INTRODUCTION TO ARTIFICIAL INTELLIGENCE

QUESTION BANK ANSWERS

UNIT-1

1. a. What are the types of Agents?

Answer:-Agents can be grouped into four classes based on their degree of perceived intelligence and capability :

- ✓ Simple Reflex Agents
- ✓ Model-Based Reflex Agents
- ✓ Goal-Based Agents
- ✓ Utility-Based Agents

❖ What are the types of Intelligent System?

- ✓ Systems that think like humans
- ✓ Systems that think rationally
- ✓ Systems that behave like humans
- ✓ Systems that behave rationally

b. What is the space complexity of BFS Algorithm?

Answer:-Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.Where the d= depth of shallowest solution and b is branch factor.

c. Write down the properties name of Search Algorithm.

Answer:-Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

- ✓ **Completeness**
- ✓ **Cost Optimality**
- ✓ **Time complexity**
- ✓ **Space complexity**

d. What is the time complexity of DFS algorithm?

Answer:-Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$

Where, m= maximum depth of any node and n is maximum number of child node of any node (branch factor).

e. Define problem solving agent.

Answer:-Problem Solving Agents decide what to do by finding a sequence of actions that leads to a desirable state or solution.An agent may need to plan when the best course of action is not immediately visible. They may need to think through a series of moves that will lead them to their goal state. Such an agent is known as a **problemsolving agent.**

2. a. Explain Iterative deepening depth-first Search.

Answer:- The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

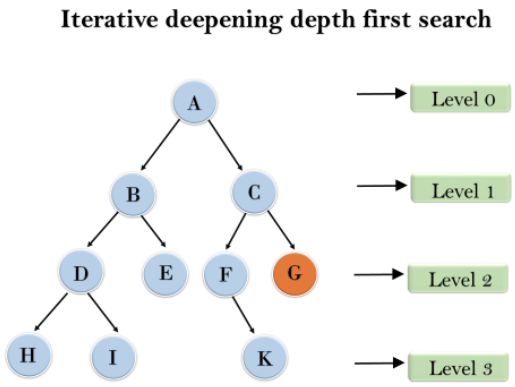
The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Here are the steps for Iterative deepening depth first search algorithm:

- Set the depth limit to 0.
- Perform DFS to the depth limit.
- If the goal state is found, return it.
- If the goal state is not found and the maximum depth has not been reached, increment the depth limit and repeat steps 2-4.
- If the goal state is not found and the maximum depth has been reached, terminate the search and return failure.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

b. Explain about different Environment Types in AI.

Answer:-

1. Accessible vs. inaccessible or Fully observable vs Partially Observable:

If an agent sensor can sense or access the complete state of an environment at each point of time then it is a fully observable environment, else it is partially observable.

2. Deterministic vs. Stochastic:

If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic

3. Episodic vs. non-episodic:

The agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the

episode itself, because subsequent episodes do not depend on what actions occur in previous episodes.

Episodic environments are much simpler because the agent does not need to think ahead.

4. Static vs. dynamic.

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static.

5. Discrete vs. continuous:

If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Otherwise, it is continuous.

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Figure 2.6 Examples of task environments and their characteristics.

c. Name the Search Algorithm Terminologies and explain them.

Answer:-Different search algorithm terminologies in AI are-

- **Search:** Searchingis a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - Search Space: Search space represents a set of possible solutions, which a system may have.
  - Start State: It is a state from where agent begins the search.
  - Goal test: It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

d. Explain BFS with suitable example.

Answer:-

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

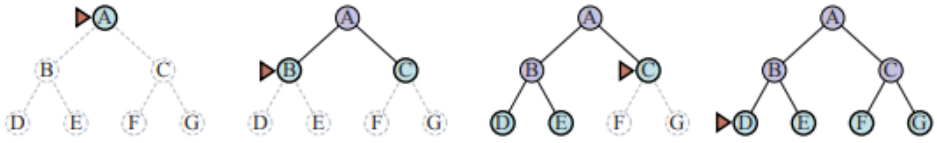


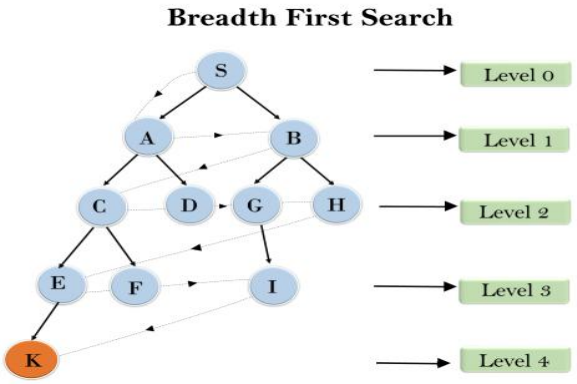
Figure 3.8 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

- Breadth-first search implemented using FIFO queue data structure.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.
- It also helps in finding the shortest path in goal state, since it needs all nodes at the same hierarchical level before making a move to nodes at lower levels.
- It is also very easy to comprehend with the help of this we can assign the higher rank among path types.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

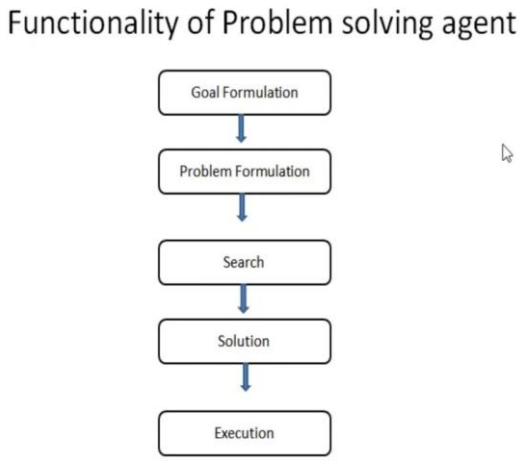


e. Explain Problem Solving Agents in brief.

Answer:- Problem Solving Agents decide what to do by finding a sequence of actions that leads to a desirable state or solution.
An agent may need to plan when the best course of action is not immediately visible. They may need to think through a series of moves that will lead them to their goal state. Such an agent is known as a **problem solving agent**, and the computation it does is known as a **search**.

The problem solving agent follows this four phase problem solving process:

1. **Goal Formulation:** This is the first and most basic phase in problem solving. It arranges specific steps to establish a target/goal that demands some activity to reach it. AI agents are now used to formulate goals.
2. **Problem Formulation:** It is one of the fundamental steps in problem-solving that determines what action should be taken to reach the goal.
3. **Search:** After the Goal and Problem Formulation, the agent simulates sequences of actions and has to look for a sequence of actions that reaches the goal. This process is called **search,** and the sequence is called a **solution**. The agent might have to simulate multiple sequences that do not reach the goal, but eventually, it will find a solution, or it will find that no solution is possible. A search algorithm takes a problem as input and outputs a sequence of actions.
4. **Execution:** After the search phase, the agent can now execute the actions that are recommended by the search algorithm, one at a time. This final stage is known as the execution phase.



Problem Formulation: A formal definition of a problem consists of five components:

1. Initial State
2. Actions
3. Transition Model
4. Goal Test
5. Path Cost

1. Initial State

It is the agent's starting state or initial step towards its goal. For example, if a taxi agent needs to travel to a location(B), but the taxi is already at location(A), the problem's initial state would be the location (A).

2. Actions

It is a description of the possible actions that the agent can take. Given a state s, **Actions**(**s**) returns the actions that can be executed in s. Each of these actions is said to be appropriate in s.

3. Transition Model

It describes what each action does. It is specified by a function **Result**(**s, a**) that returns the state that results from doing action an in state s.
The initial state, actions, and transition model together define the **state space** of a problem, a set of all states reachable from the initial state by any sequence of actions. The state space forms a graph in which the nodes are states, and the links between the nodes are actions.

4. Goal Test

It determines if the given state is a goal state. Sometimes there is an explicit list of potential goal states, and the test merely verifies whether the provided state

is one of them. The goal is sometimes expressed via an abstract attribute rather than an explicitly enumerated set of conditions.

5. **Path Cost**

It assigns a numerical cost to each path that leads to the goal. The problem solving agents choose a cost function that matches its performance measure. Remember that the optimal solution has the lowest path cost of all the solutions.

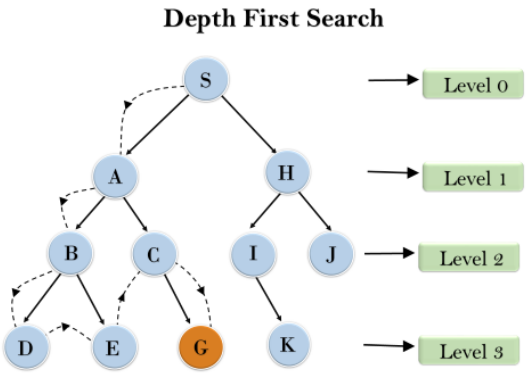
3. a. Explain DFS with suitable example.

Answer:-

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure. It generally starts by exploring the deepest node in the frontier. Starting at the root node, the algorithm proceeds to search to the deepest level of the search tree until nodes with no successors are reached. Suppose the node with unexpanded successors is encountered then the search backtracks to the next deepest node to explore alternative paths.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.DFS uses a stack data structure (LIFO) for its implementation.
- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Example:

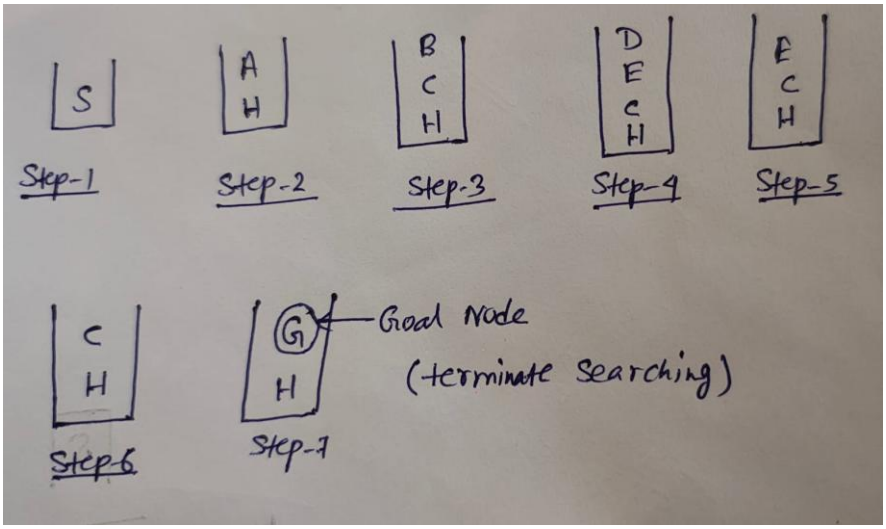
- In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:
- Root node--->Left node ----> right node.
- It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Order of Visited Nodes:

S→A→B→D→E→C →G

Visual Representation:



b. Explain about A* search algorithm.

Answer:- A* Search algorithm is one of the best and popular informed search technique used in path-finding and graph traversals.

In the A* algorithm, we consider both path cost and heuristics. In A* the f(n) function comprises two components: path cost [g(n)] and heuristic value ([h(n)]. The f(n) value for node ‘a’ can be calculated as follows:

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

$f(n)_a$ = Evaluation value at the particular node (node 'a')
$g(n)_a$ = Total path cost from start node to particular node(node 'a')
$h(n)_a$ = The heuristic value of the particular node(node 'a')

Now, we will find the best path according to the A* search algorithm for our previous problem. We have to find the f(n) value for each node and select the child node with the least f(n) value and traverse until meeting the goal node. In the example, only the f(n) values of goals in the path are mentioned.

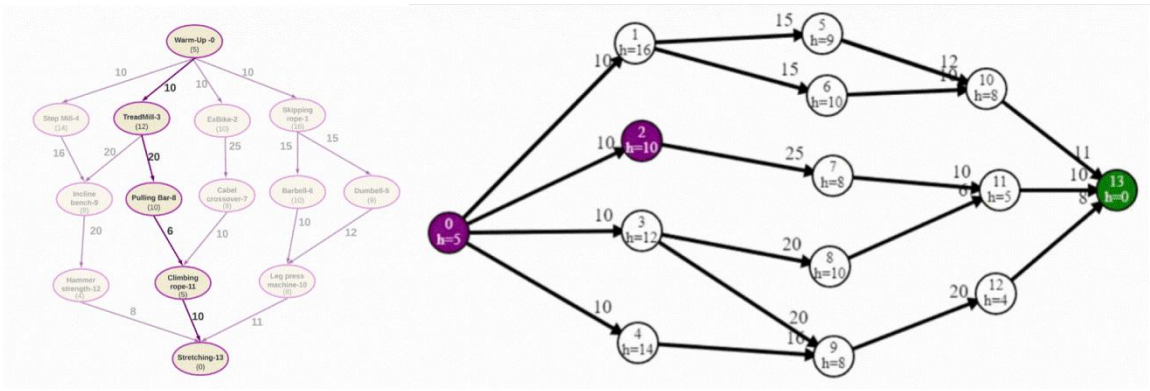


Figure 13: State-Space (left) and state-space traversal (right) in A* search

At Node-0 :

$f(n)_0 = 0 + 5 = 5$

At Node-3:

$f(n)_3 = 10 + 12 = 22$

At Node-8:

$f(n)_8 = 30 + 10 = 40$

At Node-11:

$f(n)_{11} = 36 + 5 = 41$

At Node-13:

$f(n)_{13} = 46 + 0 = 46$

$f(n)_{total} = f(n)_0 + f(n)_3 + f(n)_8 + f(n)_{11} + f(n)_{13} = 154$

Path: 0, 3, 8, 11, 13

A* algorithm is complete since it checks all the nodes before reaching to goal node/end of the state space. It is optimal as well because considering both path cost and heuristic values, therefore the lowest path with the lowest heuristics can be found with A*. One drawback of A* is it stores all the nodes it processes in the memory. Therefore, for a state space of branching factor 'b', and the depth 'd' the space and time complexities of A* are denoted by $\mathbf{O(b^d)}$. The time complexity of A* depends on the heuristic values.

c. What are the types of AI agent? Explain them with neat diagram.

Answer:- An agent can be anything that perceive its environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of perceiving, thinking, and acting.

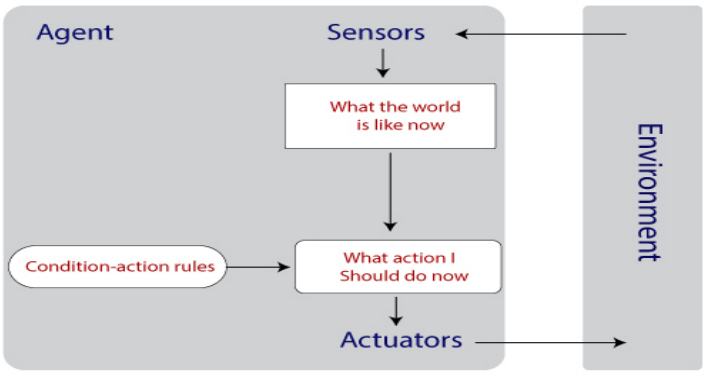
Types of agents:

Agents can be grouped into four classes based on their degree of perceived intelligence and capability :

- ✓ Simple Reflex Agents
- ✓ Model-Based Reflex Agents
- ✓ Goal-Based Agents
- ✓ Utility-Based Agents

Simple reflex agents:

- Simple reflex agents ignore the rest of the percept history and act only on thebasis ofthe current percept.
- The agent function is based on the condition-action rule.
- If the condition is true, then the action is taken, else not. This agent function only succeedswhenthe environment is fully observable.

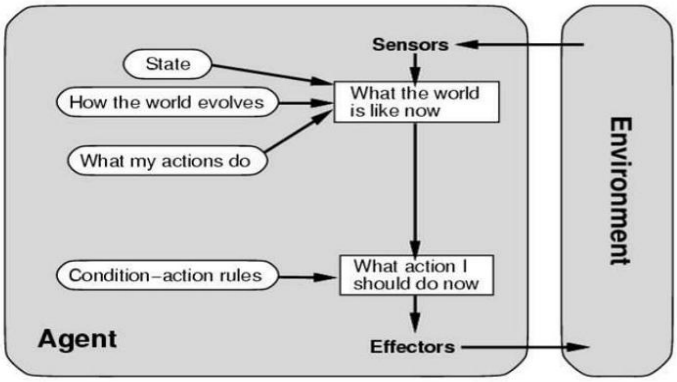


Schematic diagram of a simple-reflex agent

Model-based reflex agents:

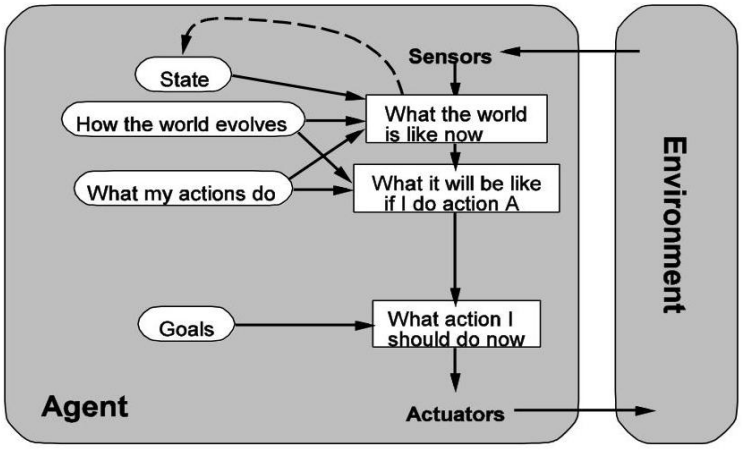
- The Model-based agent can work in a partially observable environment, and track thesituation.
- A model-based agent has two important factors:
- Model: It is knowledge about "how things happen in the world," so it is called a Model-based agent.

- Internal State: It is a representation of the current state based on percept history.



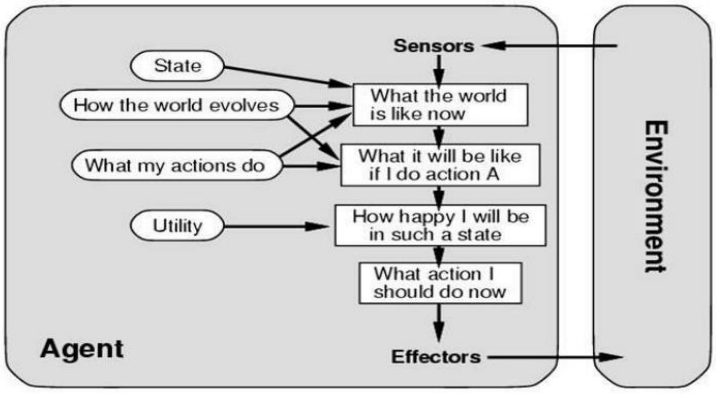
Goal-based agents:

- A goal-based agent has an agenda.
- It operates based on a goal in front of it and makes decisions based on how best to reach that goal.
- A goal-based agent operates as a search and planning function, meaning it targets the goalahead andfinds the right action in order to reach it.
- Expansion of model-based agent.



Utility-based agents:

- utility-based agent is an agent that acts based not only on what the goal is, but the best way toreach thatgoal.
- The Utility-based agent is useful when there are multiple possible alternatives, and anagent has tochoose in order to perform the best action.
- The term utility can be used to describe how "happy" the agent is.



d. Write a short note on State Space Algorithm.

Answer:- The state space representation forms the basis of most of the AI methods.

State-space consists of all the possible states together with actions to solve a specific problem. In the graphical representation of state space (such as trees), the states are represented by nodes while the actions are represented by arcs. Any state space has an initial 'Start' node and an ending 'Goal' node or multiple Goal States. The path from the initial start node to the final goal node is known as the 'Solution' for the particular problem.

Formal Description of the problem:

1. Define a state space that contains all the possible configurations of the relevant objects.

2. Specify one or more states within that space that describe possible situations fromwhichthe problem solving process may start (initial state)

3. Specify one or more states that would be acceptable as solutions to the problem. (goal states)Specify a set of rules that describe the actions (operations) available.

For a clear understanding of state space, consider the following problem [Figure 2].

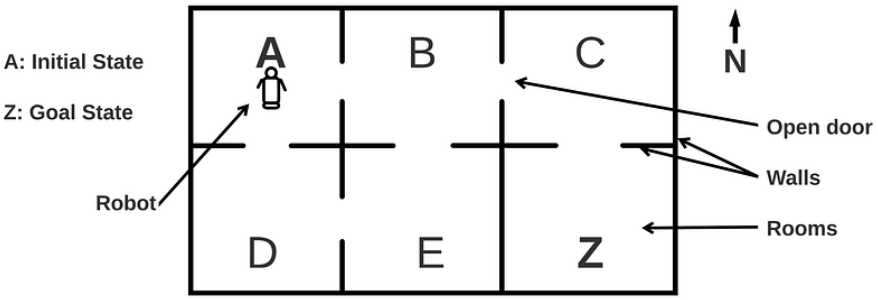


Figure 2: Example Problem

Imagine there is a robot in room ‘A’ (initial state), and it needs to go to room ‘Z’ (goal state). We can draw a state space in terms of a tree if we consider all the possible movements of the robot in each room (node). For example, when the robot is at initial state A, he can either go to B or D. When the robot moved to the next state B, he can move to C, E, or back to A [Figure 3].

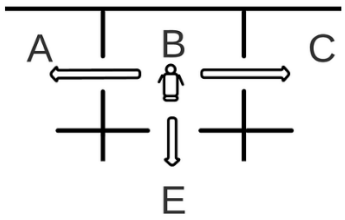


Figure 3: Possible paths for robot at state B

Based on all the possible movements of the robot at each state we can draw the state space for the above scenario as follows:

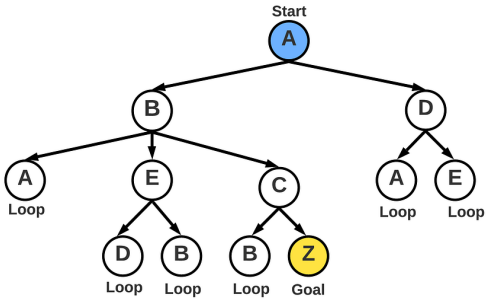


Figure 4: State-space diagram

We can identify many paths beside the direct path A, B, C, Z.

Ex: A B C Z
A B A B C Z
A D E B C Z
A D E B A B C Z
....

It can be observed that some paths are shorter while others are longer. The real problem arises here. We can figure out the best/shortest path in cases like the above example where the state space is small. But imagine a state-space with hundreds of thousands of nodes. How can we explore such a state-space?

The solution is nothing but search algorithms.

Since the state spaces are very large in general, we employ search algorithms to navigate through the state-space effectively and efficiently. A search algorithm defines how to find the path (solution) from the initial state to the goal state. Different algorithms define different methods to move from the current state (node) to the next state (node). Some algorithms provide just the systematic ways to explore the state space while others tell how to explore effectively and efficiently.

e. Write about Greedy Search Algorithm.

Answer:- In greedy search, the heuristic values of child nodes are considered. The path is determined by calculating the path with the nodes with the lowest heuristic values. Another fact to be noticed is that usually the initial node has the highest heuristic value and the goal node has the lowest. But there can be exceptions like getting a mid-range value for the initial node also.

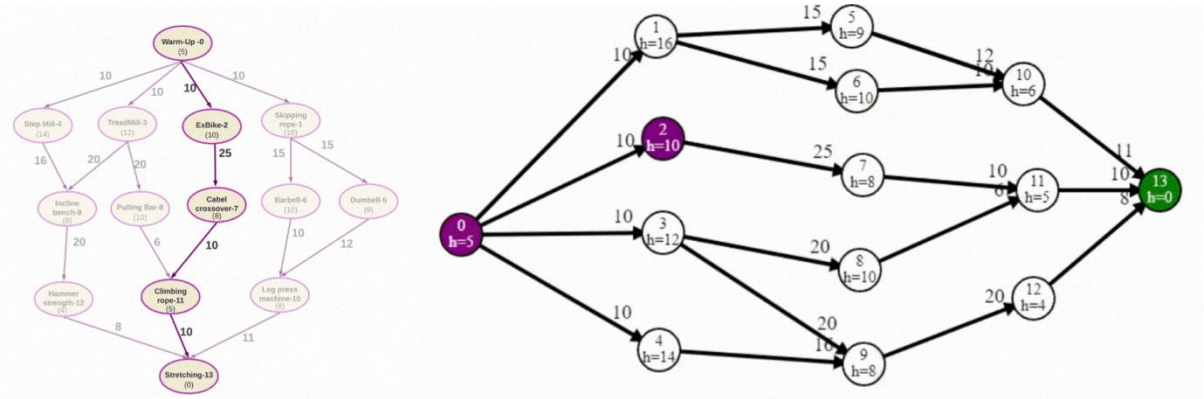


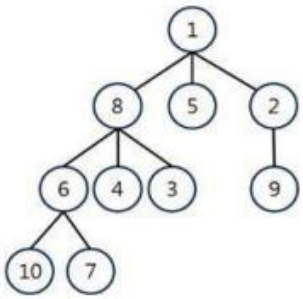
Figure 12: State-Space (left) and state-space traversal (right) in greedy search

f(n) = h(n)
= Summation(Heuristic values of nodes)
= Node0 + Node2 + Node7 + Node11 + Node13
= 5 + 10 + 8 + 5 + 0
= 28
Path: 0, 2, 7, 11, 13

In greedy search, you can see we do not visit all the child nodes of a particular node. We find heuristics of each child node only the one with the lowest value is inserted to the OPEN list to be processed. Therefore greedy search is not complete. As well as this is not the best path (not the shortest path - we found this path in uniform cost search). Therefore greedy search is not optimal, also. As a solution, we should consider not only the heuristic value but also the path cost. Here, comes the A* algorithm.

4. a. Discuss the following search Technique with the help of a given tree. Also discuss the benefits and Application of each.

i. Breadth First Search
ii. Depth First Search



Answer:-

i. Breadth First Search

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.

- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

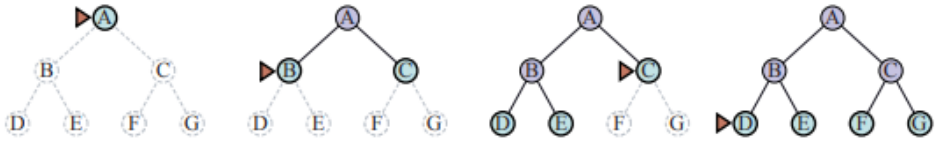
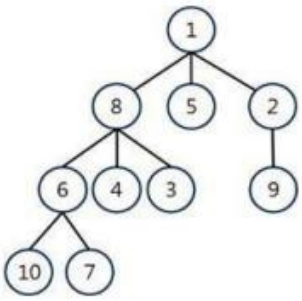


Figure 3.8 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

- Breadth-first search implemented using FIFO queue data structure.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.
- It also helps in finding the shortest path in goal state, since it needs all nodes at the same hierarchical level before making a move to nodes at lower levels.
- It is also very easy to comprehend with the help of this we can assign the higher rank among path types.

Example:



In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node 1to the bottom of the tree. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1---> 8--->5---->2--->6---->4--->3--->9---->10---->7

Advantages of BFS:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

- It also helps in finding the shortest path in goal state, since it needs all nodes at the same hierarchical level before making a move to nodes at lower levels.
- It is also very easy to comprehend with the help of this we can assign the higher rank among path types.

Applications Of Breadth-First Search Algorithm

- ✓ **GPS Navigation systems:** Breadth-First Search is one of the best algorithms used to findNeighbouring locations by using the GPS system.
- ✓ **Broadcasting:** Networking makes use of what we call as packets for communication. Thesepackets follow a traversal method to reach various networking nodes. One of the most commonly used traversal methods is Breadth-First Search. It is being used as an algorithm that is used to communicatebroadcasted packets across all the nodes in a network.

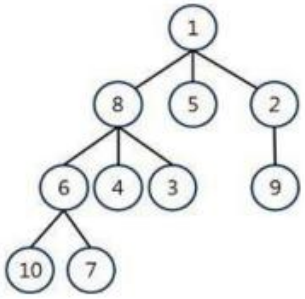
ii. Depth First Search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure. It generally starts by exploring the deepest node in the frontier. Starting at the root node, the algorithm proceeds to search to the deepest level of the search tree until nodes with no successors are reached. Suppose the node with unexpanded successors is encountered then the search backtracks to the next deepest node to explore alternative paths.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.DFS uses a stack data structure (LIFO) for its implementation.
- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Example:

- In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:
- Root node--->Left node ----> right node.
- Let’s consider 9 is the goal node.
- It will start searching from root node 1, and traverse 8, then 6, then 10 and 7, after traversing 7, it will backtrack to 6 then 8 the tree as 7 has no other successor.

- After backtracking it will traverse node 4 and then 3, after traversing 3, it will backtrack to 8 then 1 the tree as 3 has no other successor.
- After backtracking it will traverse node 2 and then 9 and here it will terminate as it found goal node.



Order of Visited Nodes:

$1 \rightarrow 8 \rightarrow 6 \rightarrow 10 \rightarrow 7 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 9$

Advantage of DFS:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).
- With the help of this we can stores the route which is being tracked in memory to save time as it only needs to keep one at a particular time.

Applications Of Depth-First Search Algorithm

- ✓ **Finding Strongly Connected Components of a graph:** A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.
- ✓ **Web crawlers:** Depth-first search can be used in the implementation of web crawlers to explore the links on a website.
- ✓ **Model checking:** Depth-first search can be used in model checking, which is the process of checking that a model of a system meets a certain set of properties.

b. Define heuristic search? Explain different types of heuristic search with suitable examples.

Answer:- A heuristic search/informed search is a strategy used in AI to optimize the search process by using a heuristic function to estimate the cost of

reaching a goal. Instead of exhaustively exploring all possible paths, a heuristic search uses this estimate to prioritize the most promising paths, reducing computational complexity and speeding up decision-making.

In informed search algorithms additional information is used to make the search more efficient and effective. That additional information iscalled **heuristics**. Heuristics are not theories but some common sense experience like information.

In informed search algorithms, to find the best node to be visited next, we use an *evaluation function* **f(n)** which assists the child node to decide on which node to be visited next. Then we traverse to the next node with the **least f(n)** value. Depending on the f(n), we have two informed search algorithms as greedy search and A* search algorithms.

2.1 Greedy Search Algorithms

In greedy search, the heuristic values of child nodes are considered. The path is determined by calculating the path with the nodes with the lowest heuristic values. Another fact to be noticed is that usually the initial node has the highest heuristic value and the goal node has the lowest. But there can be exceptions like getting a mid-range value for the initial node also.

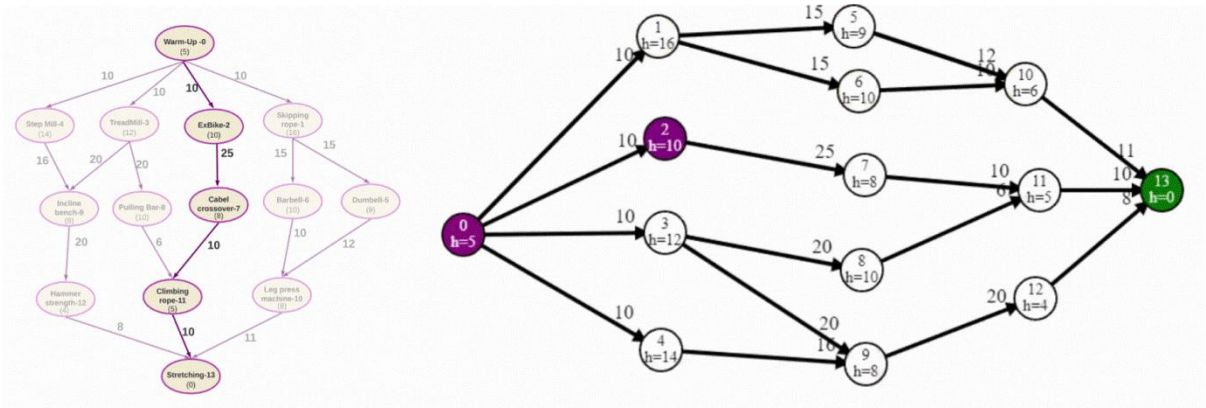


Figure 12: State-Space (left) and state-space traversal (right) in greedy search

f(n) = h(n)
= Summation(Heuristic values of nodes)
= Node0 + Node2 + Node7 + Node11 + Node13
= 5 + 10 + 8 + 5 + 0
= 28
Path: 0, 2, 7, 11, 13

In greedy search, you can see we do not visit all the child nodes of a particular node. We find heuristics of each child node only the one with the lowest value is inserted to the OPEN list to be processed. Therefore greedy search is not complete. As well as this is not the best path (not the shortest path - we found this path in uniform cost search). Therefore greedy search is not optimal, also. As a solution, we should consider not only the heuristic value but also the path cost. Here, comes the A* algorithm.

2.2 A* Search Algorithms

In the A* algorithm, we consider both path cost and heuristics. In A* the f(n) function comprises two components: path cost [g(n)] and heuristic value ([h(n)]. The f(n) value for node ‘a’ can be calculated as follows:

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

$f(n)_a$ = Evaluation value at the particular node (node 'a')
$g(n)_a$ = Total path cost from start node to particular node(node 'a')
$h(n)_a$ = The heuristic value of the particular node(node 'a')

Now, we will find the best path according to the A* search algorithm for our previous problem. We have to find the f(n) value for each node and select the child node with the least f(n) value and traverse until meeting the goal node. In the example, only the f(n) values of goals in the path are mentioned.

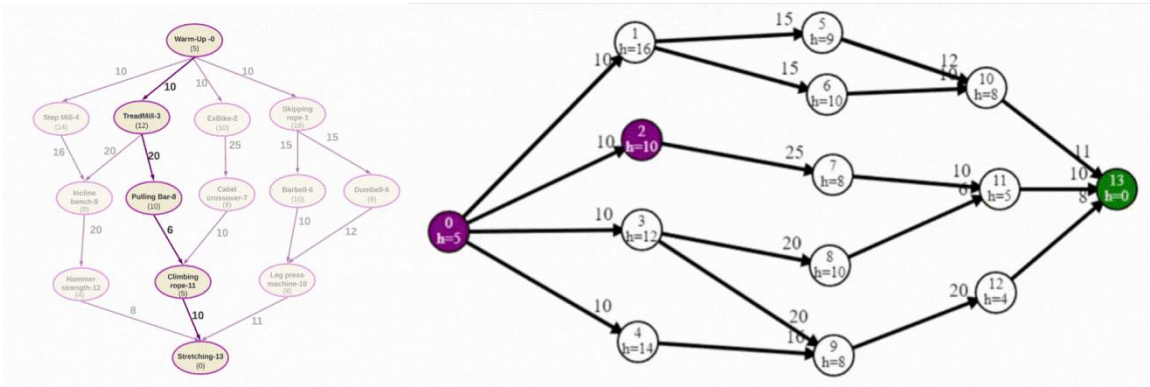


Figure 13: State-Space (left) and state-space traversal (right) in A* search

At Node-0 :

$f(n)_o = 0 + 5 = 5$

At Node-3:

$f(n)_3 = 10 + 12 = 22$

At Node-8:

$f(n)_8 = 30 + 10 = 40$

At Node-11:

$f(n)_{11} = 36 + 5 = 41$

At Node-13:

$f(n)_{13} = 46 + 0 = 46$

$f(n)_{total} = f(n)_0 + f(n)_3 + f(n)_8 + f(n)_{11} + f(n)_{13} = 154$

Path: 0, 3, 8, 11, 13

A* algorithm is complete since it checks all the nodes before reaching to goal node/end of the state space. It is optimal as well because considering both path cost and heuristic values, therefore the lowest path with the lowest heuristics can be found with A*. One drawback of A* is it stores all the nodes it processes in the memory. Therefore, for a state space of branching factor 'b', and the depth 'd' the space and time complexities of A* are denoted by **O(b^d)**. The time complexity of A* depends on the heuristic values.

c. Briefly explain Hill Climbing search and Simulated annealing algorithm with suitable example.

Answer:-

Local search algorithms are essential tools in artificial intelligence and optimization, employed to find high-quality solutions in large and complex problem spaces. Key algorithms include Hill-Climbing Search, Simulated Annealing, Local Beam Search, Genetic Algorithms, and Tabu Search.

Each of these methods offers unique strategies and advantages for solving optimization problems.

1. Hill Climbing Search

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbour has a higher value.

- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance travelled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbour state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

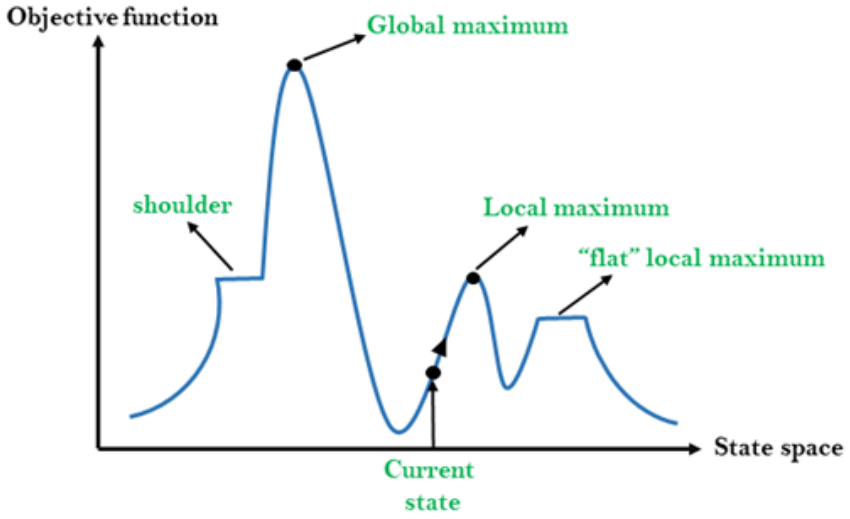
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.
- **Deterministic Nature:**
  Hill Climbing is a deterministic optimization algorithm, which means that given the same initial conditions and the same problem, it will always produce the same result. There is no randomness or uncertainty in its operation.
- **Local Neighbourhood:**
  Hill Climbing is a technique that operates within a small area around the current solution. It explores solutions that are closely related to the current state by making small, gradual changes. This approach allows it to find a solution that is better than the current one although it may not be the global optimum.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbour states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Example:

To illustrate hill climbing, we will use the 8-queens problem (Figure 4.3). We will use a complete-state formulation, which means that every state has all the components of a solution, but they might not all be in the right place. In this case every state has 8 queenson the board, one per column. The initial state is chosen at random, and the successors of a state are all possible states generated by moving a single queen to another square in the same column (so

each state has 8×7=56 successors). The heuristic cost function h is the number of pairs of queens that are attacking each other; this will be zero only for solutions. (It counts as an attack if two pieces are in the same line, even if there is an intervening piece between them.) Figure 4.3(b) shows a state that has h=17. The figure also shows the h values of all its successors.

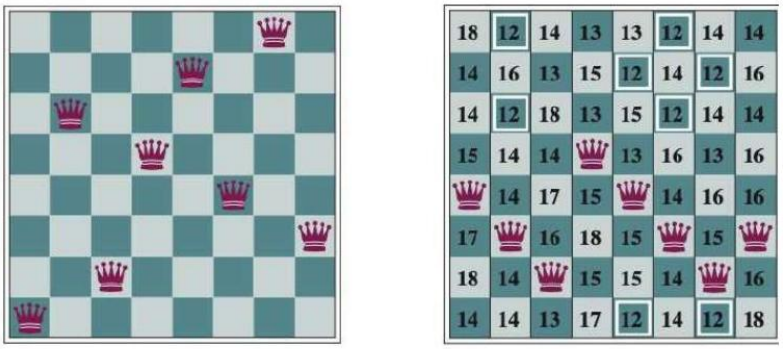


Figure 4.3 (a) The 8-queens problem: place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.) This position is almost a solution, except for the two queens in the fourth and seventh columns that attack each other along the diagonal. (b) An 8-queens state with heuristic cost estimate $h=17$. The board shows the value of h for each possible successor obtained by moving a queen within its column. There are 8 moves that are tied for best, with $h=12$. The hill-climbing algorithm will pick one of these.

2. **Simulated Annealing Search**

A hill-climbing algorithm that never makes —downhill moves towards states with lowervalue (or higher cost) is guaranteed to be incomplete, because it can stuck on a localmaximum. In contrast,a purely random walk –that is, moving to a successor chosen uniformly at random from the set of successors – iscomplete, but extremely inefficient. Simulatedannealing is analgorithm that combines hill-climbing with a random walk in some way that yields both

Efficiencyand completeness.

simulated annealing algorithm is quite similar to hill climbing. Instead of picking the bestmove,however, it picks the random move. If the move improves the situation, it is alwaysaccepted. Otherwise, the algorithm accepts the move with some probability less than 1.

The probability decreases exponentially with the —badness of the move – the amount Eby which the evaluation is worsened. The probability also decreases as the "temperature" Tgoes down: "bad moves are more likely to be allowed at

the start when temperature is high,and they become more unlikelyas Tdecreases. One can prove that if the schedule lowers Tslowly enough, the algorithm will find a globaloptimum with probability approaching 1.

Simulated annealing was first used extensively to solve VLSI layout problems. It has beenapplied widelyto factory scheduling and other large-scaleoptimization tasks.

SA is a metaheuristic optimization technique introduced by Kirkpatrick et al. in 1983 to solve the Travelling Salesman Problem (TSP).

The SA algorithm is based on the annealing process used in metallurgy, where a metal is heated to a high temperature quickly and then gradually cooled. At high temperatures, the atoms move fast, and when the temperature is reduced, their kinetic energy decreases as well. At the end of the annealing process, the atoms fall into a more ordered state, and the material is more ductile and easier to work with.

Similarly, in SA, a search process starts with a high-energy state (an initial solution) and gradually lowers the temperature (a control parameter) until it reaches a state of minimum energy (the optimal solution).

Advantages of Simulated Annealing

- **Ability to Escape Local Minima**: One of the most significant advantages of Simulated Annealing is its ability to escape local minima. The probabilistic acceptance of worse solutions allows the algorithm to explore a broader solution space.
- **Simple Implementation**: The algorithm is relatively easy to implement and can be adapted to a wide range of optimization problems.
- **Global Optimization**: Simulated Annealing can approach a global optimum, especially when paired with a well-designed cooling schedule.
- **Flexibility:** The algorithm is flexible and can be applied to both continuous and discrete optimization problems.

Unit-2

1.

a. Define Minimax Algorithm.

Answer:- The Mini-Max algorithm is a decision-making algorithm used in artificial intelligence, particularly in game theory and computer games. It is designed to minimize the possible loss in a worst-case scenario (hence "min") and maximize the potential gain (therefore "max").

b. What are the Challenges in Optimal Decision-Making?

Answer:-

- ✓ Complexity of Game Environments
- ✓ Real-Time Decision Making
- ✓ Incomplete Information
- ✓ Dynamic and Evolving Environments

c. Define Alpha and Beta.

- **Answer:- Alpha:** Alpha represents the best score that the maximizing player has found so far in a particular branch of the game tree. It is the highest score the maximizing player can achieve up to this point. Essentially, it tracks the maximizer's best-known option.
- **Beta:** On the other hand, Beta represents the best score that the minimizing player has found in a specific branch. It is the lowest score the minimizing player can allow up to this point. Beta tracks the minimizer's best-known option.

The main condition which required for alpha-beta pruning is:

- $\alpha >= \beta$

d. What are types of Constraint Satisfaction Problems?

Answer:-

- ✓ Binary CSPs
- ✓ Non-Binary CSPs
- ✓ Hard and Soft Constraints

e. State De Morgan’s Laws.

Answer:- These laws describe how **negations** distribute over **AND (∧)** and **OR (∨)** operations:

- **First Law:**
  - ¬(P∧Q)≡(¬P∨¬Q)

This means that the negation of a conjunction is equivalent to the disjunction of the negated propositions.

- **Second Law:**
  - ¬(P∨Q)≡(¬P∧¬Q)

This means that the negation of a disjunction is equivalent to the conjunction of the negated propositions.

2.

a. Write a short note on Minimax algorithm with example.

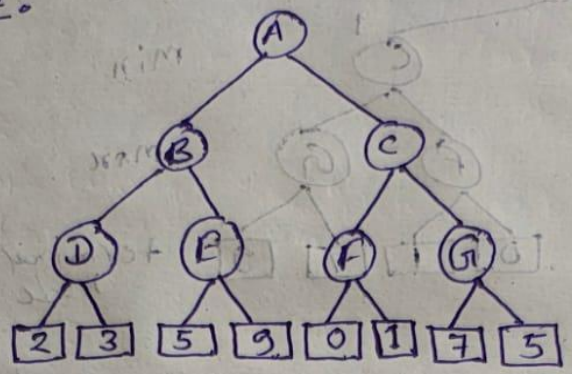
Answer:- The Min Max algorithm is a decision-making algorithm used in the field of game theory and artificial intelligence. It is used to determine the optimal strategies in games like chess, checkers, and tic-tac-toe. In a two-player game, one player takes on the role of the maximizer, seeking the best move to maximize their chances of winning, while the other player acts as the minimizer, attempting to minimize the maximizer's chances.

Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Example:

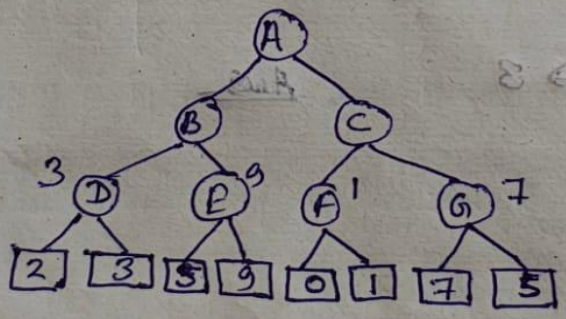
Example:



→ Maximizer
→ Minimizer
→ Maximizer
→ Terminal node

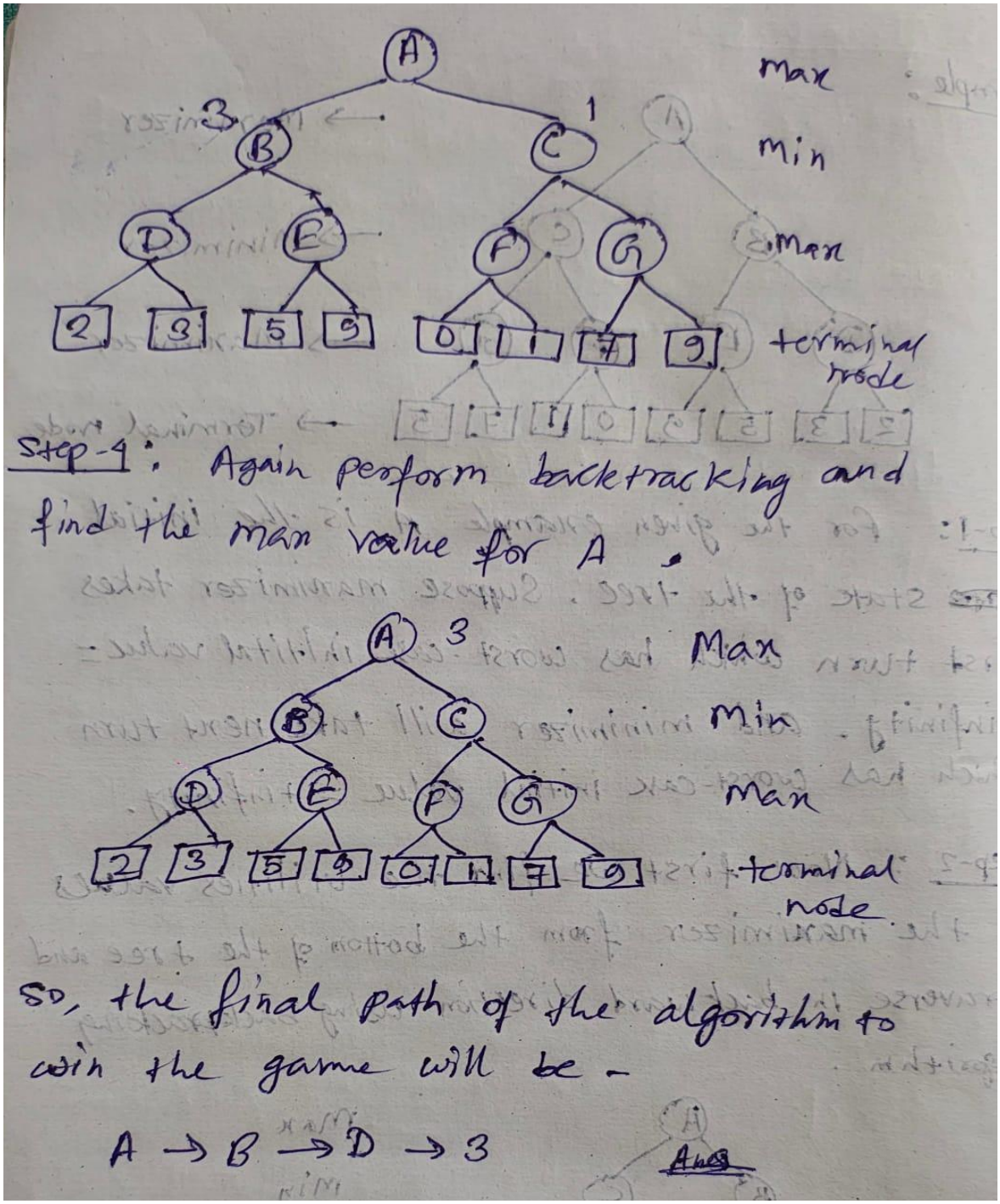
Step-1: For the given example A is the initial ~~state~~ state of the tree. Suppose maximizer takes first turn which has worst-case initial value = -infinity. and minimizer will take next turn which has worst-case initial value = +infinity.

Step-2: Now, first we find the utilities values of the maximizer from the bottom of the tree and traverse in backward direction using backtracking algorithm.



Max
Min
Max
terminal node

Step-3: Again, we have to perform backtracking and find the min value for B and C.



b. Discuss about Constraint Satisfaction Problems (CSP).

Answer:- Constraint Satisfaction Problems (CSP) represent a class of problems where the goal is to find a solution that satisfies a set of constraints. These problems are commonly encountered in fields like scheduling, planning, resource allocation, and configuration.

A **Constraint Satisfaction Problem** is a mathematical problem where the solution must meet a number of constraints. In a CSP, the objective is to assign values to variables such that all the constraints are satisfied. CSPs are used

extensively in artificial intelligence for decision-making problems where resources must be managed or arranged within strict guidelines.

Components of Constraint Satisfaction Problems

CSPs are composed of three key elements:

1. **Variables:** The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.
2. **Domains:** The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.
3. **Constraints:** The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Common applications of CSPs include:

- **Scheduling:** Assigning resources like employees or equipment while respecting time and availability constraints.
- **Planning:** Organizing tasks with specific deadlines or sequences.
- **Resource Allocation:** Distributing resources efficiently without overuse.

c. With a neat diagram explain about the architecture of knowledge-based agent.

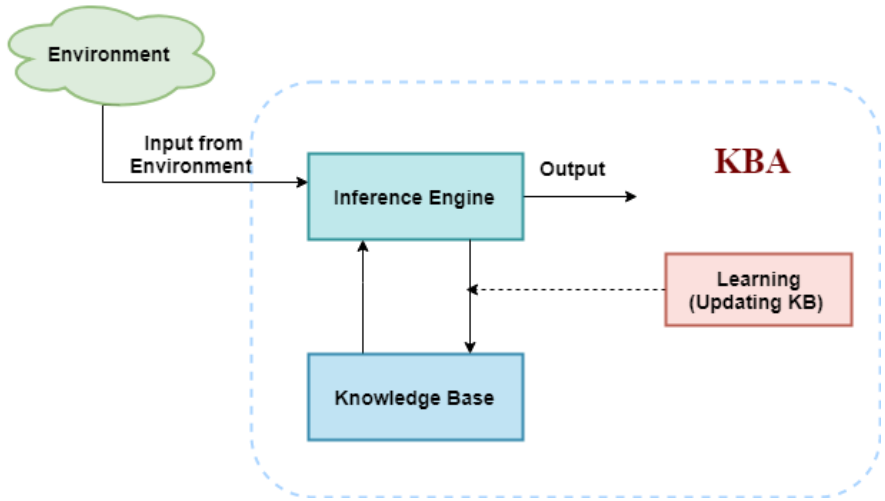
- **Answer:-** Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.

- Knowledge-based agents are composed of two main parts:
  - **Knowledge-base and**
  - **Inference system**.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) take input from the environment by perceiving the environment. The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

d. Explain logical connectives with truth table in propositional logic?

Answer:- Logical connectives are essential operators that combine **atomic propositions** to form **compound propositions**. These connectives allow AI

systems to build more complex rules and perform logical reasoning. Below are the most common connectives used in **propositional logic**:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called Truth table. Following are the truth table for all logical connectives:

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

e. Describe the Properties of Operators in Propositional Logic.

Answer:- In propositional logic, **logical operators** follow specific properties that allow us to **manipulate and simplify logical expressions**. Understanding these properties is essential for building efficient AI systems that rely on logical reasoning.

1. De Morgan's Laws

These laws describe how **negations** distribute over **AND (∧)** and **OR (∨)** operations:

- **First Law:**
  - $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$

This means that the negation of a conjunction is equivalent to the disjunction of the negated propositions.

- **Second Law:**
  - $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$

This means that the negation of a disjunction is equivalent to the conjunction of the negated propositions.

2. Commutative Property

This property states that the **order of the propositions** does not affect the result of **AND (∧)** and **OR (∨)** operations:

- **AND:**
  - $P \wedge Q \equiv Q \wedge P$
- **OR:**
  - $P \vee Q \equiv Q \vee P$

3. Associative Property

This property allows us to **group propositions** in any order when using **AND** or **OR** operations:

- **AND:**
  - $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$
- **OR:**
  - $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$

4. Distributive Property

This property states that **AND distributes over OR**, and vice versa:

- **AND over OR:**
  - P∧(Q∨R)≡(P∧Q)∨(P∧R)P
- **OR over AND:**
  - P∨(Q∧R)≡(P∨Q)∧(P∨R)

3.

a. Explain Alpha Beta Pruning Algorithm with an Example.

Answer:-

b. Explain about Propositional logic.

Answer:-

Propositional Logic is a formal system used in Artificial Intelligence (AI) to represent and reason about facts in a system. It deals with propositions, which are statements that can be either true or false. Propositional logic is used to build logical expressions using connectives like AND (∧), OR (∨), NOT (¬), IMPLIES (→), and IF AND ONLY IF (↔).

Key Components:

1. **Propositions:** Basic statements that can be true or false, e.g., "It is raining" or "John is at home."
2. **Logical Connectives:** These combine propositions to form more complex expressions:
   - AND (∧): Both propositions must be true.
   - OR (∨): At least one proposition must be true.
   - NOT (¬): Negates the truth value of a proposition.
   - IMPLIES (→): If the first proposition is true, the second must also be true.
3. **Truth Tables:** Used to evaluate the truth value of logical expressions based on all possible combinations of truth values of propositions.

Applications in AI:

- **Knowledge Representation:** Propositional logic represents facts and rules.
- **Reasoning**: It is used in inference systems to derive new facts from known facts.
- **Decision Making:** Helps agents decide on actions based on logical conclusions.

Despite its simplicity, propositional logic is limited in representing complex relationships, which is addressed by more advanced systems like First-Order Logic.

Applications of Propositional Logic in AI

1. **Knowledge Representation**:
   - Propositional logic is often used to represent facts about the world. Each proposition can represent a piece of knowledge or fact.
2. **Automated Reasoning and Inference**:
   - Propositional logic forms the basis for reasoning systems that deduce new facts or conclusions from known facts.
3. **Decision Making and Planning**:
   - Propositional logic is used in **decision-making** processes where an agent has to make decisions based on the logical relationships between propositions.
4. **Expert Systems**:
   - Propositional logic is used in expert systems to represent the knowledge base of an expert in a particular domain.
5. **Automated Theorem Proving**:
   - Propositional logic is used in automated theorem proving to prove or disprove statements using logical rules. This is essential in formal verification of systems and software.

c. Explain Propositional theorem proving.

Answer:-

Propositional Theorem Proving is a method in Artificial Intelligence (AI) used to determine whether a given logical formula (theorem) can be derived from a set of axioms or premises using formal logical rules. It plays a crucial role in automated reasoning, verification, and decision-making processes within AI systems.

Key Concepts:

1. **Propositions:**
   - These are atomic statements that are either true or false. For example, P, Q, and R can represent individual facts such as "It is raining" or "The light is on."

2. **Inference Rules:**
    - These are logical principles used to derive new facts from existing premises or axioms. Common rules include:
        - Modus Ponens: If P→Qand P are true, then Q is true.
        - Modus Tollens: If P→Q and ¬Q are true, then ¬ P is true.
3. **Theorem:**
    - A theorem is a formula that needs to be proven, typically derived from a set of axioms or facts. In AI, theorem proving is used to verify the correctness of solutions, check consistency in knowledge bases, or infer new knowledge.
4. **Axioms (Premises):**
    - These are a set of assumed truths from which theorems can be derived. They serve as the foundation for logical reasoning in propositional logic**.**

Common Methods for Propositional Theorem Proving:

1. **Truth Table Method:**
    - A truth table lists all possible truth values of the propositions and evaluates the formula based on these values. The formula is a theorem if it holds true for all possible truth assignments.
    - Limitations: For formulas with many propositions, the number of rows in the truth table grows exponentially, making it inefficient for complex formulas.
2. **Semantic Tableaux (Truth Tree):**
    - The formula is broken down into simpler components. If contradictions are found during the decomposition process, the formula is not valid. Otherwise, it is considered valid.
    - Process: The formula is negated, and the proof proceeds by systematically checking for contradictions. If no contradictions appear, the formula is a theorem.
3. **Resolution:**
    - The formula is transformed into Conjunctive Normal Form (CNF), where the formula is expressed as a conjunction of disjunctions. The resolution rule is applied to combine clauses and derive new clauses. If an empty clause (a contradiction) is derived, the formula is unsatisfiable, proving the negation is false, and the theorem is true.

- Efficient Method: Resolution is widely used in automated theorem proving systems and SAT solvers, as it allows for efficient search strategies in large knowledge bases.

4. **Davis-Putnam-Logemann-Loveland (DPLL) Algorithm:**
   - The DPLL algorithm is an efficient procedure for solving the Satisfiability (SAT) problem, which is central to propositional theorem proving. It simplifies the formula using backtracking, unit propagation, and pure literal elimination to find a solution or prove the formula unsatisfiable.

Applications in AI:

1. Automated Reasoning
2. Expert Systems
3. SAT Solvers
4. Formal Verification

d. Explain constraint satisfaction problem with graph coloring as example.

Answer:- For example, let us formulate the map coloring problem for the map of Australia, as shown below:
The CSP formulation for this problem is:

- X: {WA, NT, SA, Q, NSW, V, T}, where each variable represents a state or territory of Australia.
- D: {red, green, blue}, where each variable has the same domain of three colors.
- C: {< (WA, NT), WA!= NT >, < (WA, SA), WA!= SA >, < (NT, SA), NT!= SA >, < (NT, Q), NT!= Q >, < (SA, Q), SA!= Q >, < (SA, NSW), SA!= NSW >, < (SA, V), SA!= V >, < (Q, NSW), Q!= NSW >, < (NSW, V), NSW!= V >}, where each constraint is a binary constraint that states that two adjacent regions must have different colors.
  The algorithm starts with an empty assignment and selects the first variable to assign. According to the MRV heuristic, the variable with the

most constraints is SA, as it has six neighbors. The algorithm then tries to assign a value to SA, and according to the LCV heuristic, it chooses red, as it is the least constraining value for the neighboring variables. The algorithm then uses forward checking to prune the domains of the neighboring variables and updates the domains as follows:

- **WA: {green, blue}**
- **NT: {green, blue}**
- **Q: {green, blue}**
- **NSW: {green, blue}**
- **V: {green, blue}**
- **T: {red, green, blue}**

  The algorithm then recurses to the next level and selects the next variable to assign. According to the MRV heuristic, the variable with the most constraints is NT, as it has three neighbors. The algorithm then tries to assign a value to NT, and according to the LCV heuristic, it chooses green, as it is the least constraining value for the neighboring variables. The algorithm then uses forward checking to prune the domains of the neighboring variables and updates the domains as follows:

- **WA: {blue}**
- **Q: {blue}**
- **SA: {red}**
- **NSW: {green, blue}**
- **V: {green, blue}**
- **T: {red, green, blue}**

  The algorithm then recurses to the next level and selects the next variable to assign. According to the MRV heuristic, the variable with the most constraints is WA, as it has two neighbors. The algorithm then tries to assign a value to WA, and according to the LCV heuristic, it chooses blue, as it is the only remaining value for WA. The algorithm then uses forward checking to prune the domains of the neighboring variables and updates the domains as follows:

- **NT: {green}**
- **SA: {red}**
- **Q: {blue}**
- **NSW: {green}**
- **V: {green, blue}**
- **T: {red, green, blue}**

e. Explain the Horn Clauses and Definite Clauses in detail.

Answer:-

Horn Clauses and Definite Clauses are fundamental concepts in logic programming and artificial intelligence, particularly in knowledge representation and reasoning. These clauses are a special kind of logic formula used to simplify reasoning tasks and make logical inference more efficient. They are primarily used in Prolog and other logic programming languages to represent rules and facts.

Horn Clauses

A Horn clause is a special form of a clause (a disjunction of literals) in propositional or first-order logic. A Horn clause has at most one positive literal (i.e., a literal that is not negated) and any number of negative literals (literals that are negated).

- **Definition: A Horn clause is a disjunction of literals of the form:**

  ¬P1V¬P2V…V¬PnVQ

  where:
  - P1,P2,…,Pn are negative literals (i.e., negated propositions).
  - Q is a positive literal (i.e., a non-negated proposition).
- **Key Characteristics:**
  - At most one positive literal: Horn clauses have one or zero positive literals.
  - A rule: Horn clauses are used to represent rules in logic programming. They are often written as implications, where the left-hand side represents the conditions (antecedent), and the right-hand side represents the conclusion (consequent).
- **Example:**

  1. ¬PV¬QVR (which can be interpreted as: "If P and Q are both false, then R is true").
  2. P→ Q is equivalent to ¬PVQ.

Definite Clauses

A Definite Clause is a specific type of Horn Clause where there is exactly one positive literal. A definite clause is a disjunction of literals with exactly one positive literal and any number of negative literals.

- **Definition: A Definite Clause is a Horn clause where there is exactly one positive literal:**

  ¬P1V¬P2V…V¬PnVQ

where:

- P1,P2,…,Pn are negative literals (negated propositions).
- Q is the single positive literal (non-negated proposition).

- **Key Characteristics:**
  - Exactly one positive literal: Definite clauses always have a single positive literal.
  - Implication form: Definite clauses are typically used to represent implications where the conditions (on the left-hand side) lead to a conclusion (on the right-hand side).
  - Used in Logic Programming: Definite clauses are the core building blocks of programs in languages like Prolog. Each rule in Prolog can be represented as a definite clause.
- **Example:**
  - ¬PV¬QVR is a definite clause and can be interpreted as: "If Pand Q are false, then R is true."
  - P→ Q is equivalent to ¬PVQ, which is also a definite clause.

4.

a. Write short note on the following Algorithm:

i. Backtracking Algorithm

ii. Forward-Checking Algorithm

iii. Constraint Propagation Algorithm

Answer:-

1. Backtracking Algorithm

The **backtracking algorithm** is a depth first search method used to systematically explore possible solutions in CSPs. It operates by assigning values to variables and backtracks if any assignment violates a constraint.

How it works:

- The algorithm selects a variable and assigns it a value.
- It recursively assigns values to subsequent variables.
- If a conflict arises (i.e., a variable cannot be assigned a valid value), the algorithm backtracks to the previous variable and tries a different value.

- The process continues until either a valid solution is found or all possibilities have been exhausted.

This method is widely used due to its simplicity but can be inefficient for large problems with many variables.

2. Forward-Checking Algorithm

The **forward-checking algorithm** is an enhancement of the backtracking algorithm that aims to reduce the search space by applying **local consistency** checks.

How it works:

- For each unassigned variable, the algorithm keeps track of remaining valid values.
- Once a variable is assigned a value, local constraints are applied to neighbouring variables, eliminating inconsistent values from their domains.
- If a neighbor has no valid values left after forward-checking, the algorithm backtracks.

This method is more efficient than pure backtracking because it prevents some conflicts before they happen, reducing unnecessary computations.

3. Constraint Propagation Algorithms

Constraint propagation algorithms further reduce the search space by enforcing **local consistency** across all variables.

How it works:

- Constraints are propagated between related variables.
- Inconsistent values are eliminated from variable domains by leveraging information gained from other variables.
- These algorithms refine the search space by making **inferences**, removing values that would lead to conflicts.

Constraint propagation is commonly used in conjunction with other CSP algorithms, such as **backtracking**, to increase efficiency by narrowing down the solution space early in the search process.

b. What is knowledge-based agent in artificial intelligence? Why we use a knowledge base? What are the various levels of knowledge-based agent? Write any two approaches to designing a knowledge-based agent?

Answer:- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.

- Knowledge-based agents are composed of two main parts:
  - **Knowledge-base and**
  - **Inference system**.

Why use a knowledge base?

- Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Various levels of knowledge-based agent:

A knowledge-based agent can be viewed at different levels which are given below:

1. Knowledge level

Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Logical level:

At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs. At the logical level we can expect to the automated taxi agent to reach to the destination B.

3. Implementation level:

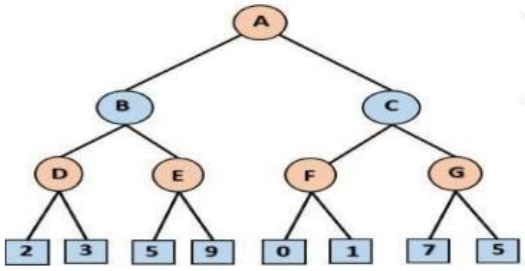
This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

1. **1. Declarative approach:** We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
2. **2. Procedural approach:** In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

c. Analyze alpha-beta pruning algorithm and the Min-max game playing algorithm for a given tree.



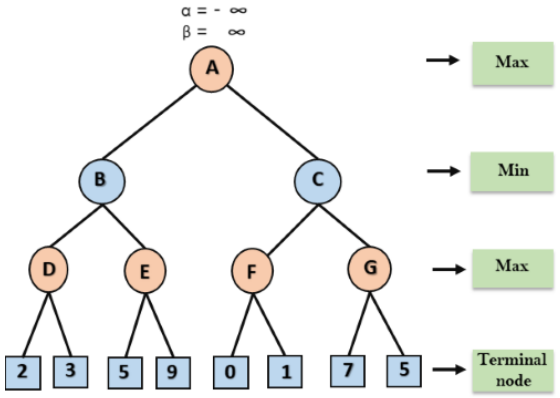
Answer:- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.

- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- **Alpha:** Alpha represents the best score that the maximizing player has found so far in a particular branch of the game tree. It is the highest score the maximizing player can achieve up to this point. Essentially, it tracks the maximizer's best-known option.
- **Beta:** On the other hand, Beta represents the best score that the minimizing player has found in a specific branch. It is the lowest score the minimizing player can allow up to this point. Beta tracks the minimizer's best-known option.
- The main condition which required for alpha-beta pruning is:α>=β

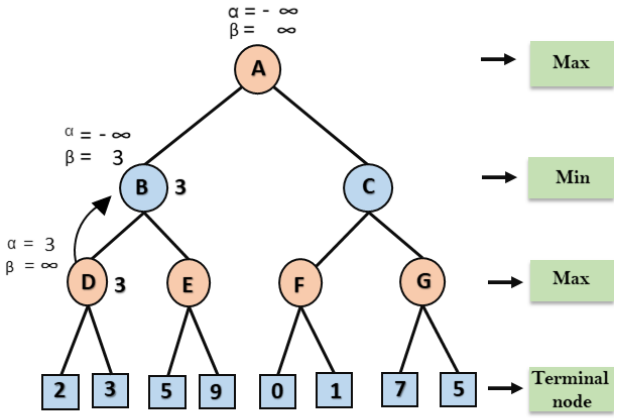
Example:

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



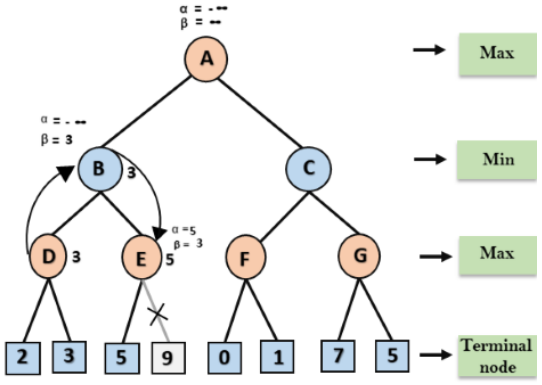
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

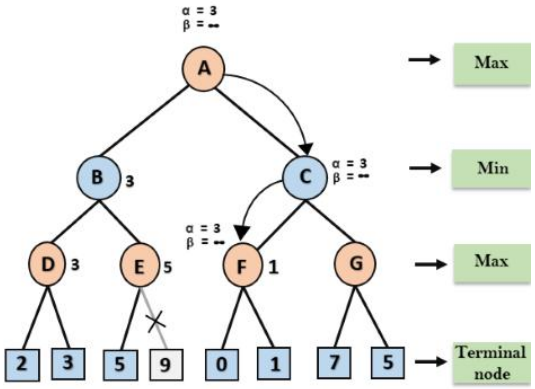
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



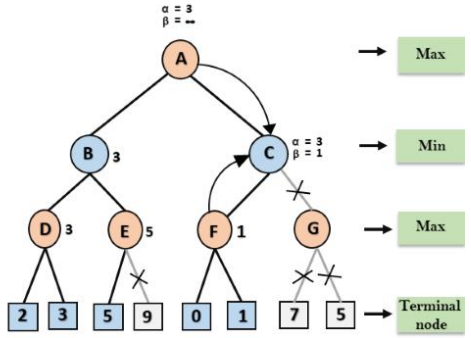
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

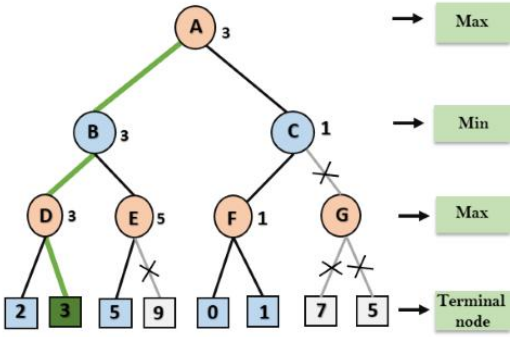
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Unit-3

1.

a. Specify the syntax of First-order logic in BNF form.

Answer:- The syntax of First-order Logic in BNF is:

<term> ::= <variable> | <constant> | <function>(<term>, <term>, ...)
<atomic_formula> ::= <predicate>(<term>, <term>, ...)
<formula> ::= <atomic_formula>
| <formula>∧<formula>
| <formula>∨<formula>
| ¬ <formula>
| ∀<variable> . <formula>
| ∃<variable> . <formula>

b. What is quantifier?

Answer:- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression.

c. Define wumpus world?

Answer:- The Wumpus world is a cave with *16* rooms *(4×4)*. Each room is connected to others through walkways (no rooms are connected diagonally). The knowledge-based agent starts from *Room[1, 1]*. The cave has – some pits, a treasure and a beast named Wumpus. The Wumpus can not move but eats the one who enters its room. If the agent enters the pit, it gets stuck there. The goal of the agent is to take the treasure and come out of the cave. The agent is rewarded, when the goal conditions are met. The agent is penalized, when it falls into a pit or being eaten by the Wumpus.

d. Evaluate the given sentence “All Pompians were Romans” write a well-formed formula in predicate logic.

Answer:- ∀x:Pompians(x) ->Romans(x)

Here, Pompians(x)**->**Romans(x) = If x is a Pompians then x is Romans and x can be anyone(all)

e. What is unification?

Answer:-Unification in AI aims to develop models and systems that can handle multiple tasks across various cognitive domains. Instead of designing specialised AI systems for narrowly defined tasks (like chatbots for conversation or algorithms for image recognition), the goal is to integrate these systems into a unified framework capable of functioning cohesively.

2.

a. Explain inference rules for quantifiers?

Answer:- As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- Universal Generalization
- Universal Instantiation
- Existential Instantiation
- Existential introduction

1. **Universal Generalization:**

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.

2. Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from ∀ x P(x) for any object in the universe of discourse.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$.

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.

- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$

b. Illustrate the syntax and semantics of first order logic.

Answer:-The syntax of FOL defines how formulas are structured using terms, predicates, connectives, and quantifiers. The semantics assigns meanings to these symbols within a domain, allowing us to interpret the formulas and evaluate their truth values.

Syntax of First-Order Logic (FOL)

The syntax of First-Order Logic specifies the rules for constructing valid expressions. These expressions include terms, predicates, logical connectives, and quantifiers.

- Terms: Represent objects in the domain.
  - Constants: Specific objects, e.g., John, 3.
  - Variables: Arbitrary objects, e.g., x, y.
  - Functions: Mappings that return objects, e.g., father(x).
- **Predicates (Atomic Formulas):** Represent relations between terms, e.g., Likes(x, y).
- **Formulas**: Formed using atomic formulas, logical connectives, and quantifiers.
  - **Connectives**: ∧ (AND), ∨ (OR), ¬ (NOT), → (IF...THEN), ↔ (IF AND ONLY IF).

- **Quantifiers**: ∀ (for all), ∃ (there exists).

The structure of a formula could be something like:
<formula> ::= <atomic_formula> | <formula>∧<formula> | ∀<variable>.<formula> | ∃<variable>.<formula>

Semantics of First-Order Logic (FOL)

The semantics of First-Order Logic defines the meaning of the formulas in a given domain of discourse.

- **Domain of Discourse:** The set of all objects under consideration (e.g., people, numbers).
- **Interpretation**: Assigns specific meanings to constants, functions, and predicates in the domain.
  - Constants: Refer to specific objects.
  - Variables: Can refer to any object in the domain.
  - Predicates: Represent relations, mapped to sets of tuples of domain objects.
- **Quantifiers**:
  - Universal Quantifier (∀): ∀x P(x)∀x \, P(x) means the predicate P(x)P(x) holds for all objects in the domain.
  - Existential Quantifier (∃): ∃x P(x)∃x \, P(x) means there exists at least one object in the domain such that P(x)P(x) holds.

c. Explain backward chaining process?

Answer:-

Backward Chaining is a reasoning technique used in logic-based systems, especially in automated theorem proving and expert systems. It is a form of goal-driven reasoning, where the process starts with a goal or query and works backward to find supporting facts or evidence.

In First-Order Logic (FOL), backward chaining involves trying to prove a query (the goal) by recursively breaking it down into sub-goals, until you reach known facts or premises that can be directly verified.

Steps in Backward Chaining Process

1. **Start with the Goal:**
   - The process begins with a specific goal or query, typically in the form of a formula that needs to be proven. For example, you want to prove Likes(John,Mary).
2. **Look for Matching Rules:**

- Check if there are any inference rules or knowledge base facts that match the goal. These rules typically have the form of implications or conditionals (e.g., A→B).
- The goal becomes the consequent (right-hand side) of the rule, and the process tries to satisfy the antecedent (left-hand side) by recursively solving the sub-goals.

For example, if you have a rule like:

$$\text{Likes}(x,y) \rightarrow \text{Food}(y)$$

and your goal is Likes(John,Mary), then you would need to prove Food(Mary).

3. **Recursive Sub-Goal Generation:**
   - The goal Likes(John,Mary) has now been transformed into a new sub-goal Food(Mary).
   - This new sub-goal is again processed using backward chaining. You will check the knowledge base or rules to find a fact or a rule that leads to Food(Mary).
   - Continue breaking down the problem in this way until you reach facts that are already known (e.g., Food(Mary) is true because "Mary is a food").
4. Verify Facts:
   - If you eventually reach a fact that is known (i.e., it directly matches an assertion in the knowledge base), then the goal has been successfully proven.
   - If you can't find any rules or facts to support a sub-goal, then the original goal can't be proven.
5. Backtrack if Necessary:
   - If one branch of reasoning fails (i.e., you can't prove a sub-goal), you backtrack and try another rule or hypothesis.

Example of Backward Chaining Process

Let's work through an example:

Given:

- Knowledge Base:
  - Likes(John,y)→Food(y) (If John likes something, it is food)
  - Food(Mary) (Mary is food)
  - Likes(John,Mary) (John likes Mary)
- Goal: Prove Food(Mary) (Is Mary food?)

Backward Chaining Steps:

1. Start with the goal: Food(Mary).

2. Check the knowledge base. Find that the fact Food(Mary) is explicitly stated in the knowledge base, so the goal is already satisfied.
    - The fact Food (Mary) is true, and no further reasoning is required.

d. Discuss diagnostic rules and causal rules in FOL?

In First-Order Logic (FOL), diagnostic rules and causal rules are two important concepts that deal with reasoning about causes and effects, particularly in fields like artificial intelligence, medicine, and systems theory. Here's an explanation of both types of rules:

Diagnostic Rules

Diagnostic rules in FOL are typically used to determine the possible causes or reasons behind observed effects. In a diagnostic context, we start with an observation and try to infer the causes or conditions that may have led to that observation.

Key Characteristics:

1. **Observation-based**: Diagnostic rules are concerned with explaining what is happening or has already happened.
2. **Inference**: These rules help infer the causes based on the observed outcomes, often with some level of uncertainty or probabilistic reasoning.
3. **Example**: In a medical diagnosis system, a symptom might be observed (like a fever), and the diagnostic rule helps deduce the possible causes (like a viral infection or bacterial infection).

Example in FOL:

Suppose you have the following knowledge:

- Fever(x): Person x has a fever.
- Infection(x): Person x has an infection.
- HasVirus(x): Person x has a viral infection.

Causal Rules

Causal rules, on the other hand, describe how one event or condition leads to another. These rules help model the dynamics of systems by specifying how the occurrence of certain events or actions causes other events or changes in the system.

Key Characteristics:

1. **Cause-and-effect**: Causal rules define relationships where one event or fact causes another.
2. **Predictive**: They are often used to predict the consequences of actions or changes in the environment.
3. **Example**: In a control system, an action (like turning on a heater) causes a change in the temperature.

e. Explain Numbers, sets, and lists in FOL.

1. Numbers in FOL

In FOL, numbers are often represented as constants or through the use of functions and relations to model arithmetic operations.

- Constants: Numbers can be treated as constants, for example, 0, 1, 2, etc. Each number can be represented as an individual constant in the domain of discourse.
- Functions: FOL may introduce functions like + (addition), - (subtraction), and * (multiplication) that take numbers as arguments and return a number as a result. For example, plus(x, y, z) could be a function representing the equation x + y = z.
- Relations: Numbers can also be defined using binary relations. For example, less_than(x, y) could represent the relationship

where x is less than y. Additionally, equality = can be used to express numerical equality (e.g., x = y).

2. Sets in FOL

Sets can be modeled in FOL, although FOL does not have an explicit set-theoretic concept like modern set theory. Instead, sets are typically modeled using **predicates** or **functions**.

- **Predicate**: A set can be defined as a predicate, where the predicate indicates membership. For example, In(x, A) could represent the statement "x is an element of set A."
- **Set operations**: Various set operations (like union, intersection, subset, etc.) can be encoded using predicates or functions:
    - Union: Union(A, B, C) could mean that C is the union of sets A and B, i.e., for any element x, In(x, C) $\leftrightarrow$ (In(x, A) $\vee$ In(x, B)).
    - Intersection: Intersection(A, B, C) means that C contains only elements that are in both A and B.
    - Subset: Subset(A, B) means that every element of A is also in B, i.e., $\forall$x (In(x, A) $\rightarrow$ In(x, B)).
    -

3. Lists in FOL

In FOL, **lists** can be represented as ordered sequences of elements. While FOL doesn't have a native concept of a list, it can be modeled using **tuples**, **functions**, or **recursion**.

- **Ordered pairs**: Lists can be modeled using ordered pairs or n-tuples. For example, a list of two elements [x, y] could be represented as a function or a predicate Pair(x, y).

- **Recursion and sequences**: A list can be defined recursively in FOL. For instance, a simple list of natural numbers might be defined using a **constructor** (like Cons(x, y)), which represents a list with the first element x and the remainder y. The empty list can be represented as Nil. So, a list [1, 2, 3] might be modeled as Cons(1, Cons(2, Cons(3, Nil))).

A list can be formalized using the following:

- First(x, L) could represent that x is the first element in the list L.
- Rest(L, L') could represent the rest of the list L, i.e., everything except the first element of L

Example in FOL

Numbers:

- **Predicate**: Successor(x, y) represents y is the successor of x.
    - Successor(0, 1) means "1 is the successor of 0."

Sets:

- **Predicate**: In(x, A) means x is an element of set A.
    - In(x, A) → ∃y In(y, B) could represent "If x is in set A, then there exists an element y in set B."

Lists:

- **Constructor**: Cons(x, y) could represent the list with first element x and remainder y.
    - Cons(1, Cons(2, Nil)) represents the list [1, 2].

3. **a. Difference between backward chaining and forward chaining**

Feature	Forward Chaining	Backward Chaining
Direction	Starts from facts and moves towards the goal	Starts from the goal and moves towards facts
Approach	Data-driven (bottom-up)	Goal-driven (top-down)
Inference Process	Applies rules to infer new facts	Searches for facts to support a given goal
Goal Knowledge	The goal is not defined at the start	The goal is defined at the start
UseCase	Exploring all possibilities	Solving a specific problem or goal

b. Explain resolution in predicate logic with suitable example.

In predicate logic, resolution is a rule of inference used to infer conclusions by refuting contradictions. It is one of the key techniques used in automated theorem proving and logic-based AI systems.

Resolution Process:

1. Select two clauses: Choose two clauses that have complementary literals (one clause has a literal and the other has its negation).
2. Unify the complementary literals: Find a unifier (a substitution) that can make the literals identical.

3. Combine the clauses: After unification, combine the remaining parts of the clauses to form a new clause, called the resolvent.
4. Repeat: Continue applying the resolution rule to new clauses until either a contradiction is derived (empty clause) or no further resolvents can be produced.

Example of Resolution in Predicate Logic:

Let's use an example with predicate logic to demonstrate resolution.

Premises:

1. ∀x (P(x) → Q(x)) (For all x, if P(x) is true, then Q(x) is true)
2. P(a) (P(a) is true for some specific a)

We want to derive Q(a) using resolution.

Step-by-step resolution:

1. Convert the premises into clausal form (a set of clauses):
    - The first premise ∀x (P(x) → Q(x)) can be written as ¬P(x) ∨ Q(x) (because P(x) → Q(x) is equivalent to ¬P(x) ∨ Q(x)).
    - The second premise is simply P(a).

   Now, we have the following two clauses:
    - Clause 1: ¬P(x) ∨ Q(x)
    - Clause 2: P(a)
2. Apply resolution:
    - In the two clauses, we have a complementary pair of literals: ¬P(x) in Clause 1 and P(a) in Clause 2.
    - We unify P(x) with P(a), so we substitute x with a.

- After unification, Clause 1 becomes: ¬P(a) ∨ Q(a), and Clause 2 is just P(a).

3. Resolve the complementary literals:
   - By resolving the two clauses, we get the resolvent: Q(a). This is the result of the resolution, as we have eliminated the complementary literals and combined the rest.

Thus, from the premises, we have derived Q(a).

c. Explain knowledge engineering process in FOL?

The Knowledge Engineering process in First-Order Logic (FOL) within Artificial Intelligence (AI) involves designing, developing, and maintaining knowledge-based systems. These systems are used to represent and reason about knowledge in a formal and structured way. FOL provides a powerful way to capture and infer logical relationships, making it ideal for representing complex knowledge in AI systems.

Here's an overview of the key steps involved in the knowledge engineering process when using First-Order Logic (FOL):

1. Problem Definition

The first step is to clearly define the problem domain. This involves understanding what the system needs to know and what kind of reasoning or tasks it should perform. In AI, this can include natural language processing, expert systems, robotics, etc.

- **Example:** In a medical diagnosis system, the problem definition would involve identifying medical conditions based on symptoms, patient history, etc.

2. Knowledge Acquisition

Knowledge acquisition is the process of gathering relevant information, facts, and rules from various sources. These sources could include human experts, databases, documents, or other systems.

- **Example:** In the medical system, this would involve gathering medical knowledge, such as rules for diagnosing diseases based on symptoms, treatments, and patient conditions.

3. Knowledge Representation

In this step, the acquired knowledge is represented in **First-Order Logic (FOL)**. FOL allows for expressing facts and relationships in a formal way using terms, predicates, variables, quantifiers, and logical connectives (AND, OR, NOT, etc.).

- **FOL Components:**
    - **Terms:** Represent objects in the domain (e.g., John, Mary, car).
    - **Predicates:** Represent relationships or properties of objects (e.g., Likes(John, IceCream), Parent(John, Mary)).
    - **Variables:** Placeholders for terms (e.g., x, y).
    - **Quantifiers:** Universal (∀) and existential (∃) quantifiers to express generality and existence.
    - **Logical Connectives:** Combine sentences (e.g., conjunction, disjunction, negation).
- **Example:** In medical diagnosis, you could represent a rule like:Cough(x) ∧ Fever(x) → Flu(x)\text{Cough(x) ∧ Fever(x) →

Flu(x)}Cough(x) ∧ Fever(x) → Flu(x)This says that if a patient x has a cough and a fever, then x is diagnosed with the flu.

4. Knowledge Structuring

Once the knowledge is represented in FOL, it must be structured effectively to ensure that reasoning and inference can be done efficiently. This may involve organizing knowledge into specific domains or hierarchies, ensuring that the relationships between concepts are logically coherent.

- **Example:** In a medical system, you might have a hierarchy where diseases are grouped under categories like "viral infections," "bacterial infections," etc.

5. Inference and Reasoning

After knowledge is structured, the system must be capable of drawing conclusions from the knowledge base using logical inference. In FOL, inference can be done using **forward chaining**, **backward chaining**, or **resolution**.

- **Forward Chaining:** Starts with known facts and applies inference rules to deduce new facts.
- **Backward Chaining:** Starts with a goal and works backward to find supporting facts or rules.
- **Resolution:** A refutation-based reasoning method to derive conclusions by combining and simplifying logical sentences.
- **Example:** If the system has facts like:

Cough(John)Fever(John)\text{Cough(John)} \quad \text{Fever(John)}Cough(John)Fever(John)

and a rule:

Cough(x) ∧ Fever(x) → Flu(x)\text{Cough(x) ∧ Fever(x) → Flu(x)}Cough(x) ∧ Fever(x) → Flu(x)

It can infer that Flu(John).

6. Validation and Verification

This step ensures that the knowledge represented in the system is both correct (valid) and complete. Validation checks that the knowledge base accurately represents real-world knowledge, while verification ensures that the system can correctly reason and produce the expected outputs.

- Example: In the medical system, validation would involve comparing the system's diagnosis to known medical cases, ensuring the system's rules correctly match established medical knowledge.

7. Maintenance and Updating

Over time, new knowledge might emerge, and existing knowledge might need to be updated. Therefore, a knowledge-based system requires ongoing maintenance. This includes adding new facts, modifying existing rules, and ensuring that the system continues to reason effectively with the updated knowledge base.

- Example: In a medical system, new research or medical findings may require updates to the rules for diagnosing diseases.

Summary:

The knowledge engineering process in First-Order Logic involves:

1. **Problem Definition**: Clearly define the domain and goals.
2. **Knowledge Acquisition**: Gather knowledge from relevant sources.

3. **Knowledge Representation**: Represent the knowledge in FOL using predicates, terms, and logical structures.
4. **Knowledge Structuring**: Organize the knowledge for efficient reasoning.
5. **Inference and Reasoning**: Use logical inference to derive new knowledge or conclusions.
6. **Validation and Verification**: Ensure the knowledge is accurate and the system works as expected.
7. **Maintenance and Updating**: Continuously update the knowledge base as new information becomes available.

d. Define Ontological Engineering? Explain with the diagram the upper ontology of the world.

Upper Ontology of the World (in AI):

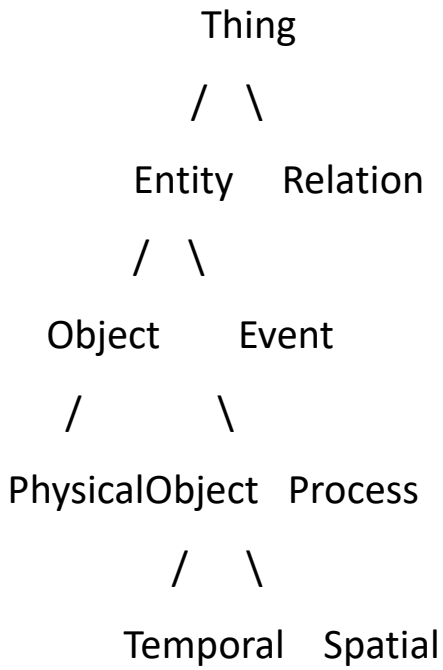
An upper ontology (or foundational ontology) is a high-level ontology that provides a common and abstract framework for a wide range of domain-specific ontologies. It defines broad, general concepts that are universal and independent of specific applications.

In the context of AI, an upper ontology serves as a foundation for other more specific ontologies, facilitating the sharing of knowledge across diverse domains. Some well-known upper ontologies include SUMO (Suggested Upper Merged Ontology), DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering), and BFO (Basic Formal Ontology).

The upper ontology of the world might include broad categories like Thing, Object, Event, Process, Relation, Time, Space, and so on. These categories are generalized concepts that can be specialized or refined in more domain-specific ontologies.

Diagram of the Upper Ontology of the World:

Here is a simplified view of an upper ontology in AI (you can think of this as a hierarchical structure):



Explanation of the Diagram:

1. Thing: The most general concept. Everything is a "thing."
2. Entity: A subset of things that exist in some sense, including objects and events.
3. Object: A physical or abstract entity with some form, structure, or identity.
4. Event: Something that happens; a state change or an occurrence over time.

5. Relation: Describes the relationships between entities (e.g., "is a part of," "is located at").
6. Process: A dynamic sequence of events that produces a change in the state of an object or entity.
7. Time and Space: Concepts that relate to the spatial and temporal properties of events, objects, and processes.

This upper ontology provides a framework for organizing knowledge across all domains, allowing specific knowledge bases or ontologies to be built upon these core structures. It helps create interoperability among various AI systems and ensures that knowledge can be shared effectively across different applications**.**

e. Write short notes on universal and existential quantification in FOL?

In First-Order Logic (FOL), quantification is used to express statements about "all" or "some" elements in a domain. Two primary types of quantifiers are used:

1. Universal Quantification (∀):

- The universal quantifier is represented by the symbol ∀ and expresses that a property or relation holds for all elements in a given domain.

The general form is:
∀x P(x)
This means P(x) is true for every possible value of x in the domain.

Example:
∀x (Human(x) → Mortal(x))
This means "For all x, if x is a human, then x is mortal."

It is used to generalize statements over all members of a domain.

2. Existential Quantification (∃):

- The existential quantifier is represented by the symbol ∃ and expresses that there exists at least one element in the domain for which a given property or relation holds.
- The general form is:
  ∃x P(x)
  This means "There exists at least one x in the domain such that P(x) is true."

Example:
∃x (Human(x) ∧ Rich(x))
This means "There exists some x such that x is both human and rich."

4 a. Explain the forward chaining process and efficient forward chaining in detail with example. What is the need of incremental forward chaining?

Forward Chaining Process in AI

Forward chaining is a data-driven approach used in rule-based systems, typically for problem-solving and inference tasks. It involves

starting with a set of known facts (known as facts or premises) and applying inference rules to derive new facts until a goal is reached or no further inference can be made.

Here's how forward chaining works step by step:

1. Start with initial facts: These are the known facts or premises that are true at the beginning.
2. Identify applicable rules: In forward chaining, you have a set of rules of the form:
    - IF condition(s) THEN action (or conclusion)

The goal is to apply rules to the current facts. The conditions of the rule (IF part) must match with the current facts for the rule to be applicable.

3. Apply the rules: When a rule is applicable (its conditions are met), you apply it, which leads to new facts being generated. These new facts become part of the knowledge base.
4. Repeat the process: The newly derived facts are then treated as potential new premises. The process continues until the goal is reached or no more rules can be applied.
5. Goal check: In some applications, the system may have a specific goal or conclusion to reach, and forward chaining continues until the goal is derived from the available facts.

Example of Forward Chaining

Let's say you have a rule-based system to determine whether someone is eligible for a loan. The system uses the following rules:

1. Rule 1: IF income > 50,000 THEN eligible for loan.
2. Rule 2: IF age > 18 AND income > 50,000 THEN eligible for loan.

You start with the following facts:

- Fact 1: Age = 25
- Fact 2: Income = 60,000

Process of Forward Chaining:

- The system starts with the facts: Age = 25 and Income = 60,000.
- Rule 1 is applied because Income > 50,000. The conclusion of this rule is that the person is eligible for a loan.
- Now, the system has derived the fact: "eligible for loan."

No further rules are needed to apply because the goal (eligibility) has been reached.

Efficient Forward Chaining

While the basic forward chaining process is simple, it can become computationally expensive when there are many rules and facts, especially if some facts and rules are redundant or irrelevant. Efficient forward chaining aims to improve the process by minimizing unnecessary computations and optimizing the way rules are applied. Here are key techniques used for efficient forward chaining:

1. Rule Prioritization:
    - Some rules might be more likely to apply or lead to the goal faster. Prioritize rules that are more likely to lead to new facts or directly to the goal.
2. Fact Deduplication:
    - Avoid applying the same rule repeatedly if it has already been applied. Similarly, prevent generating the same facts more than once.
3. Conflict Resolution:

    - When multiple rules are applicable at the same time, select one based on certain criteria (such as rule priority or heuristic measures).
4. Inference Strategies:
    - Selective Forward Chaining: Apply only those rules that have a high likelihood of leading to new facts.
    - Goal-directed Chaining: Focus on applying rules that are more likely to lead to the goal, instead of just deriving facts in a general manner.

Example of Efficient Forward Chaining

Let's consider a case where the system is determining whether a person is eligible for a loan, but the set of rules is larger and more complex.

- Rule 1: IF income > 50,000 THEN eligible for loan.
- Rule 2: IF age > 18 AND income > 50,000 THEN eligible for loan.
- Rule 3: IF credit score > 650 THEN eligible for loan.
- Rule 4: IF marital status = married AND income > 50,000 THEN eligible for loan.

If you already know that income is greater than 50,000, there's no need to check rules that focus on other factors (e.g., marital status or credit score) unless it's necessary to fine-tune eligibility. By focusing only on the most relevant rules, the system can reduce unnecessary checks and computations, resulting in better performance.

The Need for Incremental Forward Chaining

Incremental forward chaining refers to applying forward chaining in small steps or incremental updates, rather than attempting to derive all facts at once. This is particularly useful in dynamic environments where the knowledge base or facts change over time.

Reasons why incremental forward chaining is needed:

1. Dynamic Knowledge Base:
    - In real-world scenarios, facts might change over time (e.g., someone's age, income, or the availability of new information). Incremental forward chaining allows the system to update the conclusions based on the changes in the facts without recalculating everything from scratch.
2. Efficiency:
    - Instead of reevaluating all rules every time new facts are introduced, the system only re-applies relevant rules that were affected by the changes. This results in a more efficient and timely inference process.
3. Scalability:
    - In large systems with a lot of facts and rules, incremental forward chaining can ensure that only the parts of the knowledge base that have changed are recomputed. This makes the system more scalable and capable of handling large amounts of data and facts.
4. Real-Time Applications:
    - In environments such as real-time systems or interactive applications, facts may change continuously. Incremental forward chaining ensures that new information is processed quickly, keeping the system up to date with minimal computational overhead.

Example of Incremental Forward Chaining

Let’s say you have a weather forecasting system where facts are continuously updated (e.g., temperature, humidity, wind speed). Each time new data arrives, you don’t need to recompute the entire forecast from scratch.

- Initial facts: Temperature = 70°F, Humidity = 60%
- After receiving new data: Wind Speed = 10 mph

Now, the system only needs to apply rules that are relevant to the wind speed update, such as:

- Rule: IF wind speed > 8 mph THEN there's a chance of a storm.

By applying incremental forward chaining, the system only processes the changes due to the new wind speed, rather than recalculating the entire forecast.

b. Consider the following sentences:

- **John likes all kinds of food**
- **Applies are food**
- **Chicken is food**
- **Anything anyone eats and isn’t killed by is food**
- **Bill eats peanuts and is still alive**
- **Sue eats everything Bill eats**

i. **Translate these sentences into formulas in predicatelogic**

ii. **Prove that john likes peanuts using backward chaining**
iii. **Convert the formulas of a part into clause form**
iv. **Prove that john likes peanuts using resolution**

Answer:-

1. Translating the Sentences into Predicate Logic

1. **John likes all kinds of food**

$\forall x(Food(x) \rightarrow Likes(John,x))$

2. **Apples are food**

Food(Apple)

3. **Chicken is food**

Food(Chicken)

4. **Anything anyone eats and isn't killed by is food**

$\forall x \forall y((Eats(y,x) \wedge \neg KilledBy(x,y)) \rightarrow Food(x))$

5. **Bill eats peanuts and is still alive**

$Eats(Bill,Peanuts) \wedge \neg KilledBy(Peanuts,Bill)$

6. **Sue eats everything Bill eats**

$\forall x(Eats(Bill,x) \rightarrow Eats(Sue,x))$

---

2. Proving that John Likes Peanuts using Backward Chaining

Now, we will use backward chaining to prove that "John likes peanuts". We have to derive that Likes(John,Peanuts).

1. **From the formula for John likes all kinds of food, we know:**

$\forall x(Food(x) \rightarrow Likes(John,x))$

So, to prove Likes(John,Peanuts), we need to prove that Peanutsis food.

2. **From the formula for anything anyone eats and isn't killed by is food, we know:**

$\forall x \forall y((Eats(y,x) \wedge \neg KilledBy(x,y)) \rightarrow Food(x))$

This tells us that if someone eats peanuts and isn't killed by peanuts, then peanuts are food.

3. **From the formula for Bill eats peanuts and is still alive, we have:**

Eats(Bill,Peanuts)∧¬KilledBy(Peanuts,Bill)

Bill eats peanuts and is still alive, which satisfies the condition that Bill eats peanuts and isn't killed by them. Thus, by the previous rule, we conclude:

Food(Peanuts)

4. **Now, using the first formula, since we have established Food(Peanuts), we can conclude:**

Likes(John,Peanuts)

Thus, John likes peanuts.

---

3. Converting the Formulas into Clause Form

Clause form is a way to represent logical formulas using a set of disjunctions (ORs) of literals, which are variables or negations of variables. Let's convert the given formulas into clausal form:

1. John likes all kinds of food

∀x(Food(x)→Likes(John,x))≡∀x(¬Food(x)∨Likes(John,x))

In clausal form:

{¬Food(x),Likes(John,x)}

2. Apples are food

Food(Apple)

In clausal form:

{Food(Apple)}

3. **Chicken is food**

Food(Chicken)

In clausal form:

{Food(Chicken)}

4. **Anything anyone eats and isn't killed by is food**

$\forall x \forall y((Eats(y,x) \land \neg KilledBy(x,y)) \rightarrow Food(x)) \equiv \forall x \forall y(\neg Eats(y,x) \lor KilledBy(x,y) \lor Food(x))$

In clausal form:

{¬Eats(y,x),KilledBy(x,y),Food(x)}

5. **Bill eats peanuts and is still alive**

$Eats(Bill,Peanuts) \land \neg KilledBy(Peanuts,Bill)$

In clausal form:

{Eats(Bill,Peanuts)}

{¬KilledBy(Peanuts,Bill)}

6. **Sue eats everything Bill eats**

$\forall x(Eats(Bill,x) \rightarrow Eats(Sue,x)) \equiv \forall x(\neg Eats(Bill,x) \lor Eats(Sue,x))$

In clausal form:

{¬Eats(Bill,x),Eats(Sue,x)}

---

4. Proving that John Likes Peanuts using Resolution

To prove that John likes peanuts using resolution, we need to combine the clauses we have derived and attempt to derive Likes(John,Peanuts).

We start with the following clauses:

1. {¬Food(x),Likes(John,x)}
2. {Food(Apple)}
3. {Food(Chicken)}
4. {¬Eats(y,x),KilledBy(x,y),Food(x)}

5. {Eats(Bill,Peanuts)}
6. {¬KilledBy(Peanuts,Bill)}
7. {¬Eats(Bill,x),Eats(Sue,x)}

Now, let's resolve:

- From the formula Eats(Bill,Peanuts) and the formula ¬KilledBy(Peanuts,Bill), we know that Peanutsis food (from the condition "anything anyone eats and isn’t killed by is food").
- Using this, we can resolve the clause {¬Food(x),Likes(John,x)} with Food(Peanuts), which gives us:

Likes(John,Peanuts)

Thus, we have proven that John likes peanuts using resolution.

C) What is backward chaining? Explain with an example.

Backward Chaining is a reasoning technique used in artificial intelligence (AI), particularly in expert systems and rule-based systems. It is a form of *goal-driven* search, where the system starts with a goal (or hypothesis) and works backward to find the facts or conditions that support that goal.

In backward chaining, the system starts with the conclusion or goal and tries to determine which rules or conditions need to be satisfied in order to achieve that goal. It recursively checks if the required conditions (premises of rules) hold, until it either reaches known facts or fails.

Steps in Backward Chaining:

1. Start with a goal: The system begins by focusing on a specific goal it wants to achieve.
2. Look for rules that can support the goal: It searches for rules that can conclude the goal.

3. Check if conditions are met: For each rule, it checks if the conditions (premises) are true. If not, the system recursively attempts to prove the conditions.
4. Repeat until facts are found or goal is disproved: The process continues until the system either finds enough facts to prove the goal or exhausts all possibilities.

Example in AI (Expert System for Diagnosing Diseases):

Suppose we have an expert system designed to diagnose a disease based on symptoms. The rules are set up like this:

1. Rule 1: If a patient has a *fever* and a *cough*, then they might have the *flu*.
2. Rule 2: If a patient has a *fever* and a *rash*, then they might have *measles*.
3. Rule 3: If a patient has a *cough* and *sore throat*, then they might have a *cold*.

Let's say we are tasked with diagnosing a patient who has a fever and a cough.

Goal: Determine what disease the patient has (flu, measles, cold, etc.).

Backward Chaining Process:

1. Start with the goal: The goal is to diagnose the disease. So, we check which diseases match the symptoms.
2. Check Rule 1: The system asks, "Can the flu be the disease?" It checks the conditions: "Does the patient have a fever and a cough?" Yes, both symptoms are present, so the system concludes the patient might have the flu.
3. Check other rules: The system will then check other possibilities, like Rule 2 and Rule 3, for measles and cold. However, it can stop after confirming flu as a likely diagnosis.

In this example, backward chaining allows the system to focus only on the relevant rules (like fever and cough) to backtrack to a potential disease (flu) without exploring irrelevant rules.

Unit-4

1. **Define state-space search in planning.**

State-space search is a method used in AI planning where the problem is represented as a graph, with nodes as states and edges as actions, to find a path from the initial state to the goal state.

2. **What is a planning graph?**

A planning graph is a data structure used in **GraphPlan** to represent the states and actions of a planning problem efficiently.

3. **What is hierarchical planning?**

Hierarchical planning is a method where a complex problem is broken down into smaller sub-problems using **hierarchical task networks (HTN)**.

4. **What is the primary difference between forward and backward state-space search?**

Forward search starts from the **initial state** and moves toward the goal, while backward search starts from the **goal state** and works backward to the initial state.

5. **What does STRIPS stand for?**

STanford Research Institute Problem Solver – a formal language for defining planning problems.

6. **What is classical planning?**

Classical planning is an approach in artificial intelligence where an agent finds a sequence of actions that transition from an initial state to a goal state in a well-defined environment. Classical planning in AI is a foundational field that traverses the maze of complications across multiple domains. The foundation of everything from robotics to manufacturing, logistics to space exploration is classical planning, which offers an organized method for accomplishing objectives

7. **Perform an analysis of different planning approaches in terms of complexity and effectiveness.**
    - **State-space search:** Effective but computationally expensive.
    - **Graph-based planning:** Balances efficiency and expressiveness.
    - **Hierarchical planning:** Reduces complexity by decomposing tasks.
    - **SAT-based planning:** Converts planning into logical constraints for efficient solving.
8. **Explain how planning graphs work in classical planning.**
    - A planning graph is a **layered structure** representing actions and states.

- It consists of:
    1. **State levels:** Contain all possible states.
    2. **Action levels:** Contain all applicable actions.
    3. **Mutex constraints:** Indicate conflicting actions.

9. **Explain STRIPS representation and its role in automated planning.**
    - **STRIPS (Stanford Research Institute Problem Solver)** represents planning problems using:
        1. **Initial State:** Description of the starting conditions.
        2. **Actions:** Defined with **preconditions** (what must be true before the action) and **effects** (changes caused by the action).
        3. **Goal State:** Desired final condition
10. **Describe the GraphPlan Algorithm and its significance in planning.**
    - **GraphPlan Algorithm:** Constructs a **planning graph** to determine a sequence of actions efficiently.
    - **Steps:**

        1. Create an initial state level.
        2. Expand action and state levels iteratively.
        3. Identify mutex (mutual exclusion) constraints.
        4. Extract a solution if possible.

Significance: More efficient than traditional state-space search.

5-Mark Questions with Answers

11. **Write any four Key points about planning in artificial intelligence ?**
    planning in

•Uncertainty:

Unlike traditional planning where actions have predictable outcomes, in nondeterministic domains, actions can have multiple possible outcomes, depending on factors outside the agent's complete control.

•Partial Observability:

Often, an agent may not have complete information about the environment state, requiring sensing actions to gather more information before deciding on the next action.

•Conditional Planning (Contingency Planning):

A primary approach to handle nondeterminism, where the plan includes branches based on different possible outcomes, allowing the agent to adapt its actions depending on what **happens in the environment.**

•Replanning:

Since the environment may change unexpectedly, the agent may need to re-evaluate its plan based on new information received during execution.

12. **Explain the concept of classical planning with an example.**
    - Classical planning involves searching for a sequence of actions that transform an initial state into a goal state in a well-defined environment.
    - **Example:** Suppose a robot in a warehouse needs to move a package from point A to B. The classical planner determines the best sequence of actions (e.g., move forward, pick up package, move to B, drop package) to achieve this goal.
13. **Describe the forward and backward state-space search algorithms with a comparison.**
    - **Forward Search:**
        1. Begins at the **initial state** and expands toward the goal.
        2. Uses breadth-first, depth-first, or heuristic search.
    - **Backward Search:**
        1. Starts from the **goal state** and works backward to reach the initial state.
        2. Efficient when fewer goal states are defined.
    - **Comparison:**

Criteria	Forward Search	Backward Search
Direction	Initial → Goal	Goal → Initial
Efficiency	Inefficient for large states	More efficient in some cases
Heuristic Use	Can use heuristics	Requires goal-based heuristics

    - .
14. **Discuss various classical planning approaches with their advantages and limitations.**
    - **State-Space Search:** Uses search algorithms (e.g., BFS, DFS) to explore all possible actions.
        1. Pro: Finds optimal solutions.
        2. Con:High computational cost.
    - **GraphPlan:** Uses planning graphs to improve efficiency.
        1. Pro: More efficient than state-space search.
        2. Con:Limited expressiveness.
    - **SAT Planning:** Transforms planning problems into satisfiability problems.
        1. Pro: Powerful for complex problems.
        2. Con: Hard to implement.
15. **How does hierarchical planning improve efficiency in planning?**
    - It breaks down problems into **subtasks**, reducing computational complexity.
    - Example: A robot delivering a package first plans **"move to pickup location"**, then **"pick up package"**, then **"move to delivery location"**, and finally **"drop package."**

16. Explain the Comparison of Classical Planning and Hierarchical Planning

Planning is a fundamental aspect of Artificial Intelligence (AI), used in problem-solving and decision-making processes. Two important approaches in planning are **Classical Planning** and **Hierarchical Planning**. Both methods have distinct characteristics, advantages, and applications.

Classical Planning

Definition

Classical Planning is an AI approach where an agent searches for a **sequence of actions** that transforms an **initial state** into a **goal state** in a well-defined environment. It assumes:

- The world is **fully observable** (the agent knows everything).
- Actions have **deterministic effects** (they always produce the expected result).
- The environment is **static** (it does not change unless the agent acts).

Key Features

- Uses **search algorithms** (e.g., Breadth-First Search, Depth-First Search, A*).
- Typically uses **STRIPS (Stanford Research Institute Problem Solver)** or **PDDL (Planning Domain Definition Language)**.
- The planner finds an optimal **sequence of actions** to achieve the goal.

Example of Classical Planning

Robot Navigation Problem

Consider a **robot** in a warehouse that needs to move a box from location A to location B.

- **Initial State:** The robot is at A, and the box is at A.
- **Actions:** The robot can "Pick Up Box," "Move to B," and "Drop Box."
- **Goal State:** The robot is at B, and the box is also at B.

Using **state-space search**, the planner finds the optimal sequence:

1. Pick Up Box
2. Move to B
3. Drop Box

This is a **sequential** approach where each action follows a predefined plan.

2. Hierarchical Planning

Definition

Hierarchical Planning, also known as **Hierarchical Task Network (HTN) Planning**, is an approach where a complex problem is broken into **smaller subproblems** or tasks, which are solved separately. Instead of finding a direct sequence of actions, it **decomposes** the problem into multiple levels of abstraction.

Key Features

- Uses **task decomposition** (breaking down large tasks into smaller ones).
- More suitable for **real-world complex problems**.
- Actions are organized into a hierarchy of **high-level and low-level tasks**.

Example of Hierarchical Planning

Travel Planning Problem

Suppose you want to travel from **City A to City B**. Instead of searching for an exact sequence of actions, the hierarchical planner breaks the problem into subtasks:

High-Level Plan:

1. **Plan Transportation**
    - Choose flight/train/bus
2. **Book Tickets**
    - Select date, time, and class
3. **Pack Luggage**
    - Clothes, documents, essentials
4. **Travel to the Airport**
    - Take a cab, drive, or public transport

Low-Level Plan for "Plan Transportation":

1. Search for available flights
2. Compare prices and timings
3. Select and book a ticket

Instead of solving the entire problem at once, **Hierarchical Planning** first decides on **what needs to be done** (high-level plan) and then **how it will be done** (low-level actions).

Feature	Classical Planning	Hierarchical Planning
Approach	Uses **state-space search**	Uses **task decomposition**
Complexity	Computationally expensive for large problems	More efficient for complex problems
Flexibility	Less flexible, requires precise domain	More flexible, can adapt to different

Feature	Classical Planning	Hierarchical Planning
	knowledge	scenarios
Solution Type	Finds a **direct action sequence** to reach the goal	Breaks problem into **subtasks** before solving
Use Case	Works well for small, well-defined problems	Suitable for **large, real-world problems**
Example	A **robot navigating a warehouse**	A **travel planning system**

---

17) What is the knowledge representation on Internet Shopping world?

Artificial intelligence (AI) is changing the online shopping experience by making it more personalized, efficient, and convenient. AI-powered tools can analyze customer data to suggest products, answer questions, and predict demand.

How AI is used in online shopping

- **Personalized recommendations**

  AI uses algorithms to analyze user behavior and preferences to suggest products.

- **Virtual assistants**

  AI-powered chatbots and virtual assistants can answer questions and provide guidance.

- **Visual search**

  AI-powered image recognition allows users to upload images and find similar products.

- **Predictive analytics**

  AI can analyze consumer data to predict future purchasing patterns and demand.

- **Fraud detection**

  AI-powered systems can identify suspicious activities to safeguard transactions.

- **Inventory management**

  AI can help businesses forecast demand and optimize their supply chains.

- **Demand forecasting**

  AI can track trends on social media and competitor movements to determine buying patterns.

Benefits of AI in online shopping

- AI can improve customer service and satisfaction
- AI can help businesses create superior client experiences
- AI can help businesses respond to dynamic market conditions

Potential drawbacks of AI in online shopping Concerns about data privacy and security, Risk of job displacement, and Potential for algorithms to be biased.

18. **Explain Planning and Acting** Nondeterministic Domains in artificial intelligences

Planning and Acting in Nondeterministic Domains" in AI refers to the challenge of creating plans for an agent to execute in environments where the outcomes of actions are not guaranteed, meaning there is uncertainty about what state the environment will be in after an action is taken, requiring the agent to adapt its plan based on new information as it becomes available; this often involves techniques like conditional planning and replanning to handle potential variations in the environment.

Key points about planning in nondeterministic domains:

Uncertainty:

Unlike traditional planning where actions have predictable outcomes, in nondeterministic domains, actions can have multiple possible outcomes, depending on factors outside the agent's complete control.

Partial Observability:

Often, an agent may not have complete information about the environment state, requiring sensing actions to gather more information before deciding on the next action.

Conditional Planning (Contingency Planning):

A primary approach to handle nondeterminism, where the plan includes branches based on different possible outcomes, allowing the agent to adapt its actions depending on what **happens in the environment.**

Replanning:

Since the environment may change unexpectedly, the agent may need to re-evaluate its plan based on new information received during execution.

Example scenarios:

Robot navigation:

A robot navigating a room with obstacles that might move or appear unexpectedly needs to plan a path that can adapt to changing conditions.

Medical diagnosis:

A doctor may need to consider various possible diagnoses based on patient symptoms and perform additional tests to gain more information before deciding on a treatment plan.

Game playing:

In a game with elements of chance, a player needs to plan moves that consider the potential outcomes of their actions and the opponent's possible responses.

Challenges in nondeterministic planning:

Complexity:

The search space for possible outcomes can become very large, making it computationally difficult to find an optimal plan.

Bounded vs. Unbounded Uncertainty:

Some situations have a limited set of possible outcomes (bounded uncertainty), while others have a vast or unknown set of potential outcomes (unbounded uncertainty).

UNIT-5

1-Mark Questions:

Q 1.a) What does Bayes' Rule help compute in probabilistic reasoning?

Ans: Bayes' Rule helps compute the posterior probability of an event based on prior knowledge and new evidence. It updates our beliefs by incorporating observed data, making it essential for probabilistic reasoning and decision-making under uncertainty.

Q 1.B) Define a Bayesian Network.

Ans: A Bayesian Network is a graphical probabilistic model that represents variables and their conditional dependencies using a Directed Acyclic Graph (DAG). Each node represents a random variable, and directed edges indicate probabilistic influences between them.

Q 1.C) What is the primary goal of approximate inference in Bayesian Networks?

Ans: The primary goal of approximate inference in Bayesian Networks is to estimate probability distributions when exact inference is infeasible due to computational complexity. It allows for making fast, approximate decisions using techniques like Monte Carlo sampling and variational methods.

Q 1.D) What is the primary difference between deterministic and probabilistic reasoning?

Ans: The main difference between deterministic reasoning and probabilistic reasoning is that deterministic reasoning provides definite conclusions based on given premises, whereas probabilistic reasoning accounts for uncertainty and expresses outcomes in terms of likelihoods.

Q 1.E) What does a conditional probability represent?

Ans: A conditional probability represents the probability of an event occurring given that another event has already occurred. It helps in modeling dependencies between events and is a fundamental concept in Bayesian Networks.

Q 2 A) Name one real-world application of Bayesian Networks.

Ans: Medical diagnosis is a real-world application of Bayesian Networks. They are used to predict diseases based on symptoms by combining prior probabilities of diseases with observed patient data to provide more accurate diagnoses.

Q 2.B) What is the role of prior probability in Bayes' Theorem?

Ans: The prior probability in Bayes' Theorem represents the initial belief about an event before considering new evidence. It serves as the starting point for probability updates when new data is observed.

Q 2.C) Define approximate inference in the context of Bayesian Networks.

Ans: Approximate inference in Bayesian Networks refers to methods used to estimate probabilities when exact inference is computationally expensive. Techniques like Monte Carlo simulations, belief propagation, and variational approximations are used to make inference more efficient.

Q 2.D) What is the main challenge of exact inference in Bayesian Networks?

Ans: The main challenge of exact inference in Bayesian Networks is that it becomes computationally intractable for large networks due to the exponential growth of possible states. Algorithms like Variable Elimination and Junction Tree can be used but are still inefficient for complex models.

Q 2.E) How does relational probability differ from traditional probability?

Ans: Relational probability extends traditional probability by incorporating **relationships between objects and entities**, rather than treating variables as independent. It is used in **Relational Bayesian Networks** and **Probabilistic Graphical Models** to capture dependencies in structured data, such as social networks and knowledge graphs.

5-Mark Questions:

Q 3.A) Explain the concept of probabilistic reasoning and its importance in decision-making under uncertainty.

Ans: Probabilistic reasoning is a mathematical framework used to model uncertainty and make informed decisions based on available evidence. It relies on probability theory to quantify uncertainty and update beliefs when new data is obtained.

Key Characteristics of Probabilistic Reasoning:
Uses probability distributions to model uncertain events.
Accounts for partial, noisy, or incomplete information.
Updates beliefs using Bayes' Theorem when new evidence is available.
Allows for optimal decision-making despite uncertainty.
2. Importance of Probabilistic Reasoning in Decision-Making

Uncertainty is present in almost every real-world scenario. Probabilistic reasoning provides a structured way to handle uncertainty and make rational choices. Below are some key reasons why it is essential:

(a) Managing Uncertainty
Many domains involve incomplete or noisy data (e.g., medical diagnosis, finance, weather forecasting).
Probabilistic models provide a way to reason logically despite uncertainty.
(b) Decision Support Systems
Helps in making rational decisions by computing expected outcomes.
Used in risk assessment, autonomous systems, and AI-driven recommendations.
(c) Incorporating Prior Knowledge
Bayesian inference allows for the integration of prior knowledge with new evidence.
Example: In medical diagnosis, if a rare disease has a 1% prevalence, a positive test result does not guarantee the patient has the disease.
(d) Efficient Handling of Large-Scale Problems
Probabilistic reasoning scales well to complex domains (e.g., self-driving cars, fraud detection).
Models like Bayesian Networks (BNs) and Hidden Markov Models (HMMs) reduce computational complexity.
Examples of Probabilistic Reasoning in Decision-Making
1. Medical Diagnosis
2. Weather Prediction
3. Financial Risk Assessment
4. Self Driving Cars
5. Robotics

Q 3.B) Derive Bayes' Rule and explain its significance with an example.

Ans:Bayes' Theorem is a mathematical formula that helps determine the conditional probability of an event based on prior knowledge and new evidence.

It adjusts probabilities when new information comes in and helps make better decisions in uncertain situations.

For any two events A and B, then the formula for the Bayes theorem is given by:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Formula for the Bayes theorem

Where,

- **P(A)** and **P(B)** are the probabilities of events A and B also P(B) is never equal to zero,
- **P(A|B)** is the probability of event A when event B happens,
- **P(B|A)** is the probability of event B when A happens.

Bayes Theorem Statement

Bayes Theorem for n set of events is defined as,

Let $E_1, E_2, \ldots, E_n$ be a set of events associated with a sample space S, where all the events $E_1, E_2, \ldots, E_n$ have nonzero probability of occurrence and they form a partition of S. Let A be any event associated with S, then according to Bayes theorem,

$$P(E_i \mid A) = \frac{P(E_i)P(A \mid E_i)}{\sum_{k=1}^{n} P(E_k)P(A|E_k)}$$

for any k = 1, 2, 3, ..., n

Bayes Theorem Proof

According to the conditional probability formula,

$$P(E_i \mid A) = \frac{P(E_i \cap A)}{P(A)} \ldots (1)$$

Using the multiplication rule of probability,

$$P(E_i \cap A) = P(E_i)P(A \mid E_i) \ldots (2)$$

Using total probability theorem,

The following terminologies are also used when the Bayes theorem is applied:

Hypotheses: The events $E_1, E_2, \ldots E_n$ is called the hypotheses

Priori Probability: The probability $P(E_i)$ is considered as the priori probability of hypothesis E_i

Posteriori Probability: The probability $P(E_i|A)$ is considered as the posteriori probability of hypothesis E_i

Bayes' theorem is also called the formula for the probability of "causes". Since the E_i's are a partition of the sample space S, one and only one of the events E_i occurs (i.e. one of the events E_i must occur and the only one can occur). Hence, the above formula gives us the probability of a particular E_i (i.e. a "Cause"), given that the event A has occurred.

Bayes Theorem can be derived for events and random variables separately using the definition of conditional probability and density.

From the definition of conditional probability, Bayes theorem can be derived for events as given below:

$P(A|B) = P(A \cap B)/ P(B)$, where $P(B) \neq 0$

$P(B|A) = P(B \cap A)/ P(A)$, where $P(A) \neq 0$

Here, the joint probability $P(A \cap B)$ of both events A and B being true such that,

$P(B \cap A) = P(A \cap B)$

$P(A \cap B) = P(A \mid B)\, P(B) = P(B \mid A)\, P(A)$

$P(A|B) = [P(B|A)\, P(A)]/ P(B)$, where $P(B) \neq 0$

Similarly, from the definition of conditional density, Bayes theorem can be derived for two continuous random variables namely X and Y as given below:

$$f_{X|Y=y}(x) = \frac{f_{X,Y(x,y)}}{f_Y(y)}$$
$$f_{Y|X=x}(y) = \frac{f_{X,Y(x,y)}}{f_X(x)}$$

Therefore,

$$f_{X|Y=y}(x) = \frac{f_{Y|X=x}(y) f_X(x)}{f_Y(y)}$$

Bayes Theorem Applications

Bayesian inference is very important and has found application in various activities, including medicine, science, philosophy, engineering, sports, law, etc., and Bayesian inference is directly derived from Bayes theorem.

Some of the Key Applications are:

- **Medical Testing** → Finding the real probability of having a disease after a positive test.
- **Spam Filters** → Checking if an email is spam based on keywords.
- **Weather Prediction** → Updating the chance of rain based on new data.
- **AI & Machine Learning** → Used in **Naïve Bayes classifiers** to predict outcomes.

Q 3.C) How is knowledge represented in an uncertain domain using probabilistic methods?

Ans: An uncertain domain in artificial intelligence (AI) refers to a field or environment where the information available is incomplete, ambiguous, noisy, or inherently unpredictable. Unlike deterministic domains where outcomes can be predicted with certainty given the inputs, uncertain domains require AI systems to handle and reason about uncertainty in a structured manner.

Characteristics of Uncertain Domains

Incomplete Information: The system does not have access to all the data required to make a fully informed decision.

Ambiguity: Information might be unclear or open to multiple interpretations.

Noise: Data might be corrupted or imprecise due to measurement errors or external factors.

Stochastic Processes: The environment might involve random processes or events.

Importance of Handling Uncertainty

In many real-world applications, AI systems must operate effectively despite uncertainty. Accurately representing and reasoning about uncertain information is crucial for making reliable predictions and decisions. Handling uncertainty enables AI systems to:

- Make informed decisions based on probabilistic reasoning.
- Adapt to new information and changing environments.
- Provide robust and reliable performance in complex scenarios.

  Representing Knowledge in an Uncertain Domain

  In real-world applications, AI systems frequently encounter incomplete, ambiguous, or noisy information. Traditional deterministic approaches fall short in such scenarios, necessitating the use of probabilistic and fuzzy methods to handle uncertainty effectively. These methods enable AI systems to make informed decisions, predict outcomes, and adapt to changing environments.

  1. Probabilistic Reasoning

  Probabilistic reasoning involves representing knowledge using probability theory to manage uncertainty. This approach is widely used in AI for tasks such as diagnosis, prediction, and decision-making under uncertainty.

  2.Bayesian Networks

  Bayesian networks (BNs) are graphical models that represent the probabilistic relationships among a set of variables. Each node in a BN represents a variable, and the edges represent conditional dependencies. BNs allow for efficient computation of posterior probabilities given observed evidence.

  Example: A Bayesian network for a medical diagnosis system might include nodes for symptoms (fever, cough) and diseases (flu, pneumonia), with edges indicating the probabilistic dependencies between them.

  2. Hidden Markov Models

Hidden Markov Models (HMMs) are used to model time series data where the system being modeled is assumed to be a Markov process with hidden states. HMMs are widely used in speech recognition, bioinformatics, and other sequential data applications.

Example: In speech recognition, the observed sound waves are modeled as emissions from hidden phonetic states, allowing the system to decode spoken language.

3. Markov Decision Processes

Markov Decision Processes (MDPs) provide a framework for modeling decision-making in environments with stochastic dynamics. MDPs consist of states, actions, transition probabilities, and rewards, enabling the computation of optimal policies for decision-making.

Example: An autonomous robot navigating a grid world can use an MDP to determine the optimal path to its destination while accounting for uncertain movements and rewards.

4. Fuzzy Logic

Fuzzy logic is an approach to reasoning that deals with approximate rather than fixed and exact values. Unlike traditional binary logic, fuzzy logic variables can have a truth value that ranges between 0 and 1, representing the degree of truth.

Fuzzy Sets and Membership Functions

Fuzzy sets allow for the representation of concepts with vague boundaries. Each element in a fuzzy set has a membership value indicating its degree of belonging to the set.

Example: In a temperature control system, the concept of "warm" can be represented as a fuzzy set with a membership function assigning values between 0 (not warm) and 1 (completely warm) to different temperatures.

4.1Fuzzy Rules and Inference

Fuzzy rules define the relationships between fuzzy variables using if-then statements. Fuzzy inference systems apply these rules to input data to derive conclusions.

Example: A fuzzy rule for a temperature control system might be: "If the temperature is high, then reduce the heater power."

5. Dempster-Shafer Theory

The Dempster-Shafer theory, also known as evidence theory, is a mathematical framework for modeling uncertainty without the need for precise probabilities. It allows for the combination of evidence from different sources to calculate the degree of belief (or plausibility) for various hypotheses.
Example: In an expert system for fault diagnosis, evidence from different sensors can be combined using Dempster-Shafer theory to assess the likelihood of different fault conditions.

6. Belief Networks
Belief networks extend Bayesian networks by allowing for the representation of uncertainty in the strength of the dependencies between variables. They provide a way to handle imprecise and incomplete knowledge.

Example: A belief network for an intelligent tutoring system might include nodes for student knowledge, engagement, and performance, with edges representing uncertain dependencies between these factors.

7. Case-Based Reasoning
Case-based reasoning (CBR) is an approach where past cases (experiences) are used to solve new problems. In uncertain domains, CBR can be combined with probabilistic methods to estimate the likelihood of various outcomes based on similar past cases.

Example: A customer support system can use CBR to suggest solutions based on previous similar customer queries, adjusting recommendations based on the uncertainty of the current context.

Applications of Uncertain Knowledge Representation

- Medical Diagnosis: Probabilistic models like Bayesian networks are used to diagnose diseases based on symptoms and medical history.
- Autonomous Vehicles: Fuzzy logic and MDPs help autonomous vehicles navigate and make decisions in dynamic environments.
- Natural Language Processing: HMMs and probabilistic context-free grammars are used for tasks like speech recognition and language modeling.
- Robotics: Robots use probabilistic reasoning to handle sensor noise and uncertain environments for navigation and manipulation tasks.
- Finance: Probabilistic models are employed for risk assessment, fraud detection, and market prediction.

Q 3.D) Describe the semantics of Bayesian Networks with an example.

Ans: Semantics of Bayesian Networks

Bayesian Networks (BNs) are probabilistic graphical models that represent a set of random variables and their conditional dependencies using a Directed Acyclic Graph (DAG). The semantics of a Bayesian Network describe how the joint probability distribution (JPD) of a system can be decomposed into a product of conditional probabilities based on the independence assumptions encoded in the network structure.

Key Aspects of Bayesian Network Semantics

1. **Graphical Representation:**
   - A **BN consists of nodes and directed edges**.
   - **Nodes** represent random variables.
   - **Edges** represent direct probabilistic dependencies between variables.
   - The absence of an edge indicates **conditional independence** between variables.
2. **Local Conditional Probability Distributions (CPDs):**
   - Each node is associated with a **Conditional Probability Table (CPT)** if discrete or a **probability density function (PDF)** if continuous.
   - The CPT defines how the probability of a node depends on its parent nodes.
3. **Factorization of Joint Probability Distribution (JPD):**
   - The **chain rule of probability** allows the joint probability of all variables to be expressed as the product of conditional probabilities:
   - $$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i | Parents(X_i))$$
   - This factorization significantly **reduces computational complexity** compared to a full JPD.

     Example: give any one sample data and calculated JPD.

Q 3.E) What are the advantages of using Bayesian Networks for reasoning under uncertainty?

Ans: Bayesian networks provide a more efficient way to represent and compute joint probabilities by exploiting the conditional independence properties of the variables in the network.

In Bayesian networks, you represent the joint probability distribution by decomposing it into a product of conditional probabilities. This representation takes advantage of the conditional independence assumptions implied by the network structure, which often leads to more efficient computations and a more compact representation of the joint distribution.

While this is one of the key benefits, it is not the only benefit. Others include:

Modularity and interpretability: Bayesian networks represent complex relationships among variables in a graphical form, making it easy to visualize and understand the

dependencies and independencies between variables. This modularity aids in knowledge representation, making it easy to update and modify the network structure as new information becomes available.

Incorporation of domain knowledge: Bayesian networks can incorporate expert knowledge in the form of prior probabilities and network structure. This allows for the inclusion of valuable information that might not be readily available in the data itself.

Inference and reasoning: Bayesian networks facilitate probabilistic reasoning under uncertainty, allowing you to perform various types of inferences such as causal reasoning, diagnostic reasoning, and predictive reasoning. They can be used to answer queries about the probabilities of events given new evidence or observations.

Handling missing data: Bayesian networks can naturally deal with incomplete or missing data. They can be used to predict the values of missing variables based on the observed data and the underlying probabilistic relationships between variables.

Q 4.A) How does relational and first-order probability extend Bayesian Networks?

Ans: Bayesian Networks (BNs) are powerful tools for probabilistic reasoning, but they have limitations when dealing with complex, structured domains involving multiple entities and relationships. Relational and First-Order Probability extend BNs to address these challenges by incorporating objects, relations, and general rules.

Limitations of Traditional Bayesian Networks
Fixed Variables: BNs require explicitly defining all variables, making them hard to scale for large, structured problems.
Lack of Generalization: BNs cannot generalize across similar entities, requiring a separate model for each scenario.
No Representation of Relationships: BNs focus on direct dependencies but lack the ability to represent relational structures.
To overcome these limitations, Relational Probability Models (RPMs) and First-Order Probabilistic Models extend Bayesian Networks.

Applications of RPMs:

Social Network Analysis (e.g., predicting influence in social media).
Fraud Detection (e.g., identifying fraudulent transactions based on relationships).
Healthcare (e.g., modeling disease spread among patients).

First-Order Probability:

Combines First-Order Logic (FOL) with probability theory.
Unlike BNs, which use propositional variables, first-order models use quantifiers and predicates to handle general rules.
Generalizes Bayesian Networks by applying probabilistic rules to classes of objects instead of specific instances.

Feature	Traditional Bayesian Networks	Relational Probability Models	First-Order Probabilistic Models

Variables	Fixed set of random variables	Entities with relationships	Generalized rules with quantifiers
Scalability	Hard to scale for large datasets	Scales across structured data	Highly scalable with logical rules
Generalization	No generalization	Generalizes across entities	Generalizes using logical predicates
Example	Disease diagnosis for a single patient	Disease spread in a population	General disease patterns in humans

Q 4 .B) Explain how probabilistic reasoning is applied in real-world scenarios, such as medical diagnosis or artificial intelligence.

Ans: Probabilistic reasoning is widely used in real-world applications where uncertainty exists. By applying probability theory, systems can make informed decisions even with incomplete, noisy, or ambiguous data. Some key areas where probabilistic reasoning is applied include medical diagnosis, artificial intelligence, finance, and robotics.

Use case -Medical Diagnostics:

- Doctors often deal with **uncertain symptoms** and must infer the most likely disease based on **patient data, test results, and prior knowledge**.
- **Bayesian Networks** help compute the probability of diseases given symptoms.
- Probabilistic models can also improve **early disease detection** and suggest further tests.

Example: Diagnosing Pneumonia

A doctor wants to determine whether a patient has pneumonia (D), given that they have a **cough** (S).

Using **Bayes' Theorem**:

$$P(D|S) = \frac{P(S|D)P(D)}{P(S)}$$

Where:

- $P(D)$ is the **prior probability** (base rate of pneumonia cases).
- $P(S|D)$ is the **likelihood** (probability of a cough if pneumonia is present).
- $P(S)$ is the **marginal probability** of having a cough.

Real-World Use Cases

- **AI-powered diagnosis systems** (IBM Watson, Google's DeepMind).
- **Predicting disease outbreaks** (e.g., COVID-19 spread models).
- **Personalized medicine**, where treatments are suggested based on **patient history and genetic factors**.

2. Artificial Intelligence (AI)

- AI systems use probabilistic reasoning to **predict outcomes, make decisions, and update beliefs**.
- **Machine Learning models** rely on probability to classify data, detect patterns, and recommend actions.

Example: Speech Recognition

- AI assistants (Siri, Google Assistant) use **Hidden Markov Models (HMMs)** to understand speech.
- Probabilistic models estimate **word sequences** based on prior probabilities and context.
- Example:
    - Given the spoken input **"recognize speech"**, the system may also consider similar phrases like **"wreck a nice beach"**.
    - The system computes P(Words|Audio)P(Words | Audio)P(Words|Audio) to select the most likely interpretation.

Real-World Use Cases

- **Self-driving cars** predicting pedestrian movements and road conditions.
- **Recommendation systems** (Netflix, Amazon) predicting user preferences.
- **Fraud detection** in banking using probability-based anomaly detection.

Q 4 C) Calculate the P(F=True|Fe=True,C=True) using Bayesian Networks for the given probability information

1. Prior Probability Table

Flu (F)	Probability P(F)
True	0.1
False	0.9

2. Conditional Probability Table for Fever (Fe) given Flu (F)

Flu (F)	Fever (Fe = True)	Fever (Fe = False)
True	0.8	0.2
False	0.2	0.8

3. Conditional Probability Table for Cough (C) given Flu (F)

Flu (F)	Cough (C = True)	Cough (C = False)
True	0.7	0.3

False	0.1	0.9

Solution:

Step1: P(Fe=True,C=True/F=True)=0.7*0.8= 0.56

Step 2: P(F=True)=0.1

Step 3: P(Fe=True,C=True)=(0.1*0.7*0.8)+(0.2*0.1*0.9)=0.074

Step 3: Compute $P(F = True \mid Fe = True, C = True)$

$$P(F = True \mid Fe = True, C = True) = \frac{P(Fe = True, C = True \mid F = True)P(F = True)}{P(Fe = True, C = True)}$$

$$= \frac{0.056}{0.074}$$

$$\approx 0.756$$