.
	3. For each successor, if it has not been visited or enqueued before, enqueue it and mark as visited.
	4. Repeat until the goal is found or the queue is empty.
	● Goal Test:
	Applied when a node is generated (not when expanded), so BFS returns the shallowest goal node immediately upon discovery.
	
	function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure
	node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
	if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
	frontier ← a FIFO queue with *node* as the only element
	explored ← an empty set
	loop do
	if EMPTY?(*frontier*) **then return** failure
	node ← POP(*frontier*) /* chooses the shallowest node in *frontier* */
	add *node*.STATE to *explored*
	for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
	child ← CHILD-NODE(*problem*, *node*, *action*)
	if *child*.STATE is not in *explored* or *frontier* **then**
	if *problem*.GOAL-TEST(*child*.STATE) **then return** SOLUTION(*ch*
	frontier ← INSERT(*child*, *frontier*)
	
	 
	
	<u>Intuition and Visualization</u>
	
	● BFS can be thought of as a wave spreading out from the root node, visiting all nodes at distance 1, then distance 2, and so on.
	● In graph terms, BFS finds the shortest path (fewest edges) from the source to any reachable node in an unweighted graph.
		CO1	

	<u>Example of BFS Traversal</u> Consider a graph with vertices and edges: ● Start at vertex 0. ● Enqueue 0 and mark visited. ● Dequeue 0, enqueue all unvisited neighbors (1, 2). ● Dequeue 1, enqueue its unvisited neighbors (3). ● Dequeue 2, enqueue its unvisited neighbors (4). ● Continue until all reachable nodes are visited. The order of visitation would be: 0 → 1 → 2 → 3 → 4. Example : 8 puzzle problem  		
5.	Compare the working of Depth-First Search and Breadth-First Search using a suitable example such as maze navigation. Which one is more memory efficient, and why? Depth-First Search (DFS) and Breadth-First Search (BFS) are two classical uninformed search strategies in Artificial Intelligence. Their working and efficiency can be compared using the example of maze navigation, where an agent is trying to reach an exit from the	CO1	K4

	entrance. In Breadth-First Search, the agent explores the maze level by level. Starting from the entrance, it checks all possible moves (left, right, forward) before moving deeper. For example, if the maze has several junctions, BFS first explores all nodes at depth 1, then depth 2, and so on. BFS guarantees finding the shortest path to the exit since it expands nodes in increasing order of depth. However, it requires storing all nodes in the current frontier, leading to high memory consumption. In Depth-First Search, the agent goes as deep as possible along one path before backtracking. In maze navigation, the agent would follow one corridor until it either finds the exit or encounters a dead end, in which case it backtracks and tries an alternate path. DFS requires very little memory since it only stores nodes along the current path, but it may get stuck in deep or infinite paths, and it does not guarantee finding the shortest route. Working Principle ● BFS explores the shallowest unvisited node first, level by level. ● DFS explores along a path to its maximum depth before backtracking. Completeness ● BFS is complete, meaning it will always find a solution if one exists. ● DFS is not complete in infinite or very large state spaces, as it may get stuck going down one branch. Optimality ● BFS is optimal if all step costs are equal because it always finds the shallowest (shortest) solution. ● DFS is not optimal, as it may find a longer path before discovering a shorter one. Time Complexity ● BFS: O(bd)O(b^d)O(bd), where bbb is the branching factor		

	and ddd is the depth of the shallowest solution. ● DFS: O(bm)O(b^m)O(bm), where mmm is the maximum depth of the search space (can be very large or infinite). Space Complexity (Memory Requirement) ● BFS requires O(b^d) memory because it stores all nodes in the frontier at a given level. ● DFS requires only O(bm) memory, since it stores nodes only along the current path. Suitability in Maze Navigation ● BFS is useful when the shortest path is required, such as a GPS navigation system or robot path planning. ● DFS is useful when memory is limited or when the solution is expected to be deep in the search tree.		
6.	Design A* search algorithm with an example.	CO1	K4

- **A*** search is a widely used and powerful **informed search algorithm**.
- It combines the actual cost to reach a node, $g(n)$, with a heuristic estimate of the cost from that node to the goal, $h(n)$, to evaluate nodes:

$$f(n) = g(n) + h(n)$$

- Here, $f(n)$ estimates the total cost of the cheapest solution path going through node n.

Key Concepts

- $g(n)$: Actual cost from the start node to node n.
- $h(n)$: Heuristic estimate of the cheapest cost from n to the goal.
- $f(n)$: Estimated total cost of the cheapest solution through n.

Heuristic Conditions for Optimality

- **Admissible Heuristic:**
  A heuristic $h(n)$ is admissible if it **never overestimates** the true cost to reach the goal from n.
  Example: Straight-line distance in route planning is admissible because the shortest path cannot be longer than the straight line.
- **Consistent (Monotonic) Heuristic:**
  A stronger condition for graph search, where for every node n and successor n':

$$h(n) \leq c(n, a, n') + h(n')$$

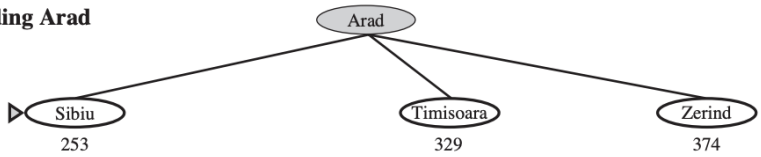
  This is a form of the triangle inequality ensuring $f(n)$ values along any path are nondecreasing.
- Every consistent heuristic is admissible, but not all admissible heuristics are consistent.

Algorithm Properties

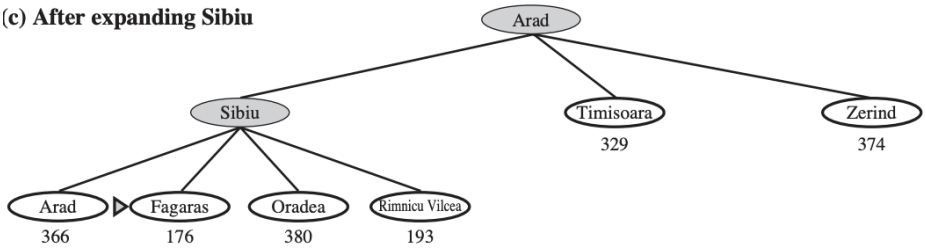
- **Completeness:**
  A* is complete if the branching factor is finite and step costs are greater than some positive constant.
- **Optimality:**
  - The tree-search version is optimal if $h(n)$ is admissible.
  - The graph-search version is optimal if $h(n)$ is consistent.
- **Optimal Efficiency:**
  Among all algorithms using the same heuristic information, A* expands the fewest nodes necessary to guarantee optimality.

(a) The initial state

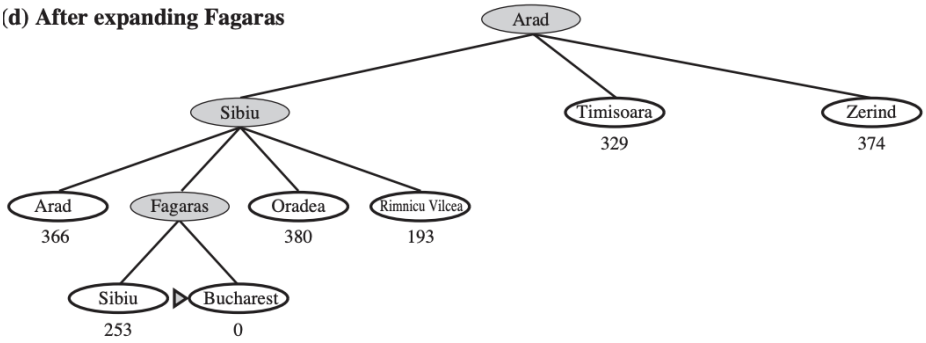
(b) After expanding Arad

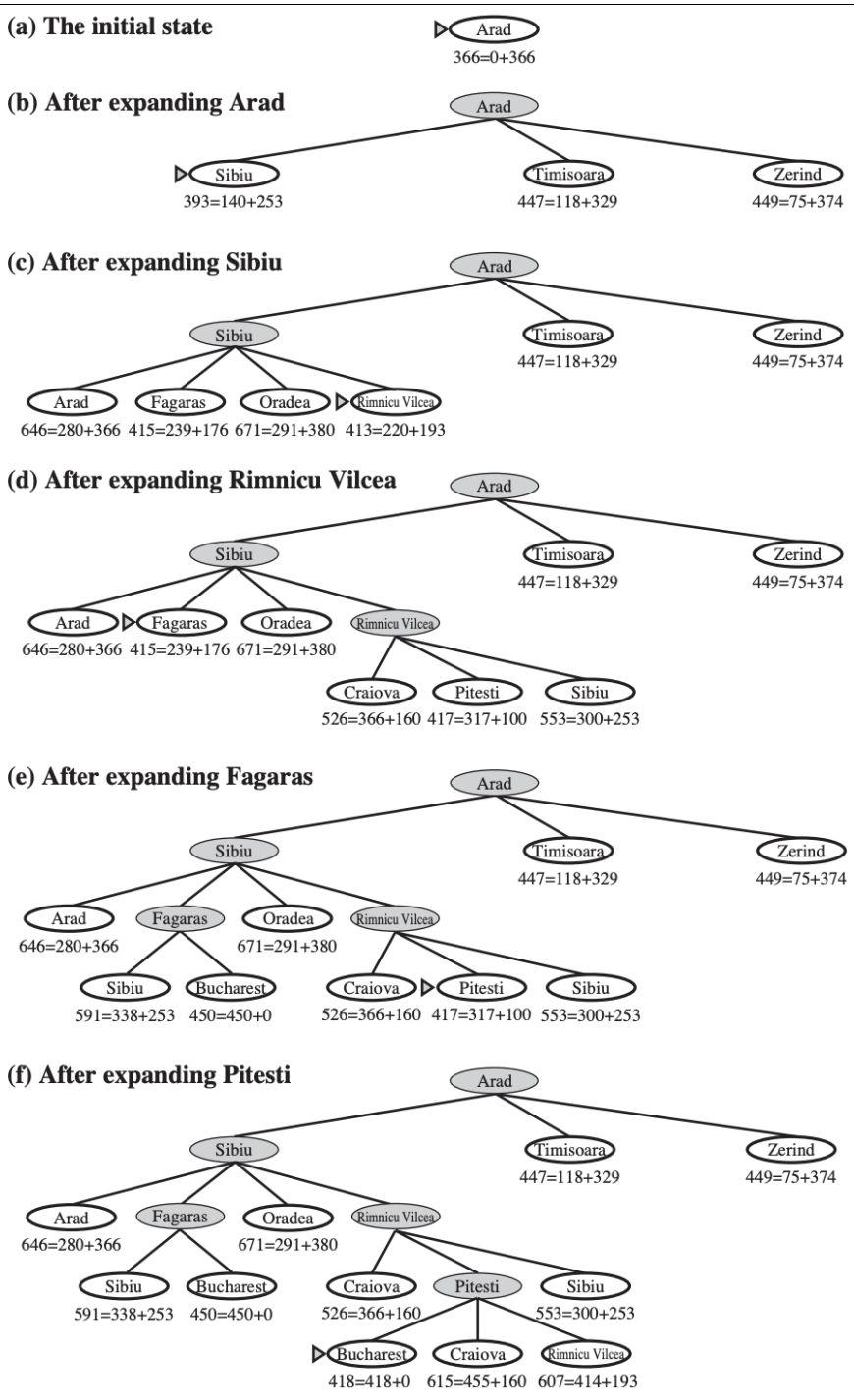


(c) After expanding Sibiu



(d) After expanding Fagaras




(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
393=140+253
447=118+329
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
413=220+193
(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
Craiova
Pitesti
Sibiu
526=366+160
417=317+100
553=300+253
(e) After expanding Fagaras
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
671=291+380
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
591=338+253
450=450+0
526=366+160
417=317+100
553=300+253
(f) After expanding Pitesti
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
671=291+380
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
591=338+253
450=450+0
526=366+160
553=300+253
Bucharest
Craiova
Rimnicu Vilcea
418=418+0
615=455+160
607=414+193

	How A* Search Works 1. Initialize the **open list** with the start node. 2. Repeatedly select the node with the lowest $f(n)$ from the open list. 3. If the node is the goal, reconstruct and return the solution path. 4. Otherwise, expand the node, generate successors, and update their g, h, f values. 5. Add successors to the open list if they are not already explored or if a cheaper path is found. 6. Continue until the goal is found or the open list is empty. **Visualization** • A* expands nodes in **contours of increasing f-cost**, similar to expanding concentric bands around the start node. • With a perfect heuristic, search is narrowly focused along the optimal path. • With $h(n) = 0$, A* reduces to uniform-cost search, expanding nodes in circular bands. **Complexity** • Time and space complexity depend on the heuristic accuracy. • The **effective branching factor** decreases with more accurate heuristics. • Worst-case complexity can still be exponential in solution depth. • Space complexity is high because A* stores all generated nodes in memory. **Practical Considerations** • A* is widely used in pathfinding (games, robotics, maps) and AI planning. • The choice of heuristic critically impacts performance. • Variants exist to reduce memory usage or find approximate solutions.		
7	Explain the role of heuristic functions in informed search strategies. Illustrate with an example how a heuristic improves search efficiency compared to uninformed search. In Artificial Intelligence, a heuristic function is a problem-specific evaluation function that provides an estimate of the cost of reaching the goal from a given state. It is denoted as h(n), where *n* is the current node. Unlike uninformed search strategies (such as Breadth-First Search and Depth-First Search), which have no information about the direction of the goal, informed search strategies use heuristics to make intelligent decisions and reduce the search effort. The role of a heuristic function is to guide the search towards the most promising states, thereby reducing the number of nodes expanded. It improves the efficiency of search algorithms by focusing on paths that are more likely to lead to the goal. A good heuristic should be admissible (never overestimates the true cost) and preferably consistent (obeys the triangle inequality). Example: Route-Finding Problem Consider a GPS navigation system where the task is to travel from City A to City G through several intermediate cities. ● In Uniform Cost Search (an uninformed search), the	CO1	K4

	algorithm explores all possible paths in order of actual cost from the start, without any knowledge of how close a state is to the goal. This may lead to exploring many unnecessary cities before reaching City G. ● In A* search (an informed search), the algorithm uses both the path cost so far, g(n), and the heuristic estimate of the cost to reach the goal, h(n). The evaluation function is: $$f(n)=g(n)+h(n)f(n)$$ If the heuristic function h(n) is chosen as the straight-line distance from the current city to the destination city, the search is intelligently guided in the right direction. For example, if City B is geographically closer to City G than City C, then A* will prefer expanding City B first, rather than blindly exploring both paths. How Heuristic Improves Search Efficiency Without heuristics, the algorithm wastes time exploring all branches, even those leading away from the goal. With heuristics, the algorithm prioritizes nodes that are likely to lead to the goal faster, thus expanding fewer states and reaching the solution more efficiently. In route-finding, this results in faster navigation and reduced computational effort.		
8	Design an intelligent agent for a smart home energy management system using the PEAS framework. Clearly identify the Performance measures, Environment, Actuators, and Sensors. In Artificial Intelligence, the PEAS framework (Performance measure, Environment, Actuators, and Sensors) is used to specify the task environment of an intelligent agent. For a Smart Home Energy Management System, the PEAS description can be given as follows: 1. Performance Measures The success of the agent is evaluated based on: ● Minimizing overall energy consumption. ● Reducing electricity bills by using appliances during off-peak hours. ● Maintaining user comfort (e.g., proper room temperature, lighting, and appliance usage).	CO1	K5

- Ensuring safety by preventing overloads and shutting down unused devices.
- Promoting sustainability by integrating renewable energy sources like solar panels.

2. Environment

The environment in which the smart home energy management agent operates includes:

- Residential building (rooms, appliances, HVAC systems, lighting, etc.).
- External electricity grid with varying peak and off-peak tariffs.
- Renewable energy sources (solar panels, batteries).
- Occupants and their usage behavior (presence/absence, preferences).
- Weather conditions that influence energy needs (temperature, sunlight).

3. Actuators

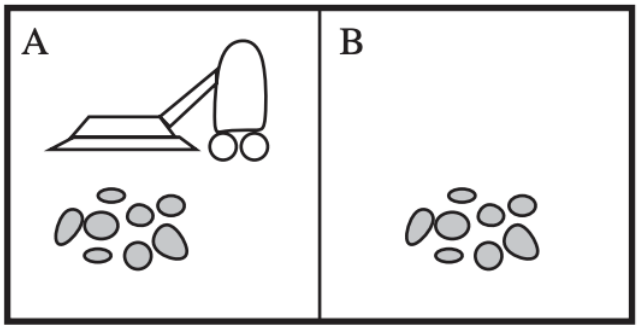
The actuators are the components that the agent controls to take action:

- Smart switches and plugs to turn devices on/off.
- Thermostats to adjust heating or cooling.
- Smart lighting systems to dim or switch lights.
- Battery storage controllers to charge or discharge energy.
- Communication systems to interact with the electricity grid for demand-response programs.

4. Sensors

	The sensors collect real-time data for the agent: ● Smart meters to measure energy consumption. ● Occupancy sensors to detect presence in rooms. ● Temperature and humidity sensors for climate control. ● Light sensors to measure natural illumination. ● Weather forecast data (from external APIs). ● Battery level and solar panel output sensors. Summary of PEAS for Smart Home Energy Management System ● Performance Measures: Minimize energy use and cost, maximize comfort, ensure safety, and promote sustainability. ● Environment: Home appliances, users, electricity grid, renewable sources, and weather conditions. ● Actuators: Smart switches, thermostats, lighting controls, storage controllers, and communication systems. ● Sensors: Smart meters, occupancy detectors, temperature/humidity sensors, light sensors, weather data, solar/battery sensors.		

Q.No	Questions	CO's	Bloom's Level
	Part C		
1.	Design an intelligent agent to operate in the Vacuum Cleaner World environment. Clearly outline the agent's percepts, actions, and internal architecture. Justify your choice of agent type and explain how it adapts to partially observable or stochastic environments.	CO1	K5



A vacuum-cleaner world with just two locations.

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

The Vacuum Cleaner World is a classical environment used to study intelligent agents. The task of the agent is to keep a set of rooms clean by perceiving its surroundings and performing suitable cleaning actions.

1. Agent's Percepts

The percepts are the inputs the agent receives from its sensors. In the Vacuum Cleaner World, these may include:

- Whether the current square is Clean or Dirty.
- The location of the agent (e.g., left room or right room in a two-room world, or coordinates in a larger grid).
- Optional advanced percepts such as battery level (in extended models) or obstacles.

2. Agent's Actions

The possible actions available to the agent are:

- Suck → Clean the current square.

- Move Left → Move to the left adjacent square.
- Move Right → Move to the right adjacent square.
- No Operation → Do nothing if the environment is already clean.

3. Agent's Internal Architecture

The design of the vacuum cleaner agent can be implemented using different types of AI agent architectures:

- Simple Reflex Agent:
  The agent acts only based on the current percept. For example, "If the square is dirty, then suck; otherwise, move to the next square." This works well in a small, deterministic, and fully observable environment.
- Model-Based Reflex Agent:
  If the environment is partially observable (e.g., the agent cannot see whether the other square is dirty), it maintains an internal model of the environment. The model records which squares have already been cleaned and where the agent is located.
- Goal-Based Agent:
  The agent is given the goal of keeping all squares clean. It decides actions based not only on current percepts but also on whether its goal is satisfied.
- Utility-Based Agent:
  In a stochastic environment (where cleaning may fail sometimes or dirt may reappear), the agent uses a utility function to balance efficiency (less movement) with cleanliness. It chooses actions that maximize expected performance, such as minimizing energy usage while keeping rooms clean.

4. Justification of Agent Type

- For a fully observable and deterministic environment, a simple reflex agent is sufficient since the agent always knows the current state and can directly act.
- For a partially observable environment, a model-based reflex agent is more appropriate, as it can remember past percepts and estimate the current world state.

	● In stochastic environments, where dirt may randomly appear or cleaning may sometimes fail, a utility-based agent is best, since it adapts to uncertainty and optimizes long-term performance.		
2.	Given a goal of finding the shortest path in a dynamic urban environment (e.g., autonomous car navigation), compare and critically evaluate the effectiveness of various search strategies such as BFS, DFS, UCS, and A. Support your evaluation with suitable examples and performance metrics like completeness, optimality, time, and space complexity In autonomous car navigation, the task is to find the shortest and most efficient path in a dynamic urban setting with traffic, signals, and possible obstacles. Different search strategies can be applied to solve this, and their effectiveness can be evaluated based on completeness, optimality, time complexity, and space complexity. 1. Breadth-First Search (BFS) BFS explores all nodes level by level from the starting point. ● Completeness: BFS is complete because it will always find a solution if one exists. ● Optimality: BFS is optimal only when all step costs are equal (e.g., assuming each road segment has the same travel cost). ● Time Complexity: O(b^d), where b is the branching factor and ddd is the depth of the shallowest solution. ● Space Complexity: O(b^d), as it stores all nodes at each level. ● Example: In city navigation, BFS could find a path, but it would treat a 1 km road and a 5 km road as equal, so it may not find the *true shortest* path in terms of distance or time. 2. Depth-First Search (DFS) DFS explores one path deeply before backtracking. ● Completeness: DFS is not complete in infinite or very large graphs (it can get stuck exploring one long route). ● Optimality: DFS is not optimal, as it may find a much longer route instead of the shortest one.	CO1	K5

- Time Complexity: O(b^m), where mmm is the maximum depth (can be very large).
- Space Complexity: O(bm), which is more memory-efficient than BFS.
- Example: An autonomous car using DFS might go far into the wrong direction (e.g., a long highway out of the city) before backtracking, wasting time and fuel.

3. Uniform Cost Search (UCS)

UCS expands the node with the lowest cumulative cost from the start.

- Completeness: UCS is complete as long as the step costs are positive.
- Optimality: UCS is optimal because it always expands the least-cost path first.
- Time Complexity: $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$, where C* is the cost of the optimal solution and εεε is the minimum step cost.
- Space Complexity: Can be high, as it stores all frontier nodes in a priority queue.
- Example: In urban navigation, UCS would prefer shorter and faster roads, and will guarantee the shortest route in terms of distance or time. However, it may expand many nodes unnecessarily before reaching the goal.

4. A* Search

A* combines the actual cost from the start, g(n)g(n)g(n), and the heuristic estimate to the goal, h(n)h(n)h(n). The evaluation function is:

$$f(n)=g(n)+h(n)f(n)$$

Completeness: A* is complete if the heuristic is admissible (never overestimates).
Optimality: A* is optimal if the heuristic is admissible and consistent.

- Time Complexity: Depends heavily on the quality of the heuristic. With a good heuristic, far fewer nodes are

	expanded compared to UCS. Worst-case is still exponential. ● Space Complexity: Requires storing all generated nodes, so memory can be a limitation. ● Example: In autonomous driving, if the heuristic is straight-line distance or predicted travel time (based on traffic data), A* will efficiently guide the car toward the destination while avoiding unnecessary exploration. It balances optimality with efficiency better than UCS. Critical Evaluation ● BFS is too simplistic for urban navigation, as roads differ in distance and cost. It is inefficient in real-world conditions. ● DFS is highly memory efficient but unreliable, as it may miss the shortest or even any solution in a vast urban grid. ● UCS guarantees the shortest path but can be computationally expensive in large cities with many intersections. ● A* is the most effective strategy in dynamic urban environments, as it uses heuristics (like traffic estimates or straight-line distances) to guide the search, drastically reducing the number of explored nodes while still guaranteeing the optimal path.		

Unit-II Knowledge Representation and Reasoning

Q.No	Questions	CO's	Bloom's Level
	PART A		
1.	Define game playing? Game playing in AI is the process of designing agents that can make intelligent decisions in competitive environments by using search and decision-making strategies (e.g., Minimax) to choose optimal moves.	CO2	K1
2.	What is Mini –Max Strategy? The Mini–Max strategy is a decision-making algorithm used in two-	CO2	K1

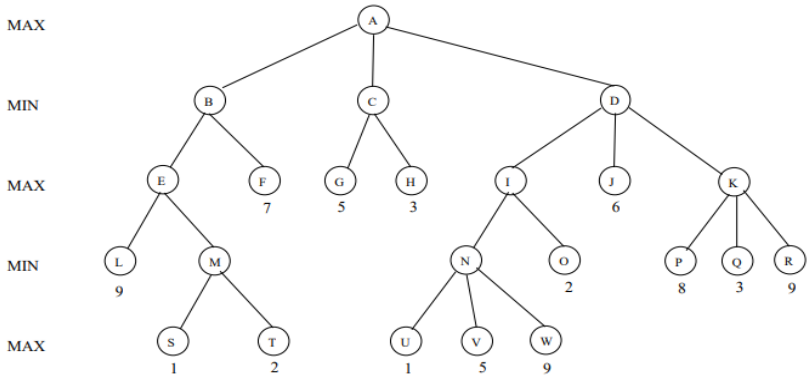
	player games. It assumes that one player (Max) tries to maximize their gain while the opponent (Min) tries to minimize it. The algorithm explores all possible moves, evaluates them, and selects the move that ensures the best outcome against the opponent's optimal play.		
3.	How is Knowledge represented? Knowledge can be represented using: 1. Logic (Propositional and First-Order Logic) 2. Semantic Networks 3. Frames 4. Production Rules 5. Ontologies	CO2	K1
4.	Summarize propositional logic? Definition: Propositional logic is a formal system where statements (propositions) are represented as symbols that can be either *true* or *false*. Basic Elements: ● Propositions (P, Q, R, …) – atomic statements. ● Logical Connectives: ○ AND (∧) ○ OR (∨) ○ NOT (¬) ○ IMPLIES (→) ○ BICONDITIONAL (↔)	CO2	K2
5.	Explain in detail about resolution ? Resolution is a rule of inference used in propositional logic and first-order logic (FOL) to prove the unsatisfiability of a set of clauses. It is based on the principle of refutation: if we want to prove a statement, we assume its negation and show that this leads to a contradiction.	CO2	K2
6.	What are the limitations of Propositional Logic? Propositional logic can represent facts using simple true or false statements, but it cannot express relationships between objects or handle complex domains.	CO2	K1

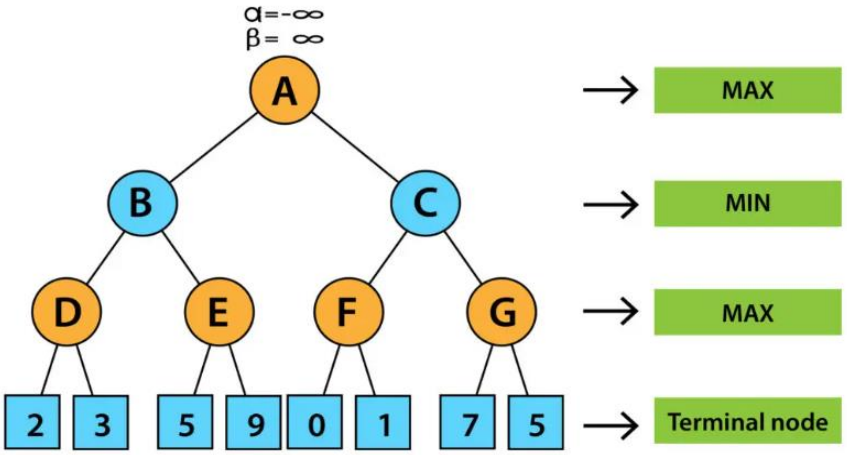
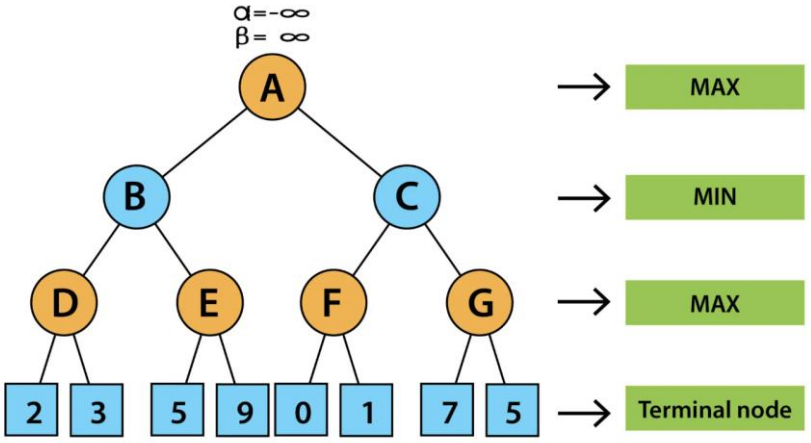
7.	Define pruning? Pruning refers to the process of eliminating branches of the search tree that do not need to be explored because they cannot influence the final decision. Pruning reduces the number of nodes evaluated, improves efficiency, and saves time and memory (e.g., Alpha–Beta pruning in the Mini-Max algorithm).	CO2	K1
8.	Explain the PEAS description of Wumpus World problem? **Component** / **Description** **Performance Measure**: Agent should find the gold and come out alive. It should minimize actions, avoid falling into pits, and avoid being eaten by the Wumpus. Maximum score is achieved by safely exiting with gold. **Environment**: A grid-based cave with rooms. Some rooms contain pits (danger), one room has the Wumpus (monster), and one room has gold (goal). Agent perceives environment through local sensory information (stench, breeze, glitter, bump, scream). **Actuators**: Agent's actuators include movement (forward, turn left, turn right), grabbing (to pick gold), shooting arrow (to kill Wumpus), and climbing out of the cave. **Sensors**: Agent perceives stench (near Wumpus), breeze (near pit), glitter (gold in current room), bump (into wall), and scream (Wumpus killed).	CO2	K2
9.	List out the basic elements of First-Order logic? 1. **Constants** – Represent specific objects (e.g., John, Apple). 2. **Variables** – Represent general objects (e.g., x, y, z). 3. **Predicates** – Represent relations or properties of objects (e.g., Loves(x, y), IsHuman(x)). 4. **Functions** – Map objects to objects (e.g., Mother(x), Square(x)). 5. **Quantifiers** – Express generality or existence: • Universal (∀) – "for all" • Existential (∃) – "there exists" 6. **Connectives** – Logical operators: • ∧ (AND), ∨ (OR), ¬ (NOT), → (IMPLIES), ↔ (IFF). 7. **Equality (=)** – States that two terms are equal. 8. **Sentences / Expressions** – Well-formed formulas built from the above elements.	CO2	K1
10.	Name the two types of quantifiers and explain with an example? 1. Universal Quantifier (∀) ● Meaning: "For all" or "For every." ● It states that the given property is true for all elements in the domain. ● Example: ∀x Human(x) → Mortal(x)	CO2	K1

	→ "All humans are mortal." 2. Existential Quantifier (∃) ● Meaning: "There exists" or "For some." ● It states that the given property is true for at least one element in the domain. ● Example: ∃x Loves(x, IceCream) → "There exists someone who loves ice cream."		
11.	Compare Forward and Backward chaining. **Aspect** | **Forward Chaining** | **Backward Chaining** **Direction of Reasoning** | Data-driven (from facts to conclusions) | Goal-driven (from goals to facts) **Starting Point** | Starts with known facts in the knowledge base | Starts with the goal (query) that needs to be proven **Process** | Applies inference rules to existing facts until a conclusion (goal) is reached | Works backward by checking which rules can satisfy the goal, then recursively proves sub-goals	CO2	K2
12.	Discuss in detail about resolution inference rule with its equation. Resolution is a single, sound inference rule used for automated reasoning in propositional logic and first-order logic (FOL). It proves a goal by refutation: add the negation of the goal to the knowledge base (KB), convert everything to CNF, and repeatedly apply resolution until you derive the empty clause (⊥), which signals a contradiction—hence the goal is entailed.	CO2	K2
13.	Give an outline of neuro-fuzzy inference? How does it combine neural networks and fuzzy logic? A Neuro-Fuzzy Inference System (NFIS) is a hybrid intelligent system that combines the human-like reasoning style of fuzzy logic with the learning and connectionist structure of neural networks. It is primarily used for decision-making, prediction, and control tasks where both uncertainty handling and learning from data are crucial.	CO2	K2
14.	Interpret about neural network in AI? How does it mimic the human brain? A Neural Network in AI is a computational model inspired by the structure and functioning of the human brain. It consists of interconnected processing units called neurons, organized in layers (input, hidden, and output). Each connection has a weight that determines the strength of influence between neurons. When data is fed into the network, the neurons process it by applying mathematical functions (activation functions) and pass the results forward through	CO2	K2

	the layers.		
15.	Write the semantics of Bayesian network? A Bayesian Network (BN) is a graphical model that represents probabilistic relationships among a set of variables using directed acyclic graphs (DAGs). The semantics of a Bayesian Network explain how the graph structure and probabilities define a joint probability distribution (JPD) over all variables. Consider a simple BN with 3 variables: • C = Cloudy • R = Rain • W = Wet Grass Structure: $C \rightarrow R \rightarrow W$ Semantics: • $P(C)$ = probability of cloudy • $P(R \mid C)$ = probability of rain given cloudy • $P(W \mid R)$ = probability of wet grass given rain Joint Probability: $$P(C, R, W) = P(C) \cdot P(R \mid C) \cdot P(W \mid R)$$	CO2	K2

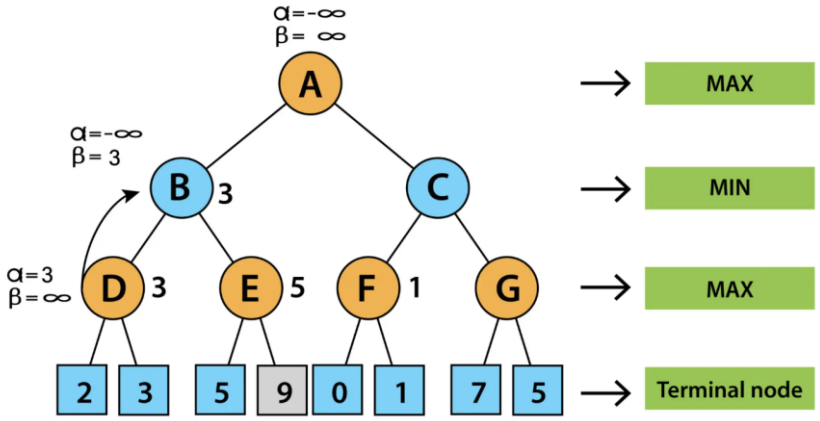
Q.No	Questions	CO's	Bloom's Level
Part – B			

1.	 Interpret the solution for the above problem using Min-Max Algorithm We evaluate the tree bottom-up, computing each internal node's value according to its type (MAX nodes take the maximum of children; MIN nodes take the minimum). The numbers shown at the leaves are the utilities. **Left subtree (rooted at B, a MIN node)** • Leaf values: $L = 9,\ S = 1,\ T = 2,\ F = 7$. • Node M is a MIN over $S(1)$ and $T(2)$: $M = \min(1, 2) = 1$. • Node E is a MAX over $L(9)$ and $M(1)$: $E = \max(9, 1) = 9$. • Node B is a MIN over $E(9)$ and $F(7)$: $B = \min(9, 7) = 7$. → Thus the left child of A has value **7**. (Optimal move at B is to choose child **F** giving 7.) **Middle subtree (rooted at C, a MIN node)** • Leaves: $G = 5,\ H = 3$. • $C = \min(5, 3) = 3$. → Middle child of A has value **3**. **Right subtree (rooted at D, a MIN node)** • Leaves: $U = 1,\ V = 5,\ W = 9,\ O = 2,\ J = 6,\ P = 8,\ Q = 3,\ R = 9$. • N is MIN over $U(1), V(5), W(9)$: $N = \min(1, 5, 9) = 1$. • I is MAX over $N(1)$ and $O(2)$: $I = \max(1, 2) = 2$. • K is MAX over $P(8), Q(3), R(9)$: $K = \max(8, 3, 9) = 9$. • D is MIN over $I(2), J(6), K(9)$: $D = \min(2, 6, 9) = 2$. → Right child of A has value **2**. (Optimal move at D is to choose child **I** giving 2.) **Root (A, a MAX node)** • A chooses the maximum among its three children: $A = \max(B(7), C(3), D(2)) = 7$. → The minimax value at the root is **7**, and the optimal first move for MAX is to go to **B** (and at B MIN's choice the resulting optimal leaf is **F** with utility 7). **Summary / Final decision** • Minimax value at root A = **7**. • Optimal play leads MAX → **B**, and at B the minimizer's best choice (to minimize MAX's payoff) results in value **7** (via child **F**). • Thus, under perfect play, the outcome utility is **7**.	CO2	K5

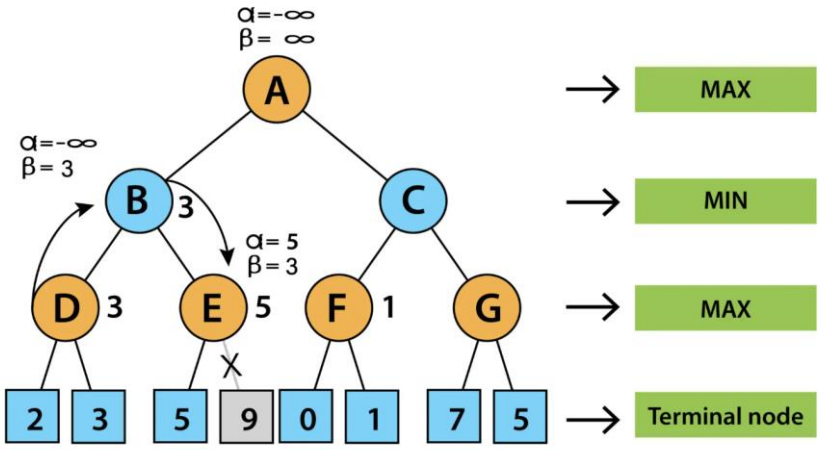
2.	 Interpret the solution for the above problem using α-β pruning Algorithm **1.** First, we will take care of the first move. So initially, we will define the worst case α = −∞ and β = +∞. If alpha is greater than or equal to beta, we will prune the node.  **2.** We didn't prune it since the initial value of alpha is less than beta. Now it's turn for MAX. Therefore, we will calculate the value of alpha at node D. At node D, the value of alpha will be max (2, 3) = 3.	CO2	K5

3. Now, node B's turn is MIN. That means that the value of alpha beta at node B will be min (3, ∞). Therefore, alpha will be – ∞ and beta will be 3 at node B.

Next, algorithms will pass the values of α= -∞ and β= 3 to the next successor of Node B, that is node E.



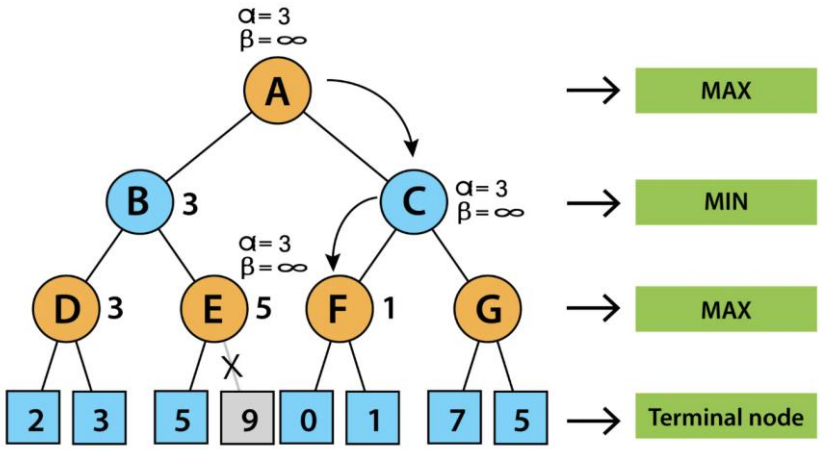
4. Now it's turn for MAX. Therefore, we will search for MAX at node E. The value of alpha at E is – ∞ and will be compared with 5. So, MAX (- ∞, 5) will be 5. Thus, at node E, alpha = 5, Beta = 5.



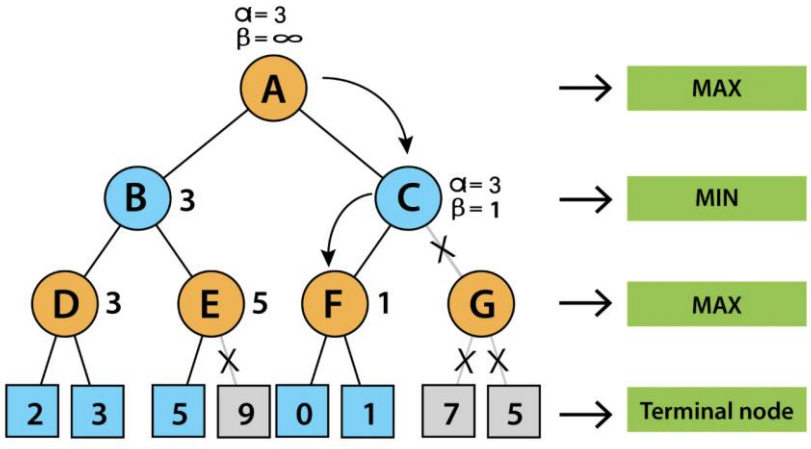
We now know that alpha is greater than beta and is also a pruning condition, so we can prune the right successor of node E, and the algorithm will not be visited, and the value of node E will be 5.

5. In the next step, the algorithm again comes from node B to node A. The alpha of node A is changed to the max value of MAX(- ∞, 3). Now, alpha and beta at node A are (3, + ∞) and will be transmitted to node C. The same values will be transferred to node F.

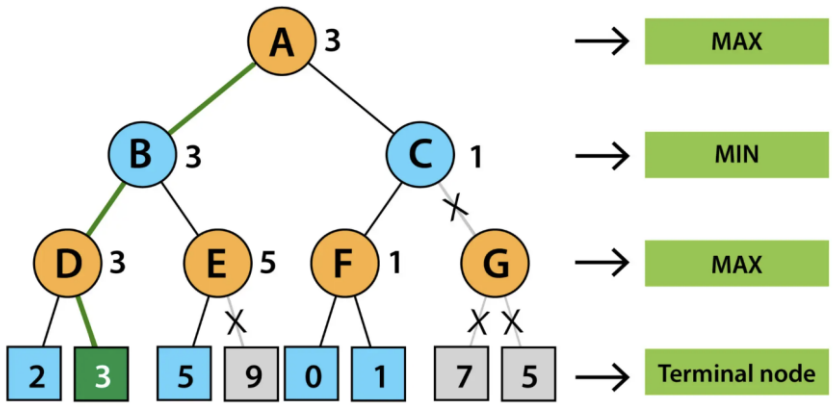
6. The value of alpha will be compared to the left branch, which is 0 at node F. Now, MAX (0, 3) will be 3 and is compared with the right child (1) and MAX(3, 1) = 3 still α is 3, but the node value of F will be 1.



7. Afterward, node F will pass the node value 1 to C and compare it to the beta value at C. Now it's turn for MIN. So, MIN (+ ∞, 1) will be 1. Now, at node C, α = 3, β= 1, and alpha is greater than beta, which again satisfies the pruning condition. Then, the successor of node C, G, will be pruned, and the algorithm did not calculate the whole subtree G.



Now, C will return its node value to A, and A will calculate MAX(1, 3), resulting in 3.

<table>
<tr>
<td></td>
<td>


This above-represented tree is the final tree that has nodes that are computed and nodes that are not computed. Therefore, in this example, the optimal value of the maximizer would be 3.
</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Interpret the types of inference rules with appropriate examples along with its truth table.
<table>
<tr><th>Rule of Inference</th><th>Form</th><th>Tautology</th><th>Description</th></tr>
<tr><td>Modus Ponens (MP)</td><td>If p → q and p, then q.</td><td>p ∧ (p → q)) → ¬q</td><td>If P implies Q, and P is true, then Q is true.</td></tr>
<tr><td>Modus Tollens (MT)</td><td>If p → q and ¬q, then ¬p.</td><td>(¬q ∧ (p → q)) → ¬p</td><td>If P implies Q, and Q is false, then P is false.</td></tr>
<tr><td>Hypothetical Syllogism (HS)</td><td>If p → q and q → r, then p → r.</td><td>((p → q) ∧ (q → r)) → (p → r)</td><td>If P implies Q and Q implies R, then P implies R.</td></tr>
<tr><td>Disjunctive Syllogism (DS)</td><td>If p ∨ q, and ¬p, then q.</td><td>(¬p ∧ (p ∨ q)) → q</td><td>If P or Q is true, and P is false, then Q is true.</td></tr>
<tr><td>Conjunction (Conj)</td><td>If p and q, then p ∧ q.</td><td>(p ∧ q) → (p ∧ q) or p → (q → (p ∧ q))</td><td>If P and Q are true, then P and Q are true.</td></tr>
<tr><td>Simplification (Simp)</td><td>If p ∧ q, then p</td><td>(p ∧ q) → p</td><td>If P and Q are true, then P is true</td></tr>
<tr><td>Addition (Add)</td><td>If p, then p ∨ q</td><td>p → (p ∨ q)</td><td>If P is true, then P or Q is true.</td></tr>
<tr><td>Absorption(Abs)</td><td>If p → q, then p → (p ∧ q)</td><td>(p → q) → (p → (p ∧ q))</td><td>If P implies Q, then P implies P or Q is true.</td></tr>
<tr><td>Resolution</td><td>If p ∨ q, and ¬p ∨ r, then q ∨ r.</td><td>p ∨ q, ¬p ∨ r ⇒ q ∨ r</td><td>If P or Q is true, and not P or R is true, then Q or R is true.</td></tr>
</table>
</td>
<td>CO2</td>
<td>K5</td>
</tr>
<tr>
<td>4.</td>
<td>Explain in detail about the Wumpus world problem and prove it using Knowledge Representation?

A cave of rooms connected by passageways.
Hazards:

Wumpus – a monster that eats intruders.

</td>
<td>CO2</td>
<td>K3</td>
</tr>
</table>

- **Bottomless pits** – fatal if entered.

- Rewards:
  - **Gold** – can be collected for a high score.
- Agent can **shoot the wumpus** (only one arrow).

A simple but powerful environment to illustrate:

- **Reasoning with uncertainty**.
- **Partially observable environments**.
- **Knowledge-based decision-making**.

PEAS Description

- **Performance Measure:**
  +1000 for exiting with gold
  –1000 for dying (pit or wumpus)
  –1 per action, –10 for shooting the arrow.
  Game ends when agent dies or exits.
- **Environment:**
  4×4 grid, agent starts at [1,1], facing right.
  Wumpus and gold placed randomly (not at start).
  Each non-start square has a 20% chance of a pit.
- **Actuators:**
  - `Forward`
  - `TurnLeft` / `TurnRight`
  - `Grab` (for gold)
  - `Shoot` (one arrow, straight line, kills wumpus)
  - `Climb` (exit, only from [1,1])
- **Sensors:** (each returns 1 bit of info)
  - **Stench**: in same or adjacent square to wumpus.
  - **Breeze**: adjacent to pit.
  - **Glitter**: in gold's square.
  - **Bump**: hits wall.
  - **Scream**: wumpus killed.

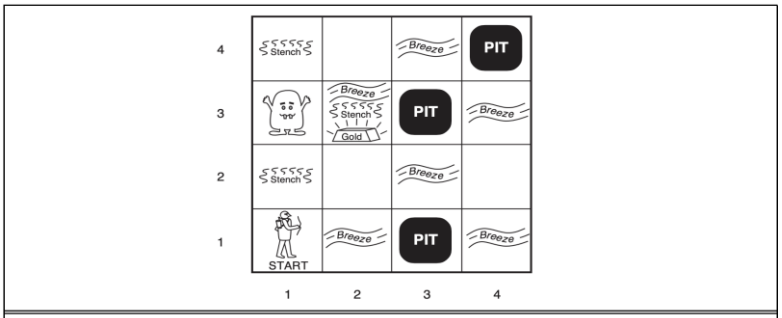


Figure 7.2 A typical wumpus world. The agent is in the bottom left corner, facing right.

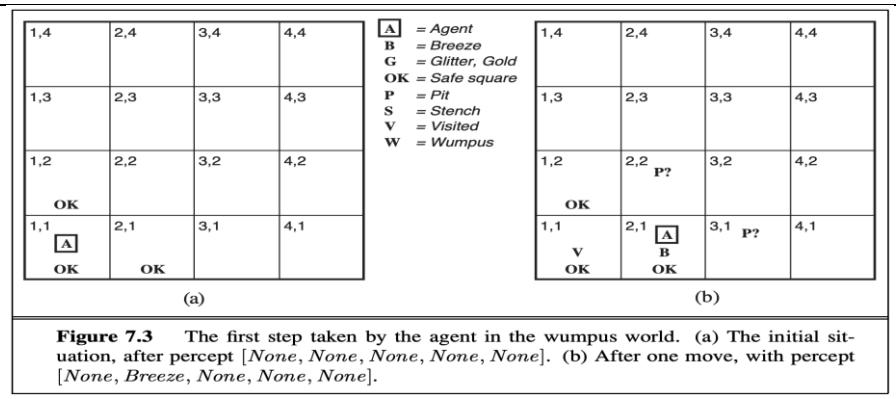


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None*, *None*, *None*, *None*, *None*]. (b) After one move, with percept [*None*, *Breeze*, *None*, *None*, *None*].

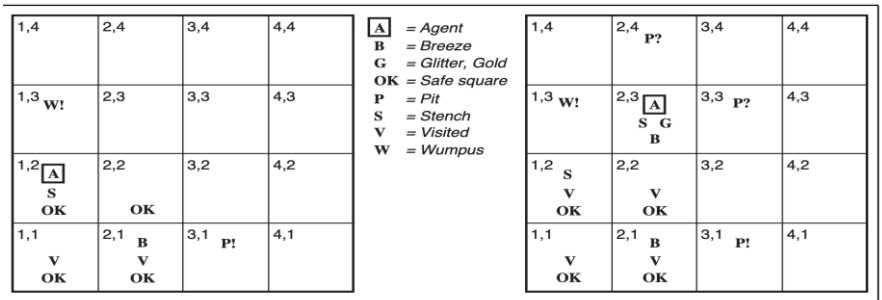


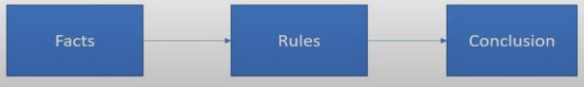
Figure 7.4 Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench*, *None*, *None*, *None*, *None*]. (b) After the fifth move, with percept [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

5.	Interpret in detail about FOL inference rules for quantifier with an example. Quantifiers in FOL are: • **Universal quantifier (∀)** → "for all" • **Existential quantifier (∃)** → "there exists" Inference rules help us move between quantified statements and specific instances.	CO2	K5

	1. Universal Instantiation (UI) • **Rule**: If something is true for **all elements**, then it is true for any particular element. • **Equation**: $$\forall x\, P(x) \longrightarrow P(c)$$ where **c** is any arbitrary constant. • **Example**: $\forall x\, Loves(x, IceCream)$ → "Everyone loves Ice Cream." By **UI**, we can infer: $Loves(John, IceCream)$ **2. Universal Generalization (UG)** • **Rule**: If something is true for an **arbitrary element**, then it is true for all elements. • **Equation**: $$P(c) \longrightarrow \forall x\, P(x)$$ provided **c** is arbitrary (not special). • **Example**: Suppose we proved $P(c)$ for an arbitrary person c. Then we can conclude $\forall x\, P(x)$. **3. Existential Instantiation (EI)** • **Rule**: If something is true for **some element**, then we can replace it with a new constant (a Skolem constant). • **Equation**: $$\exists x\, P(x) \longrightarrow P(c)$$ where **c** is a new constant (not used earlier). • **Example**: $\exists x\, HasPet(x)$ → "Someone has a pet." By **EI**, we infer: $HasPet(Alice)$ (where Alice is a new constant). **4. Existential Generalization (EG)** • **Rule**: If something is true for a particular constant, then it is true for **some element**. • **Equation**: $$P(c) \longrightarrow \exists x\, P(x)$$ • **Example**: $Loves(John, IceCream)$ By **EG**, we infer: $\exists x\, Loves(x, IceCream)$ → "Someone loves Ice Cream."		
6.	Explain in detail about forward chaining along with its proof?	CO2	K5

Forward Chaining

- Forward chaining starts with the available data and uses inference rules to extract more data until a goal is reached.
- **"What will happen next?"**
- It is a bottom up approach.
- Forward chaining is data driven because the reasoning starts from a set of data.



Forward Chaining-Example

- Example

  It is raining.
  If it is raining, the street is wet.
  The street is wet.

- Conclude from "A" and "A implies B" to "B"

  A

  A->B

  B

Example

- The law says that it is a crime for an American to sell weapons to hostile nations. Country Q, an enemy of America, has some missiles, and all of its missiles were sold to it by Jack, who is American.
- Prove that **"Jack is criminal".**

- it is a crime for an American to sell weapons to hostile nations

  $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- Q has some missiles

  $\exists x\ Owns(Q,x) \wedge Missile(x)$

  $Owns(Q,M1)$

  $Missile(M1)$

Conversion into FOL

- all of its missiles were sold to it by Jack
  $\forall x\ Missile(x) \wedge Owns(Q,x) \Rightarrow Sells(Jack,x,Q)$
  $Missile(x) \wedge Owns(Q,x) \Rightarrow Sells(Jack,x,Q)$
- Missiles are weapons
  $Missile(x) \Rightarrow Weapon(x)$
- Enemies of America are hostile
  $Enemy(x, America) \Rightarrow Hostile(Q)$
- Jack, who is American
  $American(Jack)$
- The country Q, an enemy of America
  $Enemy(Q,America)$

Step 1-Forward Chaining

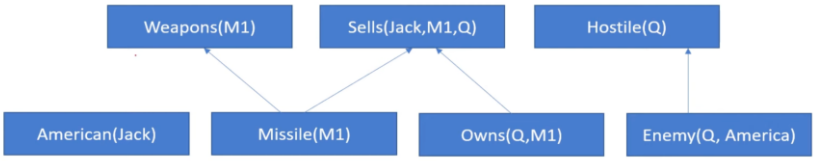
$Owns(Q,M1)$
$Missile(M1)$ => **Known Facts**
$American(Jack)$
$Enemy(Q, America)$

Forward Chaining->Facts,Rules,Conclusion

1. $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
2. $Owns(Q,M1)$
3. $Missile(M1)$
4. $Missile(x) \wedge Owns(Q,x) \Rightarrow Sells(Jack,x,Q)$
5. $Missile(x) \Rightarrow Weapon(x)$
6. $Enemy(x, America) \Rightarrow Hostile(Q)$
7. $American(Jack)$
8. $Enemy(Q,America)$

Step 2-Forward Chaining

- Missile(x) ⇒ Weapon(x)
- ∀x Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q) => **Rules that satisfy the facts**
- Enemy(x, America) ⇒ Hostile(Q)

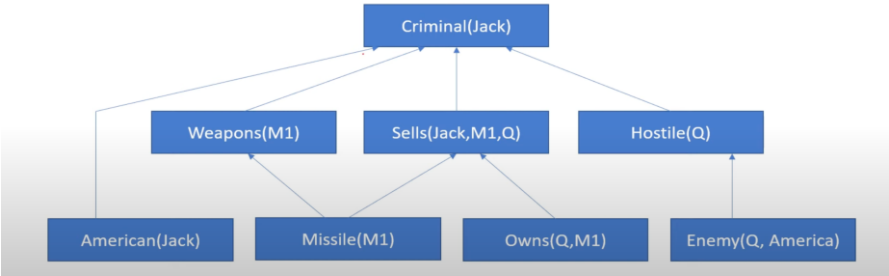


Forward Chaining->Facts,Rules,Conclusion

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

Step 3-Forward Chaining

- American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x) **Proof**



7	Explain in detail about backward chaining along with its proof?	CO2	K5

Backward Chaining
Conclusion,Rules,Facts

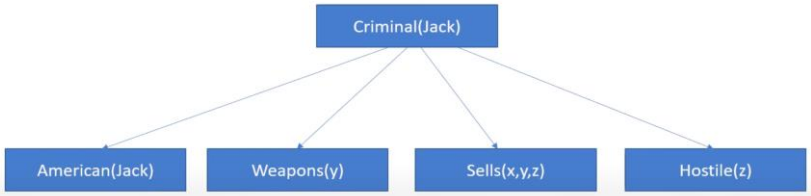
1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

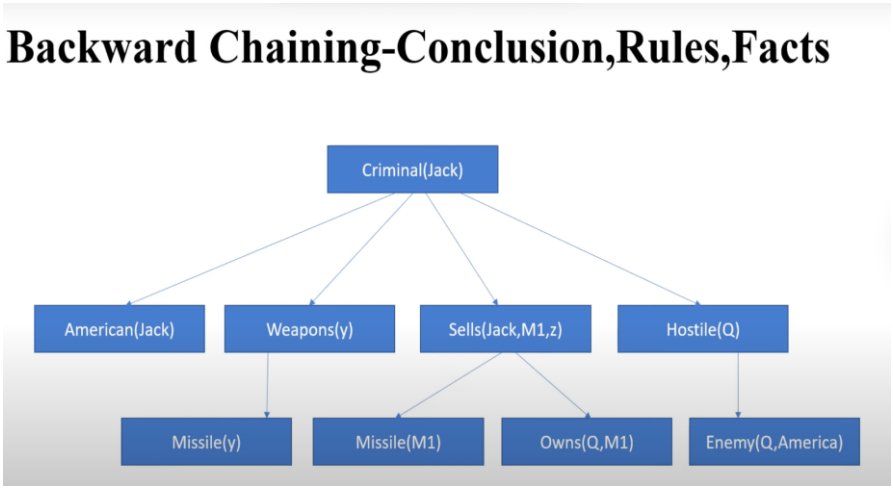
Backward Chaining-Conclusion,Rules,Facts

Backward Chaining
Conclusion,Rules,Facts

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

Backward Chaining-Conclusion,Rules,Facts



			
8	Solve the below given problem using resolution FOL	CO2	K3

a. ∀x: food(x) → likes(John, x)

b. food(Apple) ∧ food(vegetables)

c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil).

e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**

g. ∀x: alive(x) →¬ killed(x) }

h. likes(John, Peanuts)

Write down the Knowledge Base

Convert everything to CNF

a) `food(x) → likes(John, x)`

→ **C1:** `¬food(x) ∨ likes(John, x)`

b) `food(Apple) ∧ food(Vegetables)`

→ **C2:** `food(Apple)`

→ **C3:** `food(Vegetables)`

c) `(eats(x,y) ∧ ¬killed(x)) → food(y)`

Eliminate → : `¬(eats(x,y) ∧ ¬killed(x)) ∨ food(y)`

Push ¬ in: `¬eats(x,y) ∨ killed(x) ∨ food(y)`

→ **C4:** `¬eats(x,y) ∨ killed(x) ∨ food(y)`

d) `eats(Anil, Peanuts) ∧ alive(Anil)`

→ **C5:** `eats(Anil, Peanuts)`

→ **C6:** `alive(Anil)`

e) `eats(Anil, x) → eats(Harry, x)`

→ **C7:** `¬eats(Anil, z) ∨ eats(Harry, z)`

f) `¬killed(x) → alive(x)`

→ **C8:** `killed(x) ∨ alive(x)`

g) `alive(x) → ¬killed(x)`

→ **C9:** `¬alive(x) ∨ ¬killed(x)`

Negated query (for refutation):

h) add **C10:** `¬likes(John, Peanuts)`

R1. Use "alive ⇒ not killed" for Anil

- C6: `alive(Anil)`
- C9: `¬alive(x) ∨ ¬killed(x)`
- Unifier: `{x/Anil}`
- **Resolvent R1:** `¬killed(Anil)`

R2. Use the "eats & not killed ⇒ food" rule

- C5: `eats(Anil, Peanuts)`
- C4: `¬eats(x,y) ∨ killed(x) ∨ food(y)`
- Unifier: `{x/Anil, y/Peanuts}`
- **Resolvent R2:** `killed(Anil) ∨ food(Peanuts)`

	R3. Combine R1 and R2 to get food(Peanuts) • R1: `¬killed(Anil)` • R2: `killed(Anil) ∨ food(Peanuts)` • **Resolvent R3:** `food(Peanuts)` **R4. Use "if food then John likes it"** • R3: `food(Peanuts)` • C1: `¬food(x) ∨ likes(John, x)` • Unifier: `{x/Peanuts}` • **Resolvent R4:** `likes(John, Peanuts)` **R5. Refute the negated goal** • R4: `likes(John, Peanuts)` • C10: `¬likes(John, Peanuts)` • **Resolvent: ⊥ (empty clause / contradiction)** Because adding **¬likes(John, Peanuts)** led to a **contradiction**, our original query is **entailed** by the knowledge base: ✅ **Therefore,** `likes(John, Peanuts)` **is true.**		

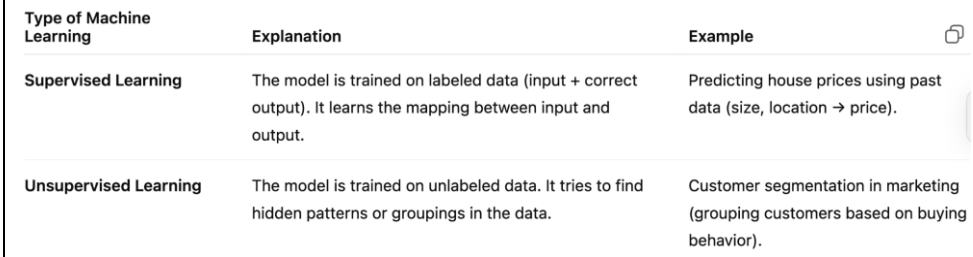
Q.No	Questions	CO's	Bloom's Level
Part c			
1.	Interpret in detail about Applications of Bayesian Network in AI □ Bayesian Network (BN) Recap A Bayesian Network is a probabilistic graphical model that represents a set of variables and their conditional dependencies using a directed acyclic graph (DAG). ● Nodes → represent random variables. ● Edges → represent conditional dependencies. ● Conditional Probability Tables (CPTs) → define the	CO1	K5

	strength of dependencies. BNs allow reasoning under uncertainty, which is crucial in Artificial Intelligence. □ Applications of Bayesian Networks in AI 1. Medical Diagnosis and Healthcare ● Use Case: Diagnosing diseases based on symptoms and test results. ● Example: If a patient has cough and fever, a BN can compute the probability of diseases like flu, pneumonia, or COVID-19. ● Why Useful: Doctors get probabilistic decision support, not just yes/no answers. 2. Natural Language Processing (NLP) ● Use Case: Speech recognition, part-of-speech tagging, word prediction. ● Example: In speech recognition, BNs model the probability of words given sounds (phonemes). ● Why Useful: Helps resolve ambiguity when multiple words sound similar. 3. Robotics and Autonomous Systems ● Use Case: Decision-making under uncertainty in robots. ● Example: A self-driving car uses BNs to infer whether an object detected is a pedestrian, cyclist, or another car based on sensor data. ● Why Useful: Improves safety by reasoning about uncertain, noisy sensor inputs. 4. Fraud Detection and Security ● Use Case: Detecting fraudulent transactions in banking or cyberattacks. ● Example: A BN can calculate the probability of a		

	transaction being fraudulent based on factors like location, amount, time, and user history. ● Why Useful: It provides probabilistic alerts instead of rigid rule-based alerts. 5. Machine Learning & Data Mining ● Use Case: Feature selection, classification, and causal discovery. ● Example: In credit scoring, BNs can model dependencies between income, job stability, credit history, and loan repayment probability. ● Why Useful: Learns causal relations from data and avoids overfitting. 6. Fault Diagnosis in Engineering Systems ● Use Case: Detecting and predicting system failures. ● Example: In aircraft engines, BNs infer the probability of component failure based on vibration, temperature, and fuel efficiency readings. ● Why Useful: Supports predictive maintenance by identifying risks before failure. 7. Recommendation Systems ● Use Case: Personalized suggestions in e-commerce, streaming services. ● Example: A BN can infer a user's preference for a movie based on their past watch history, ratings, and similarity with other users. ● Why Useful: Provides more context-aware recommendations. 8. Genetics and Bioinformatics ● Use Case: Modeling gene interactions and predicting hereditary diseases. ● Example: BNs model the probability of inheriting a disease		

	gene given family history and genetic tests. ● Why Useful: Enables predictive and preventive healthcare.		
2.	Explain in detail about forward and backward chaining with appropriate example. Refer Part B question	CO1	K5

Unit – III Introduction to Machine Learning

Q.No	Questions	COs	Blooms Level
	Part A		
1.	Define Machine Learning. How does it differ from traditional programming? Machine Learning is a subfield of Artificial Intelligence (AI) that focuses on designing algorithms and systems that can learn patterns from data and improve their performance automatically over time without being explicitly programmed.	CO3	K1
2.	List any two types of Machine Learning and give one example for each  Type of Machine Learning | Explanation | Example Supervised Learning | The model is trained on labeled data (input + correct output). It learns the mapping between input and output. | Predicting house prices using past data (size, location → price). Unsupervised Learning | The model is trained on unlabeled data. It tries to find hidden patterns or groupings in the data. | Customer segmentation in marketing (grouping customers based on buying behavior).	CO3	K2
3.	What is the role of a training dataset in supervised learning? In supervised learning, the training dataset is a collection of labeled input-output pairs that teaches a model to recognize patterns and map inputs to their correct outputs.	CO3	K2
4.	List any two real-life applications of Machine Learning. Machine Learning in Healthcare	CO3	K1

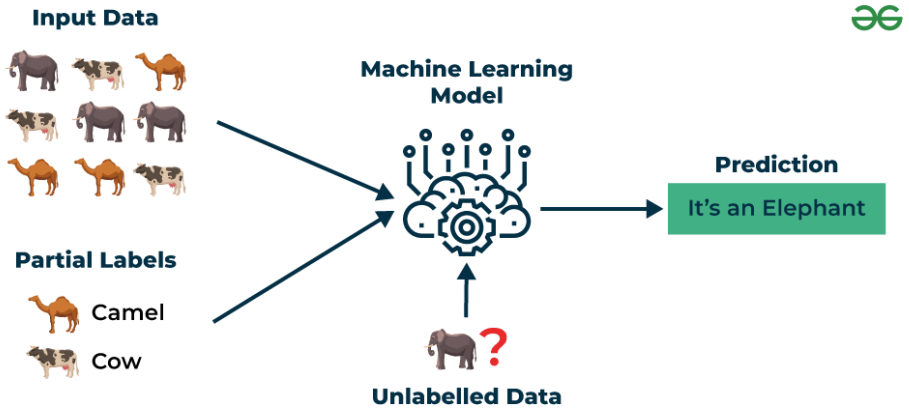
<table>
<tr><td></td><td>Machine Learning in Finance
Machine Learning in Marketing and Advertising
Machine Learning in Autonomous Vehicles
Machine Learning in Retail
Machine Learning in Energy</td><td></td><td></td></tr>
<tr><td>5.</td><td>Define supervised learning with an example
Supervised learning is a type of machine learning where an algorithm learns from a dataset that contains both input features and their corresponding correct output labels. This means the data is "labeled," acting as a guide for the algorithm to learn the relationship between the inputs and the desired outputs.</td><td>CO3</td><td>K1</td></tr>
<tr><td>6.</td><td>List any two algorithms used in supervised learning
Linear Regression
● Used for predicting a continuous value.
● Example: Predicting house prices based on size, number of rooms, and location.

Decision Tree
● Used for classification and regression tasks.
● Example: Classifying whether a patient has diabetes (Yes/No) based on medical test results.</td><td>CO3</td><td>K1</td></tr>
<tr><td>7.</td><td>Differentiate between supervised and unsupervised learning.
<table><tr><th>Feature</th><th>Supervised Learning</th><th>Unsupervised Learning</th></tr><tr><td>Definition</td><td>A type of learning where the model is trained using labeled data (input + correct output).</td><td>A type of learning where the model is trained using unlabeled data (only input, no correct output).</td></tr><tr><td>Goal</td><td>To predict outcomes for new inputs.</td><td>To find hidden patterns, structures, or groups in the data.</td></tr><tr><td>Data Requirement</td><td>Needs labeled data (expensive & time-consuming to prepare).</td><td>Works with raw/unlabeled data (easier to collect).</td></tr><tr><td>Output</td><td>Prediction of a class label (classification) or value (regression).</td><td>Formation of clusters or dimensionality reduction.</td></tr><tr><td>Examples</td><td>Predicting email as spam/not spam, predicting house prices.</td><td>Grouping customers with similar buying behavior, market segmentation.</td></tr></table></td><td>CO3</td><td>K2</td></tr>
</table>

8.	Explain the role of labeled data in supervised learning Labeled data is raw data that has been assigned labels to add context or meaning, which is used to train machine learning models in supervised learning.	CO3	K2

Q.No	Questions	COs	Blooms Level
	Part B		
1.	Give an overview of machine learning with suitable diagrams and also explain on the various types of machine learning with its advantages and disadvantages. ● Machine Learning (ML) is a branch of Artificial Intelligence (AI) where systems learn from data and improve performance without being explicitly programmed. Instead of writing step-by-step instructions, we feed data into algorithms that learn patterns and make predictions/decisions. Types of Machine Learning 1. Supervised Learning ● Definition: Model is trained with labeled data (input + output known). ● Example: Predicting house prices based on features like area, location, and number of rooms. ● Algorithms: Linear Regression, Decision Trees, Support Vector Machines (SVM). Advantages: ● High accuracy with enough labeled data. ● Easy to evaluate performance. Disadvantages: ● Requires large labeled datasets.	CO3	K3

- Labeling data is costly and time-consuming.



2. Unsupervised Learning

- Definition: Model learns patterns from unlabeled data (only inputs, no outputs).
- Example: Customer segmentation in marketing (grouping customers by behavior).
- Algorithms: K-Means Clustering, Hierarchical Clustering, PCA.

Advantages:

- Works well for exploratory analysis.
- Can discover hidden patterns.

Disadvantages:

- Hard to evaluate accuracy.
- Risk of meaningless clusters/patterns.

3. Reinforcement Learning (RL)

- **Definition**: An agent learns by interacting with an environment, receiving **rewards/penalties** for actions.
- **Example**: Self-driving cars, game-playing AI (e.g., AlphaGo).
- **Algorithms**: Q-Learning, Deep Q-Networks.

	✓Advantages: ● Useful for sequential decision-making. ● Can outperform humans in certain tasks. ✗Disadvantages: ● Requires a lot of training time. ● Risky in real-world environments.		
2.	Describe in detail how you would build a supervised learning model using this dataset. Include steps such as data preprocessing, algorithm selection, training, validation, and performance evaluation. Justify your choices at each step **1**. Data Preprocessing (a) Data Cleaning ● Handle Missing Values: If some attributes (like age, relation, etc.) are missing, decide whether to impute (mean/mode) or drop them. ● Remove Duplicates: Ensure no repeated entries. ● Convert Categorical to Numeric: For columns like *gender, ethnicity, relation*, use: ○ Label Encoding (for binary categorical features, e.g., Male/Female). ○ One-Hot Encoding (for multi-class categorical features like country_of_res). Justification: Machine learning algorithms work better with numerical values. (b) Feature Scaling ● Apply Min-Max Scaling or Standardization to numerical attributes (like age, A1–A10 scores). ● Example: Convert age into a standard range (0–1).	CO3	K4

Justification: Prevents bias toward features with larger ranges.

(c) Splitting Data

- Train-Test Split (80-20 or 70-30):
    - Training data → to fit the model.
    - Testing data → to evaluate generalization performance.
- Optionally, use Validation Set (or k-fold Cross-Validation).

Justification: Prevents overfitting and ensures unbiased evaluation.

2. Algorithm Selection

Since it's a supervised learning (classification) task, we can choose algorithms such as:

- Logistic Regression → simple, interpretable, works well for linearly separable data.
- Random Forest → handles categorical + numerical data, good for nonlinear patterns, reduces overfitting.
- Support Vector Machine (SVM) → effective in high-dimensional spaces.
- Neural Network (MLP) → for complex relationships if dataset is large.

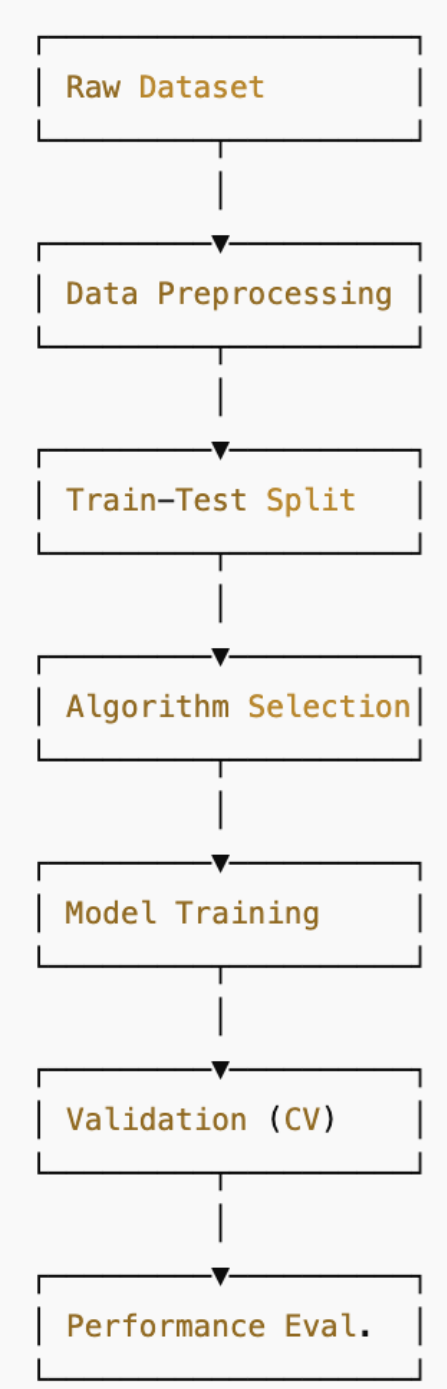
Choice Example:

- Start with Logistic Regression for baseline.
- Then try Random Forest for improved performance.

3. Model Training

- Feed the training set into the selected algorithm.
- Tune hyperparameters using Grid Search / Random Search:

	○ Logistic Regression → regularization (C, penalty). ○ Random Forest → number of trees, max depth. Justification: Hyperparameter tuning improves accuracy and avoids under/overfitting. 4. Validation ● Use k-Fold Cross-Validation (e.g., k=5 or 10). ● The dataset is split into *k parts*, train on (k-1), test on 1, repeat. Justification: Provides a robust estimate of performance, not just on a single train-test split. 5. Performance Evaluation Use the test set with classification metrics: ● Accuracy: Overall correctness. ● Precision & Recall: Important if classes are imbalanced (e.g., ASD vs Non-ASD). ● F1-Score: Harmonic mean of precision and recall. ● Confusion Matrix: To see true positives/negatives vs false predictions. ● ROC-AUC Curve: For probability-based classifiers. Justification: Accuracy alone may be misleading if dataset is imbalanced.		

			
3.	Supervised learning in machine learning is broadly classified into classification and regression techniques. Explain both types in detail with suitable examples, use-cases, and differences in approach. Also, discuss how the choice between classification and regression depends on the nature of the problem and the type of output variable. Supervised learning is a type of machine learning where a model is trained using labeled data (input + output pairs). The goal is to learn a	CO3	K4

mapping from inputs (features) to outputs (target labels/values).

Supervised learning is broadly classified into two techniques:

1 Classification

 Definition

Classification is the task of predicting a discrete categorical label (output belongs to one of a set of predefined classes).

The model learns decision boundaries that separate different classes.

 Example

- Predicting if an email is spam or not spam.
- Diagnosing a patient as diabetic or non-diabetic based on medical records.

 Use-Cases

- Healthcare: Disease diagnosis (cancer detection: malignant/benign).
- Finance: Fraud detection (fraud / no fraud).
- Marketing: Customer churn prediction (will leave / will not leave).
- Computer Vision: Handwritten digit recognition (0–9).

 Algorithms Commonly Used

- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- Neural Networks

2 Regression

 Definition

	Regression is the task of predicting a continuous numerical value. The model fits a curve or line that best approximates the relationship between input variables and a continuous output. □ Example ● Predicting the price of a house based on area, location, and number of rooms. ● Predicting temperature for tomorrow based on weather conditions. □ Use-Cases ● Economics: Predicting GDP growth. ● Healthcare: Predicting blood sugar level. ● Business: Predicting sales revenue. ● Engineering: Predicting energy consumption. □ Algorithms Commonly Used ● Linear Regression ● Polynomial Regression ● Support Vector Regression (SVR) ● Random Forest Regression ● Neural Networks		
4.	You are tasked with developing a predictive model for a complex real-world dataset. Explain how you would detect and diagnose overfitting and underfitting during model development. Critically assess the trade-offs involved in addressing these issues, and propose a strategy that balances model bias and variance effectively	CO3	K5

1. Overfitting

- Definition: Model learns training data too well, including noise, so it performs poorly on unseen data.
- Detection:
    - High training accuracy, low validation/test accuracy.
    - Validation loss starts increasing while training loss continues to decrease.
    - Large gap between training and validation performance.

Example:
A decision tree with depth = 50 that memorizes all training points but fails on new samples.

2. Underfitting

- Definition: Model is too simple, fails to capture data patterns, resulting in poor performance on both training and validation sets.
- Detection:
    - Low training accuracy and low validation accuracy.
    - Both training and validation losses are high and decrease very slowly.
    - Predictions look almost random (model too rigid).

Example:
Using linear regression to model a nonlinear relationship like housing prices vs. features.

Bias-Variance Trade-off

- Bias (Underfitting): Model assumptions too strong → can't capture complexity → high training error.
- Variance (Overfitting): Model too flexible → memorizes noise → poor generalization.
- Goal: Find the sweet spot where both training and validation

errors are low and close.

Diagnosing and Addressing Issues

1. If Overfitting is Detected

- Reduce variance:
    - Use regularization (L1, L2, dropout).
    - Reduce model complexity (shallow trees, fewer layers).
    - Increase training data (data augmentation or collection).
    - Apply early stopping during training.
    - Use cross-validation to monitor generalization.

2. If Underfitting is Detected

- Reduce bias:
    - Use a more complex model (e.g., move from linear regression to polynomial regression or neural networks).
    - Add more relevant features (feature engineering).
    - Reduce regularization if it's too strong.
    - Increase training time (more epochs).

Balanced Strategy to Manage Bias & Variance

1. Start simple: Begin with a baseline model (e.g., linear/logistic regression, decision tree) to understand data patterns.
2. Cross-validation: Use k-fold CV to measure generalization and tune hyperparameters.
3. Learning curves: Plot training vs. validation error:
    - If both are high → underfitting.
    - If there's a large gap → overfitting.
4. Regularization tuning: Adjust L1/L2 penalties, dropout,

	pruning, etc., to balance bias and variance. 5. Ensemble methods: Use Random Forest, Gradient Boosting, or Bagging to reduce variance without huge bias. 6. Final evaluation: Use test set (never touched before) to estimate true generalization performance.		

Q.No	Questions	COs	Blooms Level
	Part C		
1.	Given a dataset of customer purchasing behaviour, how would a machine learning algorithm learn from this data to predict future purchases Suppose you have a dataset that contains information like: ● Customer ID ● Age, Gender, Income, Location (demographic features) ● Purchase history (past transactions, product categories, frequency, amount spent) ● Time spent on website / app usage data ● Promotions/campaigns responded to ● Target label: *whether the customer purchased a product or not* (for classification) OR *how much they will spend in the next month* (for regression). --- Steps in Building the Predictive Model 1. Data Preprocessing ● Cleaning: Handle missing values (e.g., income not provided). ● Encoding: Convert categorical data (e.g., Gender = Male/Female → 0/1).	CO3	K6

- Scaling: Normalize continuous values (e.g., income, age) so the algorithm treats them equally.
- Feature Engineering: Create meaningful features such as "average monthly spend," "loyalty score," or "days since last purchase."

2. Algorithm Selection

- If predicting whether a customer will purchase → Classification
  Examples: Logistic Regression, Decision Trees, Random Forest, Gradient Boosting, or Neural Networks.
- If predicting how much a customer will spend → Regression
  Examples: Linear Regression, Random Forest Regressor, Gradient Boosting Regressor, etc.

3. Training the Model

- Split dataset into Training (70%) and Test (30%).
- Train the model on training data by finding patterns (e.g., frequent buyers may have high income, respond to promotions, etc.).
- The algorithm adjusts its parameters/weights to minimize error (e.g., misclassified purchases).

4. Validation

- Use Cross-validation (k-fold) to ensure the model generalizes well.
- Use a Validation Set (separate from training) to tune hyperparameters like learning rate, tree depth, number of hidden layers, etc.

5. Performance Evaluation

- For Classification (purchase or not):
  - Accuracy, Precision, Recall, F1-score, ROC-AUC.
- For Regression (spending amount):

	○ Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R^2 Score.		
2.	You are provided with a housing dataset containing features such as house size, number of bedrooms, age, location, and sale prices. How would you apply regression techniques in machine learning to build a model that predicts house prices accurately? Describe the steps you would take from data preprocessing to model evaluation." Step 1: Understanding the Problem ● Goal: Predict continuous output (house price). ● Suitable technique: Regression (Supervised Learning). ● Example: "Given house features, estimate its selling price." Step 2: Data Preprocessing 1. Data Cleaning ○ Handle missing values (e.g., impute median for missing age or bedrooms). ○ Remove duplicates/outliers (e.g., houses with price = 0). 2. Feature Encoding ○ Convert categorical variables like *location* into numerical format. ■ One-hot encoding for nominal categories (urban, rural, suburban). ■ Label encoding if ordinal. 3. Feature Scaling ○ Apply normalization/standardization (important for models like Linear Regression, Ridge, Lasso). ○ Example: Scale *house size* from square feet to a normalized range.	CO3	K6

4. Train-Test Split
    - Split dataset (e.g., 80% training, 20% testing).
    - Prevents data leakage.

Step 3: Feature Engineering

- Derive new useful features:
    - Price per square foot = Sale price / Size.
    - Age groups (new, mid-aged, old).
- Handle multicollinearity (highly correlated features → may cause instability in linear regression).

Step 4: Algorithm Selection

You can try multiple regression techniques:

1. Linear Regression
    - Simple baseline model.
    - Easy to interpret but may not handle non-linear relationships.
2. Regularized Regression (Ridge, Lasso, ElasticNet)
    - Prevents overfitting.
    - Good for datasets with many features.
3. Decision Tree / Random Forest Regression
    - Captures non-linear relations.
    - Robust to missing values and outliers.
4. Gradient Boosting (XGBoost, LightGBM, CatBoost)
    - Advanced, high-performance models.
    - Often used in real-world Kaggle competitions.

Strategy: Start with Linear Regression → Evaluate → Try tree-based

models for better accuracy.

Step 5: Training

- Fit the chosen regression model using the training dataset.
- Example: In scikit-learn → `LinearRegression().fit(X_train, y_train)`.

Step 6: Validation

- Use cross-validation (e.g., k-fold CV) to check performance consistency.
- Helps avoid overfitting.

Step 7: Performance Evaluation

Common metrics for regression:

- Mean Absolute Error (MAE) → Average error in predictions.
- Mean Squared Error (MSE) / Root MSE (RMSE) → Penalizes large errors.
- R^2 (Coefficient of Determination) → How much variance in price is explained by features.

Example:

- MAE = $10,000 → On average, predictions are off by $10,000.
- R^2 = 0.87 → Model explains 87% of price variability.

Step 8: Model Improvement

- Hyperparameter tuning (e.g., max depth for trees, alpha for Ridge/Lasso).
- Feature selection to remove irrelevant features.
- Ensemble methods (Random Forest + Gradient Boosting).