# Ch-1

Q. What is "Artificial Intelligence and Artificial Intelligence Technique?

Ans → AI may be defined as the branch of computer science that is concerned with the automation of intelligent behaviour
→ AI is study of how to make computers do things which, at the moment, people do better

There are three most important AI techniques
1) Search
→ Provides a way of solving problems for which no direct approach is available
→ It also provides a framework into which any direct techniques that are available can be embedded

2) Use of Knowledge
→ Provides a way of solving complex problems by exploiting the structure of the objects that are involved

3) Abstraction
→ Provides a way of separating important features and variations from unimportant ones that would otherwise overwhelm any process

Q. Differentiate between strong AI and weak AI

Ans See Image

Here you go 👇 **clean tabular format**

| Basis | Weak AI (Narrow AI) | Strong AI (General AI) |
|---|---|---|
| **Definition** | AI designed for a specific task | AI with human-like intelligence and understanding |
| **Scope** | Limited to one domain | Can perform any intellectual task like humans |
| **Awareness** | No self-awareness or consciousness | Self-aware and conscious (theoretical) |
| **Learning ability** | Cannot generalize beyond its training | Can learn and apply knowledge across domains |
| **Examples** | Siri, Alexa, ChatGPT, face recognition | No real examples yet (only hypothetical/science fiction) |
| **Current status** | Exists today and widely used | Not yet achieved in reality |

If you need a **short 2-mark/3-mark answer**, tell me! ✅

Q.3 Define AI. What are Task Domains of AI [S-24, W-24]

Ans There are three task domains
1) Mundane Tasks
2) Formal Tasks
3) Expert tasks

1) Mundane tasks
→ Tasks that humans learn to do easily, often from birth or early childhood
→ Simple for humans, complex for computers because they require a lot of "common sense" knowledge about world

- Examples
  1) Perception
     → Computer Vision
     → Speech, Voice

  2) Natural Language Processing
     → Understanding
     → Language Generation
     → Language Translation

  3) Common Sense Reasoning

  4) Planning

  5) Robot Control

2) Formal Tasks
→ These are tasks based on formal rules and logic are like games or maths. Humans usually learn these later in life through formal education or practice
Examples
1) Games
→ Go
→ Chess (Deep Blue)
→ Checkers

2) Mathematics
→ Geometry
→ Logic
→ Integration & Differentiation

3) Theorem Proving

3) Expert Tasks
These are tasks that typically require specialized knowledge and training that only human expert possess. AI aims to replicate this expertise
→ AI research has been very successful in this area because expert tasks often rely on specific, deep knowledge rather than broad common sense, making them easier (in some ways) for computers to handle
o Examples
1) Engineering
→ Design
→ Fault finding
→ Manufacturing
→ Monitoring
2) Scientific Analysis
3) Financial Analysis
4) Medical Diagnosis

Q-4 State various applications of AI [2 times]

Ans
1) Gaming
2) NLP (Natural Language Processing)
3) Expert Systems
4) Computer Vision
5) Speech Recognition
6) Handwritting Recognition
7) Intelligent Robots
8) Data Mining
9) Scheduling
10) Optimization

1) See Image

Q-5 Explain Turing Test

Ans Purpose: The turing test, proposed by Alan Turing, it checks if a computer can act human-like that a person can't tell if they are talking to a computer or another human. It tests if a computer can "act humanly"

- **Perception:** Making sense of the world through senses.
  - **Computer Vision:** Enabling computers to "see" and interpret images or videos (like recognizing faces or objects).
  - **Speech/Voice Recognition:** Enabling computers to understand spoken language.
- **Natural Language Processing (NLP):** Enabling computers to understand, generate, or translate human language.
  - **Understanding:** Figuring out the meaning of text or speech.
  - **Language Generation:** Creating human-like text or speech.
  - **Language Translation:** Translating from one language to another (like Google Translate).
- **Common Sense Reasoning:** Making logical deductions about everyday situations (e.g., knowing that dropping something makes it fall).
- **Planning:** Figuring out a sequence of actions to achieve a goal.
- **Robot Control:** Controlling the movement and actions of robots in the physical world.

---

## 2. Formal Tasks

        - **Go, Chess (like Deep Blue), Checkers:** AI programs that can play these games, sometimes better than human champions.
    - **Mathematics:** Solving mathematical problems.
        - **Geometry:** Proving geometric theorems.
        - **Logic:** Applying logical rules.
        - **Integration and Differentiation:** Performing calculus operations.
    - **Theorem Proving:** Proving mathematical theorems based on axioms and rules.

3. **Expert Tasks**
   - These are tasks that typically require specialized knowledge and training that only human experts possess. AI aims to replicate this expertise.
   - AI research has been very successful in this area because expert tasks often rely on specific, deep knowledge rather than broad common sense, making them easier (in some ways) for computers to handle.
   - **Examples:**
     - **Engineering:**
       - **Design:** Assisting in designing complex systems.
       - **Fault Finding:** Diagnosing problems in machinery or systems.
       - **Manufacturing/Monitoring:** Overseeing and controlling manufacturing processes.
     - **Scientific Analysis:** Analyzing scientific data and forming hypotheses.
     - **Financial Analysis:** Analyzing financial data, predicting market trends.
     - **Medical Diagnosis:** Assisting doctors in diagnosing diseases based on symptoms and test results.

he major application areas of AI:

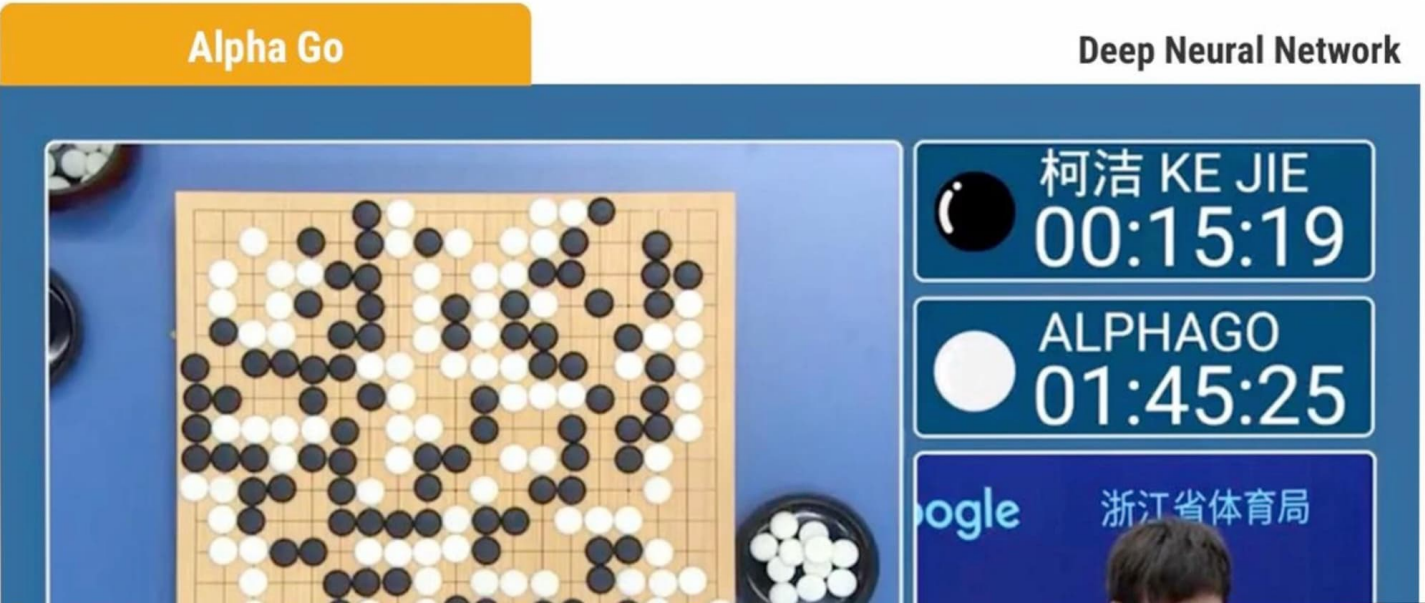
g:

AI plays a vital role in games like chess, poker, Go, and even video games.

It helps create non-player characters (NPCs) that act intelligently.

AI can analyze a huge number of possible moves in strategy games to find the best one.

*Example:* Deep Blue (chess program that beat the world champion) or AlphaGo (Go program).



- **Natural Language Processing (NLP):**
  - This allows computers to understand, interpret, and generate human language (like English).
  - It's used in spam filters (like in Gmail), machine translation (like Google Translate), chatbots, and analyzing text for sentiment (is it positive or negative?).
  - *Example:* Gmail automatically moving spam emails to the spam folder.

- **Expert Systems:**
  - These are computer programs designed to act like a human expert in a specific field.
  - They use a knowledge base (facts and rules) and an inference engine (reasoning mechanism) to solve complex problems or provide advice.
  - *Example:* Medical diagnosis systems that help doctors identify diseases based on symptoms, or IBM Watson which can answer complex questions.


- **Computer Vision:**
  - This field enables computers to "see" and interpret information from images or videos.
  - It's used in face detection and recognition (like on your phone or camera), self-driving cars to identify obstacles, medical image analysis, and security surveillance.
  - *Example:* Cameras detecting faces to focus correctly.

**Speech Recognition:**

- *This allows computers to understand spoken language and convert it into text or commands.*
- *It can handle different accents, background noise, and variations in human speech.*
- *Example: Virtual assistants like Siri, Alexa, or Google Assistant responding to your voice commands.*

- **Handwriting Recognition:**
  - Software that can read handwritten text (either on paper or a screen) and convert it into editable digital text.
  - *Example:* Apps that convert handwritten notes into typed text.
- **Intelligent Robots:**
  - Robots that can perform tasks given by humans, often in the physical world.
  - They use sensors (like cameras, temperature sensors, pressure sensors) to perceive their environment and processors to make decisions.
  - They can learn from mistakes and adapt to new situations.
  - *Example:* Robots used in manufacturing assembly lines, or home automation robots for cleaning.

- **Data Mining:**

---

  - AI techniques are used to discover patterns and insights from large amounts of data.
  - *Example:* Market Basket Analysis in supermarkets to see which items customers often buy together, leading to product recommendations.

- **Scheduling:**
  - AI helps in creating optimal schedules for complex tasks, like airline flight scheduling or resource allocation in projects.
  - *Example:* NASA using AI to schedule activities for space missions.

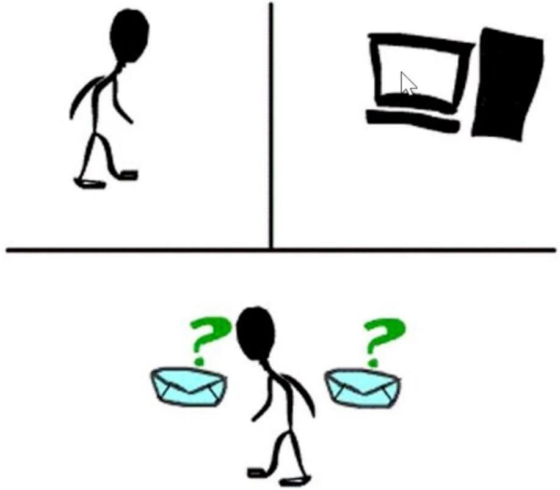
- **Optimization:**
  - Finding the best solution among many possibilities.
  - *Example:* Google Maps finding the shortest or fastest route between two places.

Q. Explain the "Turing test". **3M S23**

---

**Turing Test**

- **Purpose:** The Turing Test, proposed by Alan Turing, it checks if a computer can act so human-like that a person can't tell if they are talking to a computer or another human. It tests if a computer can "act humanly".



- **How it Works:**

- **How it Works:**
  - There are three participants: a human interrogator, a human, and a computer.
  - The interrogator is in a separate room and communicates with both the human and the computer through text messages (like a chat).
  - The interrogator asks questions to both the human and the computer.
  - The computer tries its best to give answers that make it seem like a human. The human answers truthfully.
  - The interrogator's job is to figure out which participant is the computer and which is the human.
- **Passing the Test:** If the interrogator cannot reliably tell the computer apart from the human, the computer is said to have passed the Turing Test. Passing suggests the machine shows intelligent behavior equivalent to, or indistinguishable from, that of a human.

---

Ch-2

Q-1 What is State-Space of a Problem?

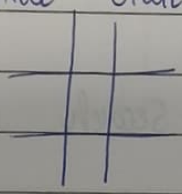
Ans State

→ A State is like a snapshot or a picture of the problem at a specific moment

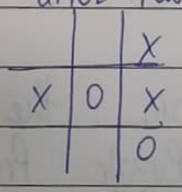
→ State is representation of problem elements at a given moment

Eg:

Initial State



State after few moves




State Space

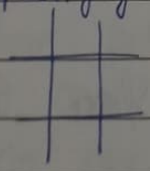
→ A state space is the set of all possible states reachable from initial state

→ It includes initial state also

Defining a Problem using State Space Search (Tick Toc)

1) Defining the State Space: We can use a 3x3 grid as shown above

2) Specifying the Initial State: The empty 3x3 board



3) Specifying the Goal State(s): Any board configuration where player whose turn it is (let's say X) has three of their marks in a row (horizontally vertically or diagonally) or any configuration where all squares are filled and

no has won (a draw)

Eg: Goal State (X wins)

| | | |
|---|---|---|
| X | O | O |
| | X | |
| | | X |

4) Specifying the Rules/Actions: Placing the current players mark (X or O) into any currently empty square on the board

Q-2 Explain the state space search with the use of 8 Puzzle Problem

Ans The 8-Puzzle Problem

→ It is simple game played on 3×3 grid
→ The grid contains 8 square tiles numbered 1 through 8, and one empty square (blank)
→ The only valid move is to slide a tile that is horizontally or vertically adjacent to the empty square into the empty square position
→ The goal is to start from a given arrangement (initial state) and reach a specific target arrangement (goal state) by sliding the tiles

Defining the 8-Puzzle using State Space Search

1) State:
→ A state represents a specific configuration of tiles on 3×3 grid

→ Tile shows position of each numbered tile and empty square

Eg

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | - | 5 |

2) Initial State
→ Starting Configuration

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | - | 5 |

3) Goal State
→ Target Configuration

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | - | 4 |
| 7 | 6 | 5 |

) Actions/Rules (Operators)
→ These are the allowed moves that transform from one state into another
→ Actions are based on moving the empty blank square
o Move BLANK UP, DOWN, LEFT, RIGHT
Moves are possible only if the blank square is not at an edge

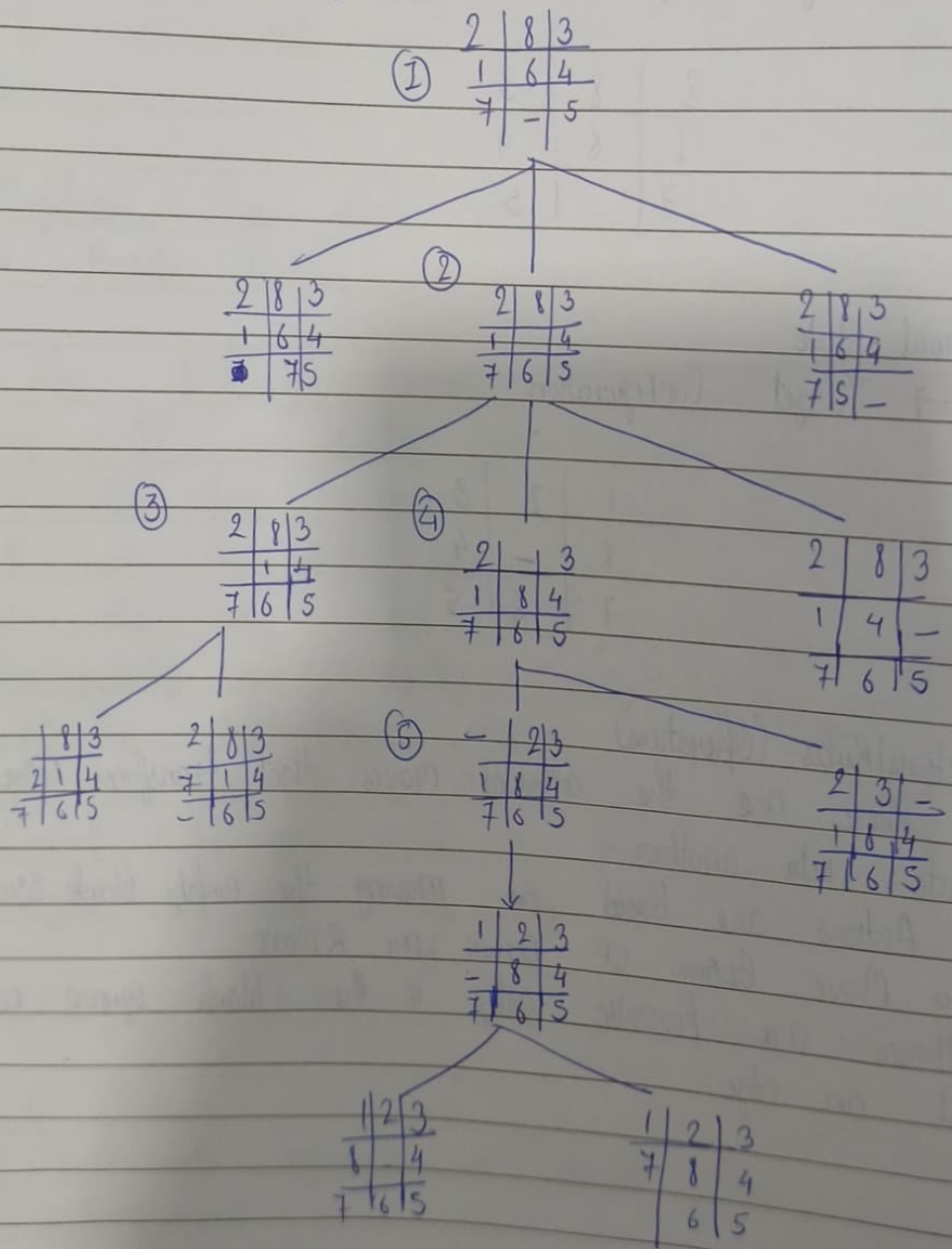
Date
Page

## State Space

→ Set of all possible configurations of 3x3 grid

→ It will be 9! here. (8tiles +1 blank) which is 362,880, However only half of these are reachable from any given starting state. which is 181,440 unique configurations

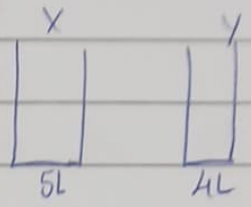
## Solving the 8-Puzzle via State Search



Q) Take 5-24 Sum as Example in Q3

Q3) Enumerate Classical "Water Jug Problem" Describe State Space for this problem and also give the solution

Ans State Space Representation




1) State
   - → Set pair (x, y)
   - → x: 5 litre jug $0 \le x \le 5$
   - → y: 4 litre jug $0 \le y \le 4$
   - → Eg State: (3,4) means 3L in 5L jug(x) & 4L in 4L jug(Y)

2) Initial State
   - → (0,0) because start is empty

3) Goal State
   Goal State : (x, 2), where x can be $0 \le x \le 5$ and y should be Compulsory Filled 2 litres

4) Actions / Rules
   - → These are the actions you can perform to change from one state to another
   - → Fill 5L jug completely from Pump
   - → Fill 4L jug " " " " " "
   - → Empty the 5L jug onto the ground
   - → " " the 4L jug " " " " " "
   - → Pour water from 5L jug into the 4L jug until the 4L jug is full or the 5L jug is empty
   - → Opposite of 1

| Litres in 5L jug (x) | Litres in 3L jug (y) | Action |
|---|---|---|
| 0 | 0 | Fill 5L jug |
| 5 | 0 | Pour 5L into 4L |
| 1 | 4 | Empty 4L jug |
| 1 | 0 | Pour all into 4L jug |
| 0 | 1 | Fill 5L jug |
| 5 | 1 | Pour 5L into 4L until full |
| 2 | 4 | Empty 4L jug |
| 2 | 0 | Pour all (2L) |
| 0 | 2 | Done |

The Sequence of states
$(0,0) \rightarrow (5,0) \rightarrow (1,4) \rightarrow (1,0) \rightarrow (0,1) \rightarrow (5,1) \rightarrow (2,4) \rightarrow (2,0) \rightarrow (0,2)$
represents a path found through the state space search from initial state to a goal state

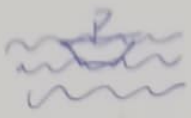
Q-5 State 3 missionaries 3 cannibals problem. Give its State Space Search

Ans Missionaries and Cannibals Problems.
→ 3 M & 3C on one side of the sewer
→ All want to cross the river
→ On same side of river missionaries count can't be less than cannibals
→ Only one boat available that can hold only two people at a time
→ (ML, CL, B)

ML: M on left, CL: C on left, B: Blank
If B: 0 then boat on Right Side
If B: 1 then boat on left side

| Note |
|---|
| M = C ✓ |
| M > C ✓ |
| M < C ✗ |

CCC
MMM



Date
Page

→ MR : 3 - ML
CR - 3 - CL

1) Initial State
→ (3, 3, 1)
All 3 M & 3C and boat is on the left bank

2) Goal State
→ (0, 0, 0)
All 3M & 3C and boat is on the Right bank

3) State Actions (Operators)
move action

4) Constraints
M = C ✓
M > C ✓
M < C ✗

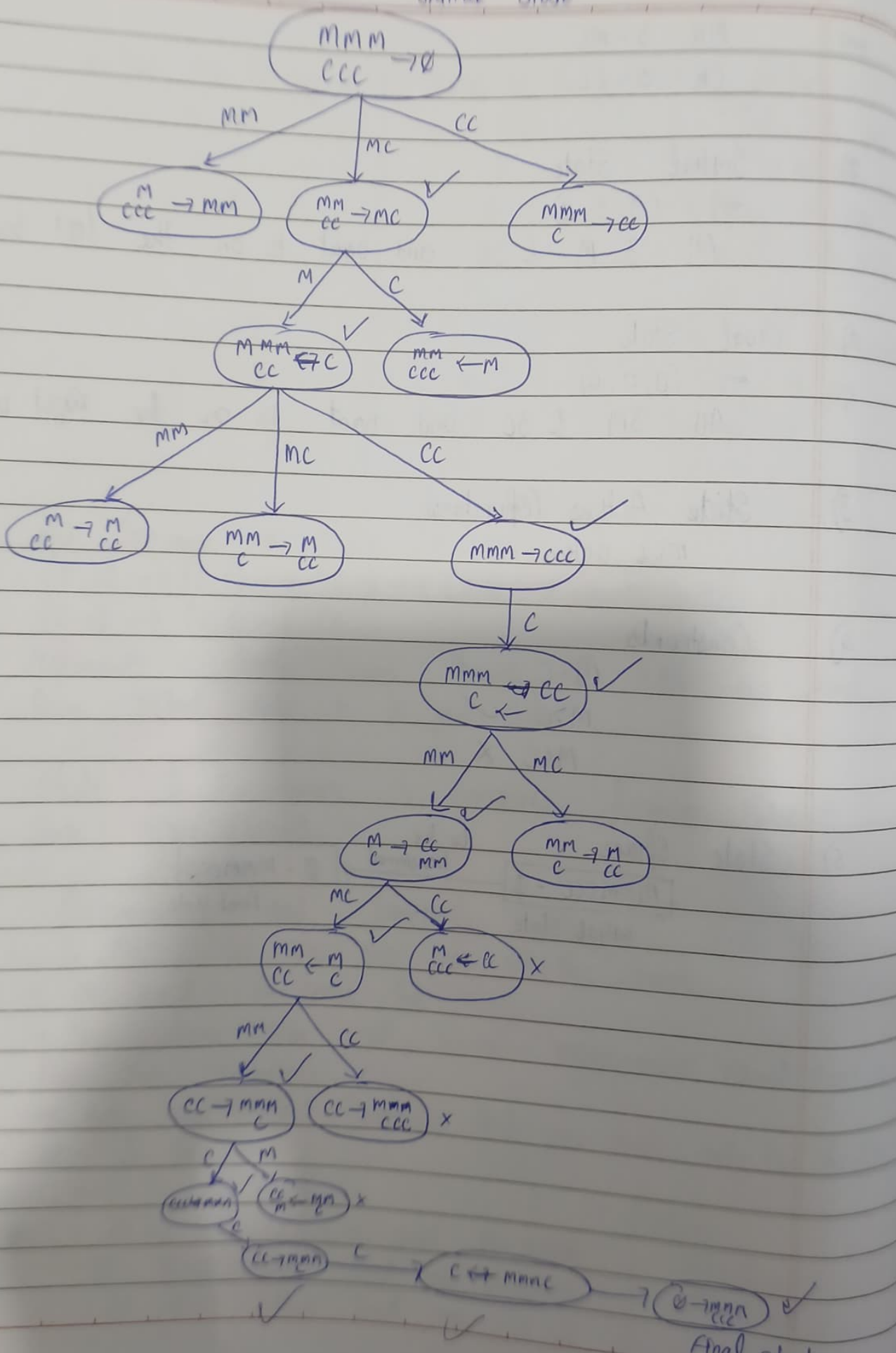
5) State Space

MMMCCC - ϕ → (to be transformed) → ϕ - MMMCCC

Initial State

Final state

Initial State



Q-6 State Money and Banana Problem. Give its shred state space representation

Ans Problem: A monkey is in a room with a box and some bananas hanging high up. How can monkey get bananas

Goal: the monkey needs to have banana in its hand

1) States: A state describes:
   - Where monkey is
   - Where box is
   - If monkey is on floor or on box
   - If monkey has bananas yet

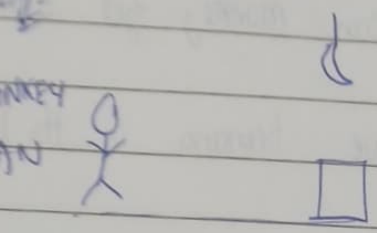
2) Initial State:
   - → Monkey on floor
   - → Monkey does not have bananas
   - → The box is somewhere in room (not under bananas)
   - → The monkey is somewhere in room (not near box)

3) Actions
   - → Go: Walk to diff. place
   - → Push: Push box to diff. place
   - → Climb up: box upon
   - → Climb down: box the niche
   - → Grasp: Grab the bananas

4) Goal Test
   - → Does the monkey have the bananas?

Solution Path

→ finding the solution means finding the sequence of actions that changes the Initial state to Goal state



Explain AI Problem Characteristics in detail.

1) Is the problem decomposable?
2) Can solution steps be ignored or undone?
   - → Ignorable
   - → Recoverable
   - → Irrecoverable
3) Is the problem's universe predictable?
   - → Certain Outcome
   - → Uncertain Outcome
4) Is a good solution absolute or relative?
   - → Any Path Problem
   - → Best Path Problem
5) Is the solution a state or a path?
   - → State
   - → Path
6) What is the role of Knowledge?
   - → Knowledge helps constraint Search
   - → Knowledge is essential for finding a solution?

7) Does the task require interaction with problem?
→ Solitary
→ Conversational

Q-8 Explain Production System Characteristics

Ans: A production system is a simple AI model for solving problems using rules
→ It works like a set of "IF-THEN" rules
→ These rules operate on a database of facts to find a solution

Components of Production System

1) A Set of Rules (Production Rules)
→ "IF-THEN" Statements
→ IF (Condition)
→ THEN (Action)
→ Eg: IF (4-jug is empty) THEN (Fill jug)

2) A Database (A Knowledge or Working Memory)
→ This holds all facts about current state of problem
→ See Image

3) A Control Strategy
→ Brain
→ more than 1 "IF" part true at same time then conflict
→ "resolve" that conflict and pick the best rule to guide the search
→ the good strategy must be systematic (organized, so it doesn't get lost in a loop) and cause motion (make

**AI Problem Characteristics**

When you face a new AI problem, thinking about these seven characteristics can guide you in selecting the right approach:

1. **Is the problem decomposable?**
    - Can you break the problem down into smaller, independent subproblems?
    - If yes (like solving a complex math integral by breaking it into parts), you can solve each part separately and combine the results. This is usually easier.
    - If no (like the Blocks World problem where moving one block affects others), the steps are interdependent, and you need a more integrated plan.
2. **Can solution steps be ignored or undone?**
    - **Ignorable:** If you make a wrong step, can you just ignore it and carry on? (e.g., in theorem proving, proving an unnecessary lemma doesn't prevent the final proof). These problems are often simpler.
    - **Recoverable:** If you make a wrong step, can you go back and undo it? (e.g., in the 8-puzzle, you can backtrack if you make a bad move). These require methods that allow backtracking.

- **Irrecoverable:** If you make a wrong step, is it permanent? (e.g., in chess, once you make a move, you can't take it back). These problems require careful planning and decision-making at each step, as mistakes are costly.

3. **Is the problem's universe predictable?**
   - **Certain Outcome:** Do you know exactly what will happen when you perform an action? (e.g., in the 8-puzzle or Water Jug problem, moving a tile or pouring water has a definite result). Planning works well here.
   - **Uncertain Outcome:** Can an action have unpredictable results? (e.g., in card games, shuffling or drawing a card has a random element). These problems require planning for probabilities and might not guarantee a solution. Solving them can be much harder.
4. **Is a good solution absolute or relative?**
   - **Any-Path Problem:** Is any solution acceptable, regardless of the path taken to reach it? (e.g., finding *any* sequence of moves to solve the 8-puzzle or Water Jug problem). Heuristics can often find *a* solution quickly.

4. **Is a good solution absolute or relative?**
    - **Any-Path Problem:** Is any solution acceptable, regardless of the path taken to reach it? (e.g., finding *any* sequence of moves to solve the 8-puzzle or Water Jug problem). Heuristics can often find *a* solution quickly.

---

    - **Best-Path Problem:** Do you need the *best* (e.g., shortest, cheapest, fastest) solution? (e.g., the Traveling Salesman Problem requires the shortest route). These usually require more exhaustive search to compare different paths and find the optimal one.

find the optimal one.

5. **Is the solution a state or a path?**
   - **State:** Does the solution only require reaching a final state without needing to know *how* you got there? (e.g., natural language understanding - the final interpretation matters, not the steps taken to get it).
   - **Path:** Does the solution require the sequence of steps (the path) taken to reach the goal state? (e.g., the Water Jug problem requires the sequence of pours; the 8-puzzle requires the sequence of moves).
6. **What is the role of knowledge?**
   - How much knowledge is needed to solve the problem?
   - **Knowledge helps constrain search:** Some problems require only basic rules (like chess rules), but a lot of extra knowledge (strategy, tactics) can dramatically improve performance.
   - **Knowledge is essential for finding a solution:** Some problems require a vast amount of knowledge just to recognize what a solution looks like (e.g., predicting political trends).

7. **Does the task require interaction with a person?**
    - **Solitary:** Can the computer solve the problem on its own, given the description, and just output the answer? (e.g., proving a simple mathematical theorem).
    - **Conversational:** Does the system need to interact with a person during the problem-solving process? This might be to get more information, ask questions, or explain its reasoning. (e.g., a medical diagnosis system needs to ask about symptoms and explain its diagnosis).

**Analysis of Tower of Hanoi Problem**

First, let's briefly state the **Tower of Hanoi problem**:

- You have three **rods** (let's call them A, B, C).
- On **rod** A, you have a stack of N disks of different sizes, arranged from largest at the bottom to smallest at the top.
- **Goal:** Move the entire stack of disks from **rod** A to **rod** C.
- **Rules:**
    1. You can only move one disk at a time.
    2. You can only move the top disk from one stack and place it onto another **rod**.
    3. You can never place a larger disk on top of a smaller disk.

Now, let's analyze it using the seven characteristics:

1. **Is the problem decomposable?**
    - **Yes.** The problem has a classic recursive structure. To move N disks from rod A to rod C, you can:
        - Move N-1 disks from rod A to rod B (subproblem 1).

1. **Is the problem decomposable?**
    - **Yes.** The problem has a classic recursive structure. To move N disks from rod A to rod C, you can:
        - Move N-1 disks from rod A to rod B (subproblem 1).
        - Move the largest disk (disk N) from rod A to rod C (simple step).
        - Move N-1 disks from rod B to rod C (subproblem 2).
    - These subproblems are smaller versions of the original problem.
2. **Can solution steps be ignored or undone?**
    - **Recoverable.** If you make a move that violates the rules or a move that doesn't lead towards the optimal solution, you can always backtrack and undo that move by moving the disk back. The standard algorithm doesn't require backtracking, but the state space allows it.

3. **Is the problem's universe predictable?**
    - **Yes.** The outcome of moving a disk is completely predictable. If you move the top disk from rod X to rod Y, it results in a definite, known new state, assuming the move is legal. There's no uncertainty or chance involved.
4. **Is a good solution absolute or relative?**
    - It's typically considered a **Best-Path Problem**. The standard goal is to find the solution with the *minimum* number of moves. There is a well-known optimal algorithm (the recursive one) that finds this shortest path. Just finding *any* way to move the disks would be an any-path approach, but inefficient.
5. **Is the solution a state or a path?**
    - **Path.** The solution isn't just knowing the final state (all disks on rod C). The required output is the sequence of moves (the path) that achieves this goal state following the rules.

6. **What is the role of knowledge?**
    - **Minimal knowledge is required.** Only the rules of the puzzle (move one disk, never larger on smaller) are needed to find *a* solution. Understanding the recursive structure (the knowledge of the optimal algorithm) is needed to find the *best* solution efficiently, but vast amounts of external knowledge are not necessary.
7. **Does the task require interaction with a person?**
    - **Solitary.** The problem can be solved by a computer program without needing intermediate input or clarification from a human. The program can compute the entire sequence of moves and output it.

7) Does the task require interaction with problem?
→ Solitary
→ Conversational

Q-8 Explain Production System Characteristics

Ans: A production system is a simple AI model for solving problems using rules
→ It works like a set of "IF-THEN" rules
→ These rules operate on a database of facts to find a solution

Components of Production System

1) A Set of Rules (Production Rules)
→ "IF-THEN" Statements
→ IF (Condition)
→ THEN (Action)
→ Eg: IF (4-jug is empty) THEN (Fill Jug)

2) A Database (A Knowledge or Working Memory)
→ This holds all facts about current state of problem
→ See Image

3) A Control Strategy
→ Brain
→ more than 1 "IF" part true at same time then conflict
→ "resolve" that conflict and pick the best rule to guide the search
→ the good strategy must be systematic (organized, so it doesn't get lost in a loop) and cause motion (make

progress toward the goal)

4) A Rule Applier (or Interpreter)
→ It actually applies the chosen rule
→ It performs the action from the "THEN" part, which updates the facts in Database

o Production System Characteristics
1) Monotonic System
2) Non-Monotonic System
3) Partially Commutative System
4) Commutative System

**Components of a Production System**

A production system has four main parts:

- **1. A Set of Rules (Production Rules):**
  - These are the "IF-THEN" statements.
  - The **IF** part is the **Condition**. It checks the current facts.
  - The **THEN** part is the **Action**. It performs a task or changes a fact.
  - *Example:* IF (4-jug is empty) THEN (Fill 4-jug).
- **2. A Database (Knowledge Base or Working Memory):**
  - This holds all the facts about the current state of the problem.
  - *Example:* (4-jug is empty) and (3-jug is full).
  - This database changes as rules are applied.

- This database changes as rules are applied.

- **3. A Control Strategy:**
  - This is the "brain" that decides **which rule to apply next**.
  - You need a strategy because, in many problems, more than one rule's "IF" part might be true at the same time. This is called a "conflict."
  - The control strategy's job is to "resolve" that conflict and pick the best rule to guide the search.
  - A good strategy must be **systematic** (organized, so it doesn't get lost in a loop) and **cause motion** (make progress toward the goal).
- **4. A Rule Applier (or Interpreter):**
  - This is the part that actually **applies** the chosen rule.
  - It performs the action from the "THEN" part, which updates the facts in the database.

- **1. Monotonic System:**
  - This is a system where applying a rule **only adds new facts**.
  - It **never deletes or changes** any existing facts.
  - *Example (Math Proof):*
    - **Fact 1:** A = 5
    - **Fact 2:** B = A

if ( )
then x++;
x = 1

---

- When the system applies a rule and adds **New Fact 3:** B = 5, the old facts (A=5, B=A) are still true. Nothing was deleted.

- **2. Non-Monotonic System:**
  - This is a system where applying a rule **can change or delete** old facts.
  - An action you take now *might* prevent you from taking another action later.
  - *Example (Blocks World):*
    - **Fact 1:** (Block_A is clear)
    - **Rule:** (Stack X on A)
    - When you apply the rule Stack(B, A), you add a new fact (B is on A).
    - But this action *deletes* the old fact (Block_A is clear). This change prevents you from taking other actions, like Stack(C, A).

prevents you from taking other actions, like Stack(C, A).

- **3. Partially Commutative System:**
  - This is a system where the **order** in which you apply certain rules **does not matter** to the final result.
  - *Example (8-Puzzle):*
    - Imagine the empty space is in the middle. You can move the tile *above* it down, or the tile to the *right* of it left.
    - Doing (Move Down) then (Move Left) results in the *exact same* board state as doing (Move Left) then (Move Down). The moves are commutative.

- **4. Commutative System:**
  - This is a system that is **both Monotonic AND Partially Commutative.**
  - These are the simplest systems. The order of rules doesn't matter, and no rule ever deletes or changes another fact.
  - *Example (Shopping List):*
    - **Rule 1:** IF (need_milk) THEN (add milk_to_list)
    - **Rule 2:** IF (need_bread) THEN (add bread_to_list)
      - Running Rule 1 then Rule 2 gives the same final list as running Rule 2 then Rule 1. No facts are deleted.

Q-9 What is "Control Strategy" and what are its characteristics?

Ans A control strategy is the heart of an AI system, often within a Production System, that determines which rule or action to apply next during the process of searching for a problem solution

Characteristics

1) It should cause Motion
→ The strategy must actually choose and apply rules that lead to a new state
→ It must prevent program from looping endlessly or getting stuck by repeatedly choosing same rule or returning to same state

2) It should be systematic
→ The search process must be organized and logical in how it explores possibilities

→ A systematic strategy ensures that the program doesn't overlook potential solutions

→ It prevents exploring the same path or state multiple times unnecessarily, which wastes effort

o Classification of Control Strategies

→ Uninformed / Blind Search

→ They search the entire space without having extra information about how close a state is to the goal.

Eg: BFS & DFS

→ Informed / Directed Search: These strategies use additional knowledge about problem domain, called heuristics, to estimate which path is most promising.

Eg: Best-First Search & A* (A-Star)

Q-10 Difference betwn DFS and BFS

Ans See Image

Q-11 Difference betwn Informed and Uninformed Search

Ans See Image

Q-12 What is Heuristic Search? Discuss with an example

Ans Heuristic Search (also called Informed Search or Directed Search) is an AI problem-solving technique that uses "rules of thumb" or "smart glasses" to find a solution more quickly

Here is the **difference between BFS and DFS in tabular form** 👇

| Basis | BFS (Breadth-First Search) | DFS (Depth-First Search) |
|---|---|---|
| **Traversal order** | Visits level by level (neighbors first) | Goes deep along a path before backtracking |
| **Data structure used** | **Queue** | **Stack** (or recursion) |
| **Memory usage** | High (stores many nodes at same level) | Low (stores only current path) |
| **Shortest path** | **Finds shortest path** in an unweighted graph | Does **not guarantee** shortest path |
| **Completeness** | Complete (will find a solution if it exists) | May get stuck in deep paths (not always complete) |
| **Suitable for** | Finding shortest path, nodes close to root | Solving puzzles, path existence, topological tasks |
| **Traversal pattern** | Horizontal movement | Vertical movement |
| **Time complexity** | O(V + E) | O(V + E) |
| **Example usage** | Social networks (finding closest connections) | Maze solving, backtracking problems |

Here is the **difference between Informed and Uninformed Search** in a clear **tabular format** 👇

| Basis | Uninformed Search (Blind Search) | Informed Search (Heuristic Search) |
|---|---|---|
| **Definition** | Searches without any extra knowledge about goal location | Uses additional information (heuristics) to guide the search |
| **Use of heuristics** | ❌ Does **not** use heuristics | ✅ Uses heuristic functions (h(n)) |
| **Search efficiency** | Slower and explores more nodes | Faster and more efficient (guided towards goal) |
| **Optimality** | Some are optimal (e.g., BFS), some not (DFS) | Can be optimal (e.g., A*), depending on heuristic |
| **Memory requirement** | Can be high (e.g., BFS) or low (DFS) | Generally higher due to storing heuristic values |
| **Examples** | BFS, DFS, Uniform Cost Search | A* Search, Greedy Best-First Search, Hill Climbing |
| **Knowledge required** | Only problem definition (start, goal, actions) | Requires extra domain knowledge (heuristic estimation) |
| **Goal approach** | Blindly explores until it finds the goal | Moves intelligently toward goal using estimates |

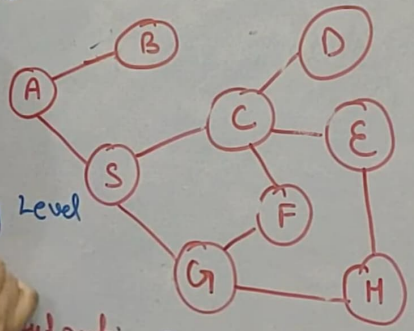
Breadth First Traversal - Breadth First Search (BFS) Graph in Data Structure in Hindi

www.youtube.com – to exit full screen, press esc

# Graph Traversal in D.S. (BFS)

BFS ⇒ In breadth first search, one node is selected as a start position. It's visited and marked, then all unvisited nodes adjacent of the next node are visited and marked in some sequential order.

Queue status.




BFS
Level-by Level
Tra

utput: A, B, S, C, G, D, E, F, H

5:29 / 6:27

→ Instead of searching blindly, it uses additional knowledge about problem to intelligently guide the search toward the most promising paths.
→ This extra knowledge is provided by a heuristic function. This function looks at a current state (like a puzzle layout) and gives it a score or number. This score estimates how close that state is to the final goal.

Eg: The 8-Puzzle

See Image

Q-13 Explain Hill Climbing method in brief OR Write objective of Hill Climbing method. Discuss any one type of hill algorithm

Ans Objective: The main objective of Hill Climbing is to find a good solution to problem by always moving towards a better state. It is a heuristic search techniques

Simple Hill Climbing Algorithm
1) Start at an initial state. Evaluate this state (get its score) and call it the CURRENT state
2) Start a loop
3) Select a neighbouring state that you haven't tried yet. Let's call this the NEW state.
4) If there are no more neighbours to check, stop the loop. The CURRENT state is your final answer
5) Evaluate the NEW state

**Example: The 8-Puzzle**

- **Problem:** You have a 3x3 grid with 8 numbered tiles and one empty space. The goal is to slide tiles to reach a specific final layout.
- **Uninformed Search (like BFS):** Would explore all possible moves from the start, then all moves from those new states, level by level. It has no idea if a move is good or bad.
- **Heuristic Search:** A heuristic function can be used to "score" each possible move.
    - **A Simple Heuristic:** "Count the number of tiles that are *not* in their correct final position." A lower score is better because it means the state is *closer* to the goal.
- **How it works:**
    - Imagine the current state has **5** tiles in the wrong place.
    - You have two possible moves:
        1. **Move A** leads to a new state with **4** tiles in the wrong place.
        2. **Move B** leads to a new state with **6** tiles in the wrong place.
    - The heuristic search will choose to explore **Move A** first, because its score (4) is better (lower) than Move B's score (6). It is *guessing* that Move A is a step in the right direction. (Example in Handwritten Notes - Page No.: 15)

Q-13 Explain Hill Climbing method in brief OR Write objective of Hill Climbing method. Discuss any one type of hill algorithm

Ans Objective: The main objective of Hill Climbing is to find a good solution to problem by always moving towards a better state. It is a heuristic search techniques

Simple Hill Climbing Algorithm

1) Start at an initial state. Evaluate this state (get its score) and call it the CURRENT state
2) Start a loop
3) Select a neighbouring state that you haven't tried yet. Let's call this the NEW state.
4) If there are no more neighbours to check, stop the loop. The CURRENT state is your final answer
5) Evaluate the NEW state

6) If the NEW state's score is better than CURRENT state score:
- Set CURRENT = NEW (you've moved uphill)
- Go back to Step 2 (to start searching from this new CURRENT state)

7) If the NEW state is not better, go back to Step 3 (and check a different neighbour from original CURRENT state)

Q-14 Explain the algorithm for steepest hill climbing

Ans Instead of moving to the first better neighbour, it checks all neighbours and picks the best one

- Algorithm Steps

1) Start an Initial state. Evaluate this state and call it the CURRENT state
2) Start a loop:
3) Generate all possible neighbouring states that you can reach from CURRENT
4) Evaluate every one of these neighbours
5) Find the neighbour with highest score (the "steepest" uphill move). Let's call this the BEST_NEIGHBOUR
6) If the BEST_NEIGHBOUR score is better than the CURRENT state's score:
   → SET CURRENT = BEST_NEIGHBOUR (you've made the best possible move)
   → Go back to Step 2 (to continue the climb from new state)
7) If the BEST_NEIGHBOUR's score is not better than CURRENT (meaning no neighbour uphill)

Q-15 Why Hill Climbing is Better than Generate-and-Test

Ans See Image

Q-16 Write the proposed Solutions for the Problems occured in hill climbing

Ans 1) For Local Maxima
→ The main solution is Backtracking. If you get stuck at a Peak, go back to an earlier, worse state and try exploring a new path

2) For Plateaus:
→ The solution is make a big jump. If you are on a flat area, pick a state randomly from somewhere else in the problem and restart the climb from that new spot

3) For Ridges
→ The solution is to allow for bigger steps or sequences of moves. Instead of just evaluating one move, try applying two or three moves (or move in diagonal directions) before evaluating the new state

Q-17 Explain Best-First Search with Algorithm

Ans → Informed (Heuristic) Search Algorithm
→ Mix of DFS & BreadthFS
→ Main Idea: Instead of exploring blindly (like BFS or DFS), Best First Search uses a heuristic function (a scoring function) to estimate how promising each state (node) is

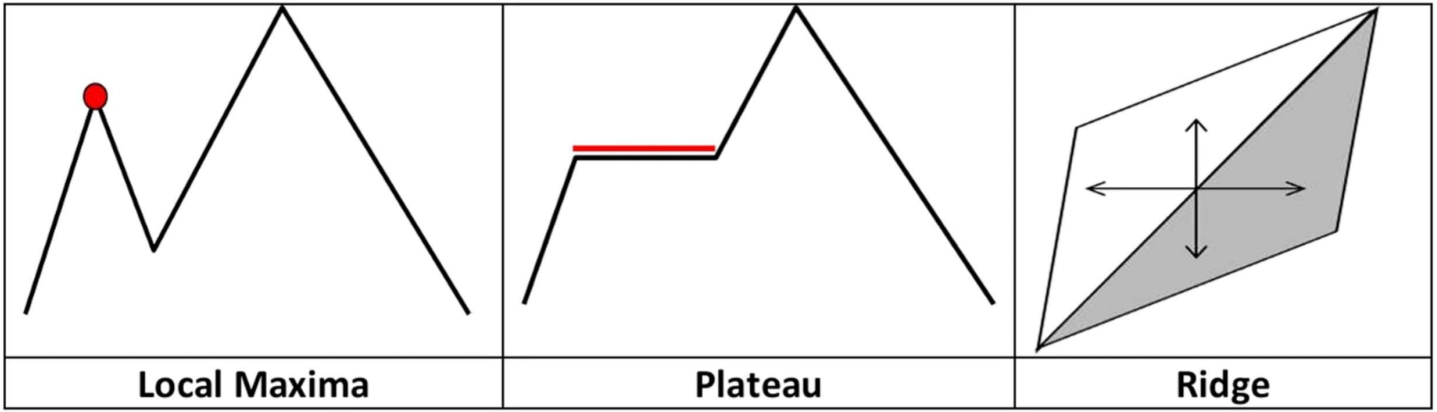
**Why Hill Climbing is Better than Generate-and-Test**

- **Generate-and-Test** is a "blind" search. It generates a completely random solution and then checks if it's the goal. It doesn't learn anything from its guesses. If a guess is bad, it throws it away and generates another random one.
- **Hill Climbing** is "smarter" because it uses **feedback**.
    - It doesn't guess randomly every time. It starts at one point and only moves to a **new state** if that state is **better** (uphill) than the current one.
    - This feedback from the heuristic function helps *guide* the search in the correct direction, making it much more efficient than the random stumbling of Generate-and-Test.

1. **Local Maximum:**
    - A local maximum is a state (a "peak" or "foothill") that is **better than all of its neighbors**, but it is **not the best solution** in the entire problem (which is the "global maximum" or the highest peak).
    - Since Hill Climbing only looks at its immediate neighbors, it reaches this peak, sees no better step to take, and stops. It thinks it has found the solution, even though a much better one exists elsewhere.
2. **Plateau:**
    - A plateau is a "flat" area in the search space.
    - In this area, the CURRENT state and all of its neighboring states have the **exact same evaluation score**.
    - Since no neighbor is "better", the algorithm doesn't know which way to move and stops, unable to find the path that leads off the plateau and back uphill.
3. **Ridge:**

and stops, unable to find the path that leads off the plateau and back uphill.

3. **Ridge:**
   - A ridge is a special kind of problem, like a long, narrow mountain top.
   - The path to the top moves along this ridge (e.g., diagonally).
   - However, the algorithm's allowed moves (e.g., only North, South, East, West) may all lead "downhill" off the ridge.
   - Because every *single* move looks worse, the algorithm stops, even though a path to the top exists if it could move diagonally (or take two steps at once).



Q-12 Why Hill Climbing is Better than Generate-and-Test

Ans See Image

Q-13 Write the proposed Solutions for the problems occur in hill climbing

Ans 1) For Local Maxima
→ The main solution is Backtracking. If you get stuck at a peak, go back to an earlier, worse state and try exploring a new path.

2) For Plateaus:
→ The solution is make a big jump. If you are on a flat area, pick a state randomly from somewhere else in the problem and restart the climb from that new spot.

3) For Ridges
→ The solution is to allow for bigger steps or sequences of moves. Instead of just evaluating one move, try applying two or three moves (or move in diagonal directions) before evaluating the new state.

Q-14 Explain Best-First Search with Algorithm

Ans → Informed (Heuristic) Search Algorithm
→ Mix of DFS & BreadthFS
→ Main Idea: Instead of exploring blindly (like BFS or DFS), Best First Search uses a heuristic function (a score function) to estimate how promising each state (node) is

Algorithm

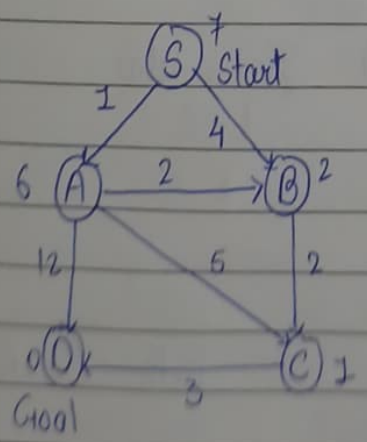
1) Use two ordered list OPEN and CLOSED
2) Start the initial node 'n_i' and put it on the ordered list OPEN
3) Create a list CLOSED. This is initially an empty list
4) If OPEN is empty then exit with failure
5) Select first node from OPEN. Remove it from OPEN and put it on CLOSED. Call this node n
6) If 'n' is the goal node exit. The solution is obtained by tracing a path backward along the arcs in the tree from 'n' to 'n_i'
7) Add the successors in the OPEN List
8) Reorder the list OPEN, according to the heuristic and go back to step 4

Q-18 Difference between Breadth FS & Best FS

Ans See Image

Q// A* Algorithm

Ans



Differentiating between breadth-first search and best-first search algorithm. in table form

Here you go ✅ **clear and exam-ready table** comparing **BFS vs Best-First Search**:

| Basis | Breadth-First Search (BFS) | Best-First Search |
|---|---|---|
| **Type of search** | Uninformed (blind) search | Informed (heuristic-based) search |
| **Selection criteria** | Expands nodes level-by-level (FIFO) | Expands node with **lowest heuristic value** |
| **Data structure used** | **Queue** | **Priority queue / ordered OPEN list** |
| **Knowledge required** | Only problem definition (no heuristics) | Requires heuristic function **h(n)** |
| **Goal behavior** | Guaranteed shortest path in **unweighted** graphs | Not guaranteed to find shortest path |
| **Time & space usage** | High (stores whole level) | Depends on heuristic; can be efficient |
| **Exploration pattern** | Systematic, complete search | Guided towards goal node |
| **Examples** | Finding shortest path in unweighted graph | Greedy Best-First Search (maze solving) |
| **Completeness** | Complete (if solution exists) | May or may not be complete |

For Start

$f(n) = g(n) + h(n)$
$= 0 + 7$
$= 7$

$S \to A$
$f(n) = g(n) + h(n)$
$= 1 + 6$
$= 7$

$S \to B$
$f(n) = 4 + 2$
$= 6$

$S \to B \to C$
$f(n) = B + 1$
$= 7$

$C \to D$
$f(n) = 3 + 0$
$= 3$

$S \to A \to B : 3 + 2 = 5$
$S \to A \to C : 6 + 1 = 7$
$S \to A \to D : 13 + 0 = 13$

$S \to A \to B \to C : f(n) = g(n) + h(n)$
$= 5 + 1$
$= 6$

$S \to A \to B \to C \to D : f(n) = g(n) + h(n)$
$= 8 + 0$
$= 8$

$S \to B \to C \to D = 9 + 0$
$= 9$

Path: $S \to A \to B \to C \to D$

$S \to A \to C \to D = 9 + 0$
$= 9$

✗ $S \to A = 11$
✗ $S \to B = 16$
✗ $S \to B \to C = 17$
✗ $S \to A \to B = 15$
✗ $S \to A \to C = 17$
✡ $S \to A \to D = 13$
✗ $S \to A \to B \to C = 16$
✡ $S \to A \to B \to C \to D = 8$
✡ $S \to B \to C \to D = 9$
✡ $S \to A \to C \to D = 9$

Q-19 Explain A* (A-Star) Algorithm

Ans → Informed (Heuristic) Search Algorithm
→ It finds the cheapest by looking at two costs at same time
1) g(n) : the actual cost to get from start to a node n
2) h'(n) : the estimated cost (the guess) to get from node n to the goal

$$F'(n) = g(n) + h'(n)$$

F(n) = Total estimated Cost
g(n) = actual cost from state n to the goal state
h(n) = this is estimated (heuristic) cost from state n to goal state

o Algorithm of A*
1) Put the start node in open (OPEN) List
2) Choose the node with the smallest F(n) = g(n) + h(n) from OPEN
3) If it is the goal, stop and trace back the path
4) Move the node to CLOSED (means its already explored)
5) Generate all successors of that node
6) For each successor:
   - Calculate g(n), h(n), F(n)
   - If a better version already exists, ignore it
   - Otherwise, keep it and set its parent
7) Add valid successors to OPEN and repeat from Step 2.

Q-20 Problem reduction using AND-OR graph

Ans Problem reduction is problem solving technique where you break down a large, complex problem into a set of smaller subproblems
→ The idea is that if you can solve all the small subproblems, you have successfully solved the main problem
→ This is also known as "decomposing" the problem

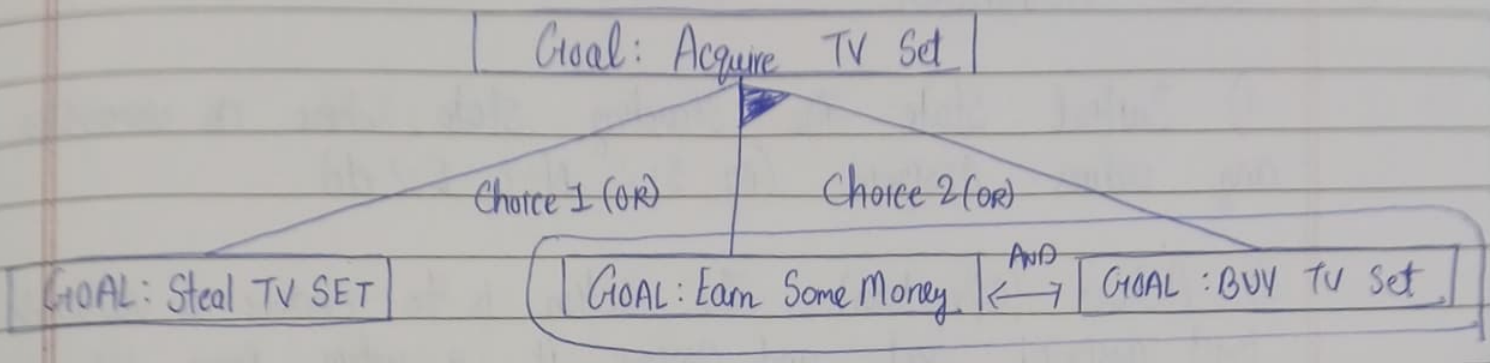
- AND OR Graphs
  An AND OR graph is diagram we use to show how a problem is broken down.

  It has two types of arcs coming from a problem

- OR arcs
  → These represent choices
  → If a problem has OR arcs, it means you only need to solve one of the subproblems to solve the main one
  → You can choose which path to take

- AND arcs
  → These represent Dependencies
  → If a problem has AND arcs, it means you must solve all of the subproblems connected by that arc
  → These are often shown with a small curved line connecting the paths

Eg: The "Acquire TV set" Problem




Q-21 Discuss with example: Constraint Satisfaction Problem

Ans Constraint Satisfaction Problem (CSP) is a specific type of problem in AI

→ The goal is to find a set of values for several variables

→ The solution must follow a set of values for several variables

→ The solution must follow a set of rules or limitations called constraints

→ Instead of finding a path (like in 8-Puzzle), a CSP is about finding a final state where all the rules are satisfied

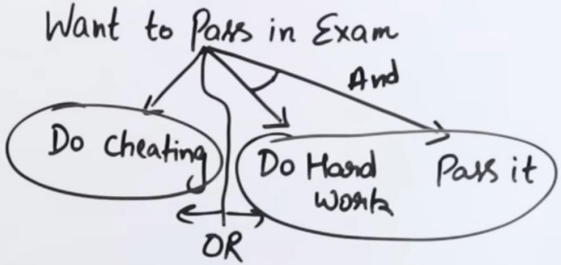
To define CSP you need three components:

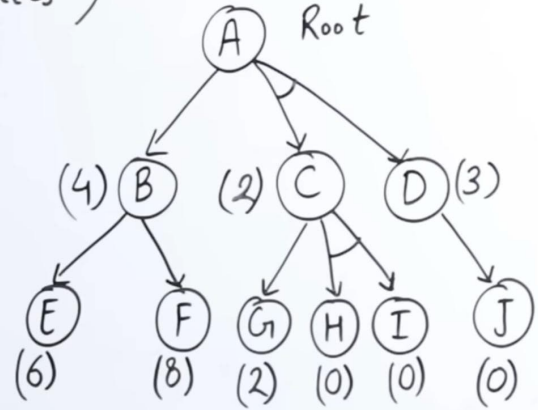
1) Variables: A set of items that you need to find values for (eg. the letters S, E, N, D, M, O, R, Y)

2) Domains: A set of possible values a variable can have (eg for cryptarithmatic, the domain for each letter is set of numbers {0, 1, 2, 3, 4, 5, 6, 7, 8, 9})

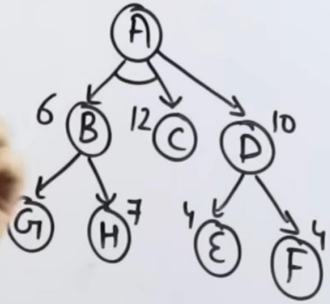
3) Constraints: A set of rules that the final values must obey (eg M cannot be 0, all letters must have unique numbers, and the sum SEND + MORE equal MONEY)

D* (AND/OR) → Problem Decomposition (Breakdown into smaller pieces)



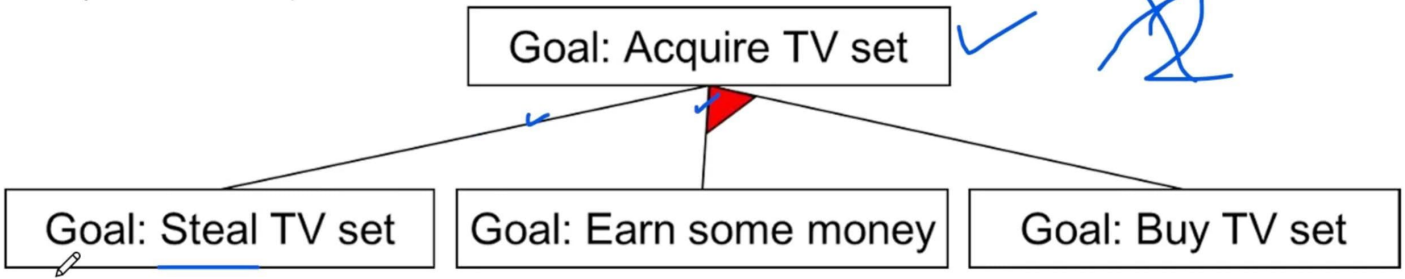




...nt explore all the solution paths once
...lution.




Like & Subscribe

**Example: The "Acquire TV set" Problem**




Let's say your main goal is **"Goal: Acquire TV set"**.

1. **Top Goal:** Acquire TV set
2. Your graph shows you have **OR** choices:
   - **Choice 1 (OR):** You could solve the subproblem Goal: Steal TV set.
   - **Choice 2 (OR):** You could solve an AND combination of two subproblems:
     - Goal: Earn some money
     - **AND**
     - Goal: Buy TV set

**How you solve it:**

- To solve the problem, you need to find one complete solution path.
- Following **Choice 1** is a complete solution.
- Following **Choice 2** is only a complete solution if you solve *both* the "Earn some money" subproblem *and* the "Buy TV set" subproblem.

The AND-OR graph lets you represent this whole plan. An AI algorithm can then search this graph to find the simplest or best solution path.

# CSP Formulation

1) Initial State: the starting state where no variables have any values assigned (eg S=?, N=?, E=? etc)

2) Actions (Operators): the action is to choose a variable that isn't assigned yet and assign it a value from its domain (eg "Assign the value 1 to M")

3) Goal State: A state where all variables have been assigned a value, AND all the constraints are satisfied.

```
  SEND
+ MORE
-------
 MONEY
```

| | |
|---|---|
| S | |
| E | |
| N | |
| D | |
| M | |
| O | |
| R | |
| Y | |

$C_4 = 1$ $C_3 = 0$ $C_2 = 1$ $C_1 = 1$

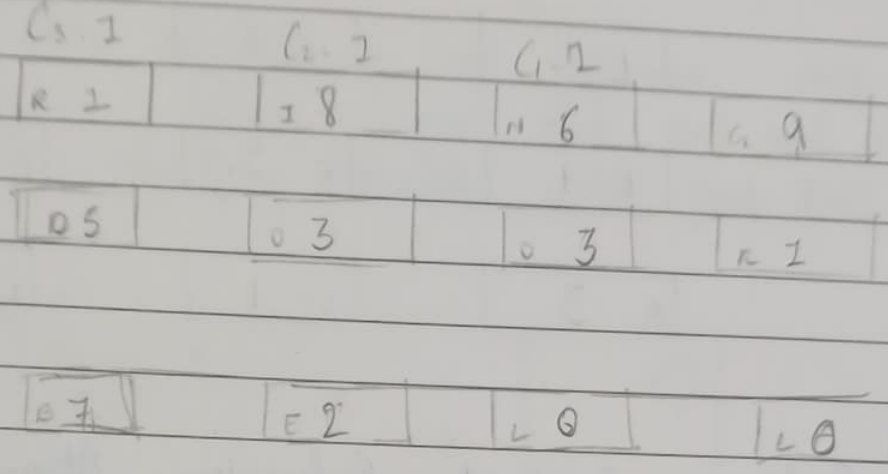
| | | | | |
|---|---|---|---|---|
| | S 9 | E 5 | N 6 | D 7 |
| + | M 1 | O 0 | R 8 | E 5 |
| M 1 | O 0 | N 6 | E 5 | Y 2 |

1) $S + M = 10$ , $S + M \geq 10$

$S = 9$, $M = 1$

1) F+0 = N ; IFG=0

If $C_2 = 1$

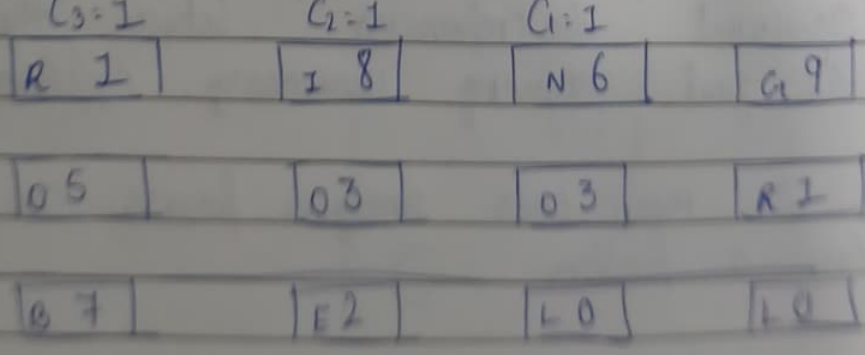



| | |
|---|---|
| 1,2 | 6,2 |
| 2,3 | |
| 3,4 | |
| 3,6 | |

| | |
|---|---|
| R | 1 |
| I | 8 |
| N | 6 |
| G | 9 |
| O | 5 |
| O | 3 |
| E | 2 |
| B | 7 |
| L | 0 |

5-25 ①-1(c)

Solution:



**Constraint Propagation Algorithm**

**Constraint Propagation** is a powerful technique used to solve CSPs more efficiently. It's not a complete algorithm on its own but a method used *during* the search.

The main idea is that when you make a guess (assign a value to one variable), you immediately check how that choice limits the options for all other related variables. This "propagates" (spreads) the consequences of your choice.

This process has two main steps that repeat:

1. **Propagate Constraints:**
    - When you assign a value (e.g., "Guess M = 1"), the system immediately checks all constraints involving M.
    - For example, a constraint is that all letters are unique. By setting M = 1, the system *propagates* this fact by removing 1 from the allowed domain (list of possible values) for all other variables (S, E, N, D, etc.).
    - This can chain together. If you find E must be 5, and there's a rule O = E + 1, propagation will automatically determine O = 6.

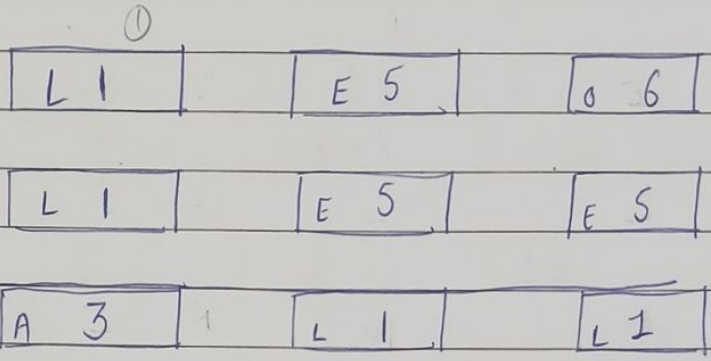
system *propagates* this fact by removing 1 from the allowed domain (list of possible values) for all other variables (S, E, N, D, etc.).
    - This can chain together. If you find E must be 5, and there's a rule O = E + 1, propagation will automatically determine O = 6.
    - This step helps you find a **contradiction** (a dead end) early, so you can backtrack immediately.
2. **Search (If Needed):**
    - If propagation doesn't solve the whole problem or find a contradiction, the system will be stuck.
    - At this point, you must make another **guess**. You pick an unassigned variable and assign it a value.
    - After you make this guess, you go back to Step 1 and propagate the constraints from this new guess.

This cycle of **"Propagate -> Guess -> Propagate -> Guess..."** is how the constraint satisfaction algorithm works.

---

Q OR Q-4 (c) S-24

Q-23 LEO + LEE = ALL




Q-24 Explain Constraint Propagation Algorithm using suitable example

Ans See Image

Q-25 Explain mean-end analysis approach to solve AI problems [4m]

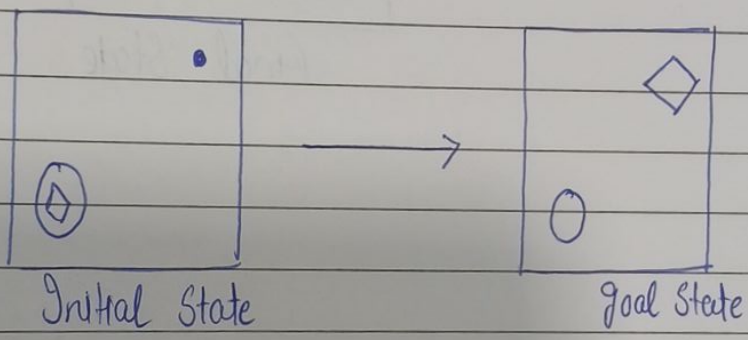
Ans A Strategic approach combining forward and backward reasoning to tackle complex and large problems in AI

→ It helps in focusing the problem-solving effort on significant discrepancies between the current state and the goal, reducing unnecessary exploration

→ The mean ends analysis process can be applied recursively for a problem

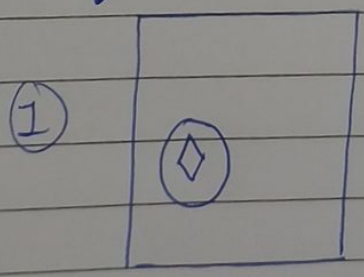
→ Following are main steps which describes the working of MEA techniques for solving a problem.

1) First, evaluate difference between Initial state and Goal state
2) Select the various operators which can be applied for each difference
3) Apply the operator at each difference, which reduces the difference between current state and goal state.
4) Apply operators repeatedly until goal state is found.

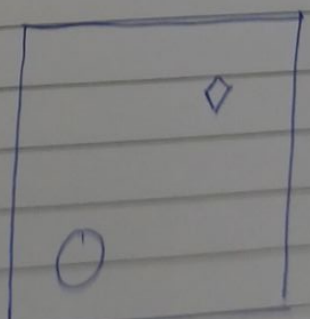



1) Delete Operation on Initial State

Delete •

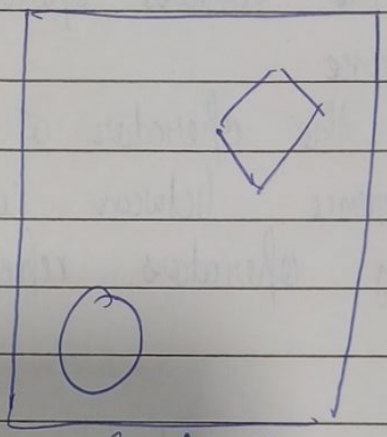



2) Move Operation on (1)



3) Expand Operation on ②

Size of △ is to be expanded now



Final State

# Ch-3

Q-1 Explain different approaches to knowledge representation

Ans
1) Representational Adequacy
2) Inferential Adequacy
3) Inferential Efficiency
4) Acquisitional Efficiency

1) Representational Adequacy
→ This is the ability to represent all the different kinds of knowledge that are needed for a specific problem
→ If your system is for medical diagnosis, it must be able to represent knowledge about symptoms, diseases, patients and tests

2) Inferential Adequacy
→ This is ability to use the knowledge the system already has (the old facts) to figure out (infer) new facts
→ It means the system can logically manipulate its knowledge to create new knowledge

3) Inferential Efficiency
→ This is the ability to make the system smarter about how it finds new facts, so it doesn't waste time
→ It means the system can use extra information to guide its search (inference) towards the most promising answers first

4) Acquisitional Efficiency
→ This is the ability for the system to learn or "acquire" new knowledge easily

→ Ideally, the system should be able to learn new things automatically, without needing a human to manually add every single new fact

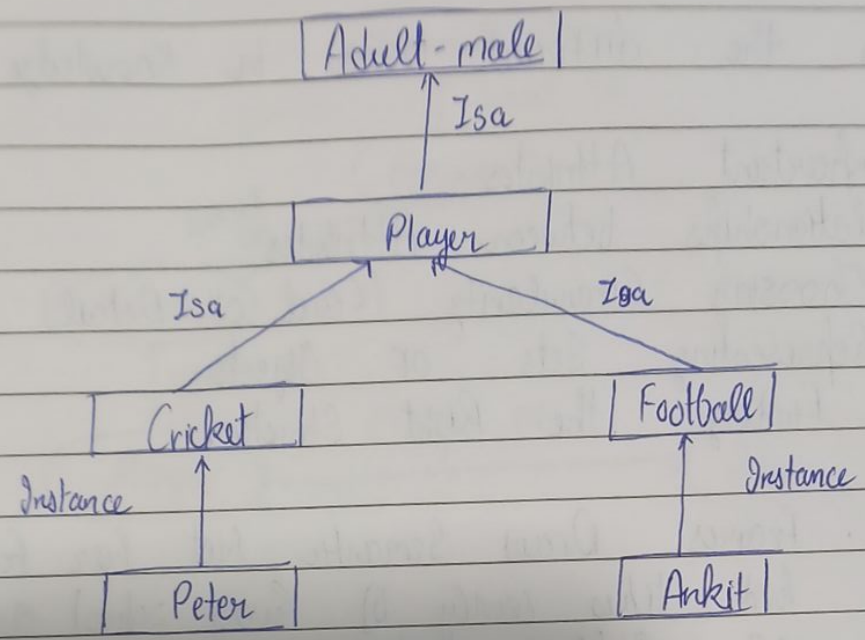
Q-2 Explain the various method (or Schemes) for Knowledge Representation

Ans 1) Relational Knowledge
→ It is simplest way to represent the facts.
→ It uses a table (like a database) where row represents objects and columns represent their attributes
→ It stores declarative fact as a set of relations.

Eg:

| Player | Height | Weight | Bats - Throws |
|---|---|---|---|
| A | 6-0 | 180 | Right - Right |
| B | | | |
| C | | | |

2) Inheritable Knowledge
→ Organizes knowledge in a hierarchy
→ Knowledge elements are grouped into "classes" (categories) and "instances" (specific objects)
→ The instances can "inherit" properties from the classes they belong to




3) Inferential Knowledge

→ This method represents knowledge using formal logic and rules

→ It uses the rules of logic to generate new information from the facts it already has

Example

All dogs have tail : $\forall x : dog(x) \rightarrow hastail(x)$

→ In logic, this is written as: For all X, if X is a dog, then X has a tail

→ If the system learns a new fact, "Fido is a dog" it can use the rule to automatically infer the new fact "Fido has a tail".

4) Procedural Knowledge

→ This method represents knowledge as a set of instructions or procedures on how to do something

Example: A computer Program and a cooking recipe

## Challenges in Knowledge Representation

1. **Important Attributes**
   - **Challenge:** How do we handle basic properties that are very common, like "is a type of" or "is an example of"?
   - **Example:**
     - We need to show that "A Person **isa** Mammal" (a type of).
     - We also need to show that "John is an **instance** of Person" (an example of).
   - **Why it's hard:** These isa and instance links are special. They allow **inheritance** (meaning John automatically gets all the properties of a Person, like "has a nose"). The system must be built to handle this special type of reasoning.
2. **Relationships Between Attributes**
   - **Challenge:** How do we show that the attributes (properties) themselves are related to each other?
   - **Examples:**
     - **Inverses:** If John "plays for" Team_A, we need the system to also know the inverse: Team_A "has player" John.
     - **Value Rules:** We need to set rules for attribute values (e.g., a person's age

inverse: Team_A "has player" John.

- **Value Rules:** We need to set rules for attribute values (e.g., a person's age cannot be greater than their parent's age).
- **Single Value:** We need to say that some attributes can only have one value (e.g., a person only has one height).

3. **Choosing Granularity (Level of Detail)**
   - **Challenge:** We must decide how much detail to store.
   - **Example:**
     - **Too Simple (Low Detail):** If we store the fact spotted(John, Sue), it's easy. But the system can't answer the question, "Did John see Sue?".
     - **Too Complex (High Detail):** We could store all the parts of "spotting" (like looking, focusing_eyes, seeing). This is very powerful but uses lots of memory and is slow.
     - **The Trick:** We must find a balance, like adding a simple rule: spotted(X, Y) -> saw(X, Y).

## 4. Representing Sets of Objects

- **Challenge:** How do we represent facts about a whole *group* (a set) of objects, not just one?
- **Example:** The fact "There are more sheep than people in Australia."
- **Why it's hard:** This fact is not about one specific sheep. It's a property of the *entire set* of sheep. The system needs a way to store facts about sets.

## 5. Finding the Right Structure

- **Challenge:** When you have a lot of knowledge, how do you find the right facts for a new situation?
- **Example:** If someone mentions a "bat," how does the system know if they mean a "baseball bat" or a "flying mammal bat"?
- **Why it's hard:** The system needs a good way to:
    - Select the right starting structure ("baseball bat" or "animal").
    - Fill in the details from the situation.
    - Know how to find a better structure if the first one is wrong.

Q-3 Explain the different issues in Knowledge Representation

Ans
1) Important Attributes
2) Relationships between Attributes
3) Choosing Granularity (Level of Detail)
4) Representing Sets of Objects
5) Finding the Right Structure

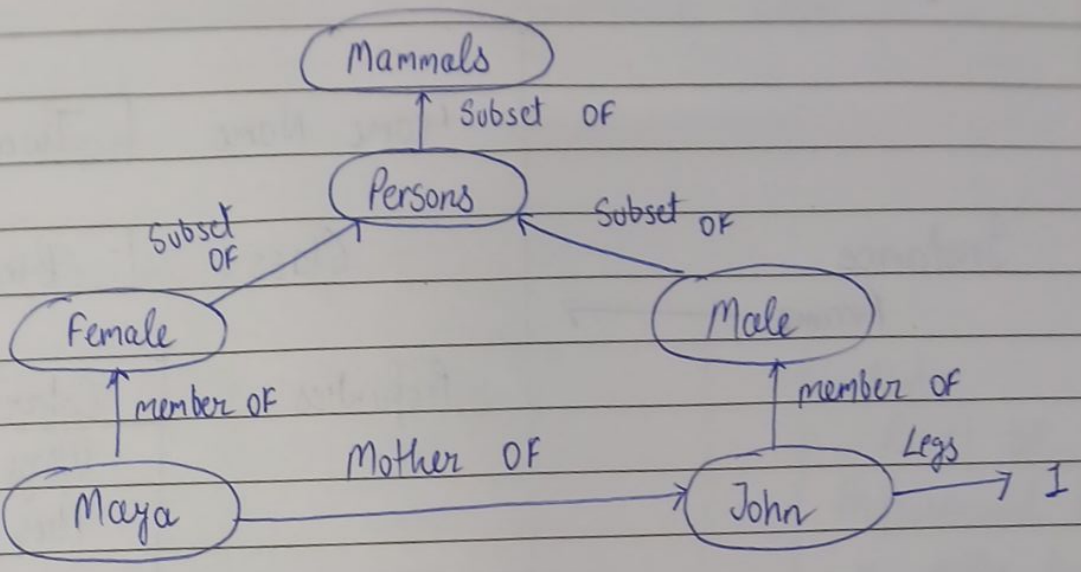
Q-4 Define Frames. Draw Semantic Net for following statements
a) Every kid likes candy b) Every school going kid likes candy OR Explain Semantic net & Frames with example

Ans Semantic Net
→ A Semantic Net is a way to represent knowledge as a graph. It's one of the "slot-and-filler" structures
→ They consists of structures (like objects or concepts) that have slots (attributes) which are filled with values.

o Components:
→ Nodes (Circles/Boxes) : These represents objects, concepts, or events (eg Person, Brooklyn)

→ Arcs (Arrows/Links) : These are labeled arrows that shows the relationship between the nodes (eg isa, instance, Team)

Eg:




- Frames
  Frame is a collection of attributes or slots and associated values that describe some real-world entity
  → A frame is like a template or form that describes a typical object or situation. It is bundle of attributes (slots) and their values

- Components of Frames
  → Slots: These are the attributes or properties of the object (like the fields on form eg Color, Wings)
  → Values (Fillers): This is the information that goes into a slot (eg Yellow, 2).

Eg:

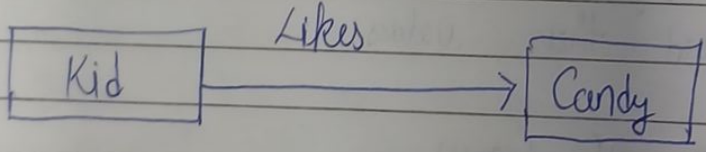
| Frame name | Bird | ← Class Frame |
|---|---|---|
| Properties | Color: Unknown<br>Wings: 2<br>Flies: True | |

Instance Frame →

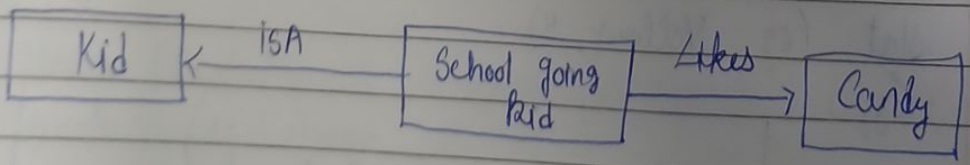
| Frame Name | Tweety |
|---|---|
| Class: | Bird |
| Properties | Color: Yellow |
| | Wings: 1 |
| | Flies: False |

Semantic Net For

a) Every Kid likes Candy




b) Every school going kid likes candy




Q-5 Difference between Prohositional logic and Predicate Logic

Ans See Image

Q-6 Discuss Limitations of Prohositional Logic

Ans See Image

O Prohositional Logic

| Symbols | Words |
|---|---|
| ⌐ | not |
| ∧ | and |
| ∨ | or |
| → | inflies (if then) |
| ←→ | iff |
| ∀ | for all |
| ∃ | there exists |

Note:
Result of Prohositional logic is either true or False

X: It is cold
Y: It is Sunny
Z: It is breezy

1) It is not cold
⇒ ⌐X

2) It is cold and it is breezy
⇒ X ∧ Z

3) If it is breezy and cold then it is not Sunny
⇒ Z ∧ X → ⌐Y

4) It will be cold iff it is Breezy
→ X ←⇒ Z

Here's the **difference**:

| Feature | Propositional Logic | Predicate Logic (First-Order Logic) |
|---|---|---|
| Basic unit | Whole statements (propositions) | Statements with **variables**, **predicates**, **functions** |
| Example statement | "It is raining" | "Raining(x)" → depends on x (place/person) |
| Can express relationships? | ❌ No | ✅ Yes (e.g., "All humans are mortal") |
| Uses quantifiers? | ❌ No | ✅ Yes — ∀ (for all), ∃ (there exists) |
| Level of expressiveness | Less expressive | More expressive |
| Typical use | Boolean logic, circuits, SAT problems | AI, databases, reasoning, mathematical proofs |

**In short:**

- **Propositional logic** deals with simple true/false statements.
- **Predicate logic** expands them using **variables + quantifiers** to express more complex facts.

If you *actually* meant something else, tell me — I'll explain! ✅

**Limitations of Propositional Logic**

The main problem with propositional logic is that it is **not very expressive**. It cannot represent real-world knowledge in a detailed way.

- **Cannot See Inside Sentences:** It treats an entire statement (like "Marcus is a man") as a single, unbreakable symbol (e.g., P). It cannot break it down to talk about the object ("Marcus") or its property ("is a man").
- **Cannot Represent Generalizations:** Because it can't talk about objects or properties, it cannot represent general rules like "All men are mortal" or "All Pompeians are Romans."

Q-6 Discuss Limitations of Prohositional Logic

Ans See Image

Prohositional Logic

| Symbols | Words |
|---|---|
| ⌐ | not |
| ∧ | and |
| ∨ | or |
| → | implies (if then) |
| ←→ | iff |
| ∀ | for all |
| ∃ | there exists |

Note:
Result of Prohositional logic is either true or false

X: It is cold
Y: It is sunny
Z: It is breezy

1) It is not cold
→ ⌐X

2) It is cold and it is breezy
→ X ∧ Z

3) If it is breezy and cold then it is not sunny
→ Z ∧ X → ⌐Y

4) It will be cold iff it is Breezy
→ X ←→ Z

# Predicate Logic (First order logic / FOPL (First-Order Predicate Logic)

Components of Predicate Logic

1) Objects / Constants :- 5, John, 6, 7
2) Variables :- x, y
3) Predicates :- Represents Relationship betwn objects
   Eg John loves Sophy  love (John, Sophy)

4) Quantifiers
   1) Universal Quantifier (∀) :- For all, For each, Everything, any thing, For every

   2) Existential Quantifier (∃) :- there exists, For some, atleast once

5) Connectives
   And (∧), OR (∨), ¬ (negation), condition (→), biconditional (↔)

Eg:

1) Marcus was a man : man (marcus)

2) All Pompeian were romans ∀x : Pompeian(x) → Roman(x)

3) Caesar was a ruler : ruler (caeser)

4) All romans were either loyal to caeser or hated him

∀x : Romans(x) → loyalto (x, caeser) ∨ hate (x, caeser)

5) Every one is loyal to someone
$\forall x : \exists y : loyalto(x, y)$

6) People only try to assasinate rulers they are not loyal to

$\forall x : \forall y : People(x) \wedge Ruler(y)$

Q-7 W-24 Q-3(b)

1) Some students took English subject
$\exists x : (Student(x) \wedge took(x, English))$

2) Every student who takes English passes it
$\forall x : (Student(x) \wedge took(x, English)) \rightarrow passes(x, English)$

3) Every person who buys policy is a smart

$\forall x : (person(x) \wedge buys(x, policy)) \rightarrow smart(x))$

4) No person buys an expensive policy

$\neg \exists x, y : (person(x) \wedge Policy(y) \wedge Expensive(y) \wedge buys(x, y))$

Q-1 S-24 Q-3(a)

1) John like all kinds of food
$\forall x : (food(x) \rightarrow likes(John, x))$

2) Apples are food
$\forall x$; food (Apples)

3) Chicked is food
food. (Chicken)

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

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

5) Bill eats peanuts and is still alive

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

6) Sue eats everything bill eats

$\forall x : [Eats(Bill, x) \rightarrow Eats(Sue, x)]$

Q-3 What is clausal form? How it is useful [4M, W-23]

Ans Clausal form is a standard, simple way of writing logical statements so a computer can understand them
→ A "clause" is a single statement where all the facts (literals) are connected only by the OR symbol
→ A full knowledge base in clausal form is just a list of these seperate clauses. The system assumes all the clauses are connected by AND

- How it is useful?
→ Its only purpose is to make the Resolution algorithm work
→ Resolution is a powerful AI methods for proving things. It is very efficient, but it requires that all the logical statements are converted into the strict clausal form before it can start

Q-9 Explain Inference Rules in Propositional Calculus

Ans An inference rule is a logical rule used to find (or "infer") new, true statements from a list of existing true statements
→ The goal is to allow a computer to "reason" and create new knowledge from old knowledge it was given
→ Modus Ponens is most famous inference rule

Modus Ponens
→ Basic and common rule of logical Inference
It states that if you know two things are true:
1) A statement A
2) A rule $A \rightarrow B$ (which means "If A is true, then B is true")
→ Then, you can logically conclude that statement B must also be true
→ In AI, instead of using ~~means~~ many different rules, it is more efficient to use a single, powerful rule that can do job of all the others
→ This one powerful rule is called Resolution. Resolution is a complete inference rule, meaning it can be used to prove any statement true

Q-10 Write and Explain algorithm for resolution in propositional logic with an example

Ans Algorithm
1) Convert all the propositions to clause form

$A \rightarrow B = \neg A \vee B$

$A \leftrightarrow B = (A \rightarrow B) \wedge (B \rightarrow A)$

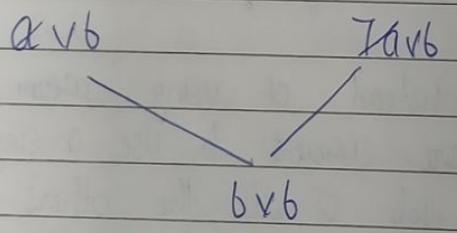
2) Negate the proof & convert the result to clause form.

3) Repeat until a contradiction is found

a) Select 2 clauses. One clause contains a literal 'L' & another clause contains a literal '¬L'. Call these as parent clauses

Eg: $a \vee b$ ('L') & $\neg a \vee b$ ('¬L')

(b) Resolve them together. The resulting clause called as ~~redundant~~ resolvent which will be distinction OR of all the literals in L & ¬L (OR operation)




(c) If Resolvent is empty then a Contradiction is found

Example

$P, (P \wedge Q) \rightarrow R, (S \vee T) \rightarrow Q, T$

prove that R is true

1) P

2) $(P \wedge Q) \rightarrow R$

$\rightarrow \neg(P \wedge Q) \vee R$

$\rightarrow (\neg P \vee \neg Q) \vee R$

$\Rightarrow (\neg P \vee \neg Q) \vee R$

2) $(S \vee T) \rightarrow Q$

$\Rightarrow \neg(S \vee T) \vee Q$

$\Rightarrow (\neg S \wedge \neg T) \vee Q$ [Different Symbol then distributive]

$\Rightarrow (\neg S \vee Q) \wedge (\neg T \vee Q)$

$\Rightarrow (\neg S \vee Q) \wedge (\neg T \vee Q)$

3) a) $\neg S \vee Q$

4) b) $\neg T \vee Q$

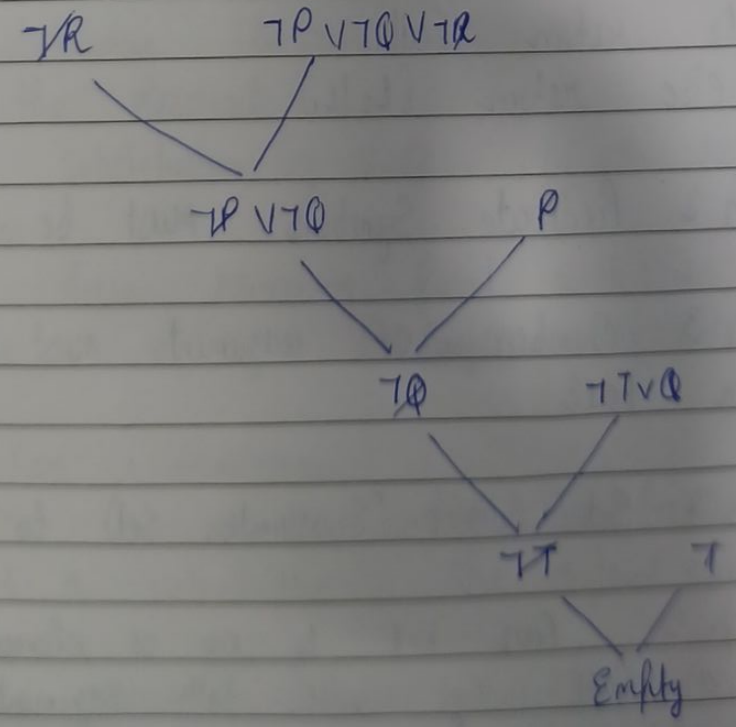
4) T

R is Proof

Step:2 Negate the Proof

$\Rightarrow \neg R$

Step:3



Q-11 Explain Unification in Predicate logic

Ans → Unification is a "matching" process
→ In simple propositional logic, it's easy to find opposites : rain and ¬Rain (Not rain).
→ In predicate logic it is harder
The statements like (John, x) and ¬Likes (John, Apples) are not an exact match
→ Unification is algorithm that finds a set of substitutions (like let "x = Apples") to make the two statements identical
→ Thus "matching" step is required before the Resolution algorithm can be used in predicate logic

Unification Algorithm
Step 1: If $L_1$ & $L_2$ are both variables or constants
a) If $L_1$ & $L_2$ are identical then return NIL
b) Else If $L_1$ is a variable then is $L_1$ occurs in $L_2$ return Fail
else return $(L_2/L_1)$

Step 2 Predicate Symbols must be same otherwise return File

Step:3 Number of arguments must be same otherwise return Fail

Step:4 Set SUBST (Substitution Set) to NIL

Step:5 For i:1 to no of elements in $L_1$
a) Call unify with $i^{th}$ argument of $L_1$ & $i^{th}$ argument of $L_2$ and put result in S

4) If S ≠ Nil do
(i) Apply S to remainder of both L1 L2
(ii) SUBST := APPEND (S, SUBST)
Return SUBST

Algorithm for Resolution of Predicate Logic

1) Convert all facts and rules into clause Forms
2) Negate the Goal : take statement that you want to prove and add ¬(NOT) to it.
3) Add this negated goal clause to your list of clauses
4) Start a loop
a. Find two "parent" clauses that have parts that can be made opposite by unification
Eg: (¬EasyCourse(x) ∨ Likes (Steve, x)) and (¬Likes(Steve, BK301))
b. Unify those two parts
Eg: Unify Likes (Steve, x) and Likes (Steve, BK301) to get the substitution (BK301/x)
c. Create the Resolvent:
→ Apply substitution (eg x=BK301) to the rest of the parent clauses
→ Combine these remaining part with ∨
→ Eg: The resolvent is (¬EasyCourse(BK301))

d. Check for Contradiction
→ If the resolvent is empty clause (nothing left), you found a contradiction. The original goal is True

e. Add new resolvent clause to list and continue the loop
5) If loop stops because no new clauses can be made, the goal cannot be proven

Q-6 FW S-24 Q-4(c)

(1) Steve only like easy courses

$\forall x : [EasyCourse(x) \rightarrow Likes(Steve, x)]$

(2) Science courses are hard

$\forall x : [ScienceCourse(x) \rightarrow \neg EasyCourse(x)]$

(3) All the courses in haveFun deptt. are easy)

$\forall x : [In(x, HaveFunDeptt) \rightarrow EasyCourse(x)]$

(4) BK301 is HaveFun Deptt Course

$In(BK301, HaveFunDeptt)$

Step:2 Convert Logic into Clause Form

1) $[EasyCourse(x) \rightarrow Likes(Steve, x)]$

$\Rightarrow \neg EasyCourse(x) \vee Likes(Steve, x)$

2) $[ScienceCourse(x) \rightarrow \neg EasyCourse(x)]$

$\Rightarrow \neg ScienceCourse(x) \vee \neg EasyCourse(x)$

3) $[In(x, HaveFunDeptt) \rightarrow EasyCourse(x)]$

$\Rightarrow \neg In(x, HaveFunDeptt) \vee EasyCourse(x)$

4) $In(BK301, HaveFunDeptt)$

Step : 3

Goal: What course would Steve like

G: likes (Steve, BK301)

A: ¬G: ¬likes (Steve, BK301)

Replace x with BK301

Clause 1 ¬EasyCourse(x) ∨ Likes (Steve, x) ¬likes (Steve, x)

Clause 3

¬EasyCourse(x) ¬In (x, HaveFunDept) ∨ ~~EasyCourse(x)~~

¬In (x, HaveFunDept) In (x, HaveFunDept)

Empty clause().

Conclusion:

Steve would like BK301

Q-13

1) Raj likes all kind of Food

$\forall x : (Food(x) \rightarrow Likes(Raj, x))$

2) Apples are Food

Food (apples)

3) Anything anyone eats and isn't killed by food

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

4) Sachin eats peanuts and is still alive

$Eats(Sachin, Peanuts) \wedge \neg Killedby(Sachin, Peanuts)$

5) Vinod eats everything Sachin eats

$\forall x : (Eats(Sachin, x) \rightarrow Eats(Vinod, x))$

1) $x : Food(x) \rightarrow Likes(Raj, x)$

$\neg Food(x) \vee Likes(Raj, x)$

2) Food(apples)

3) $\neg East(x,y) \vee Killedby(y,x) \vee Food(x)$

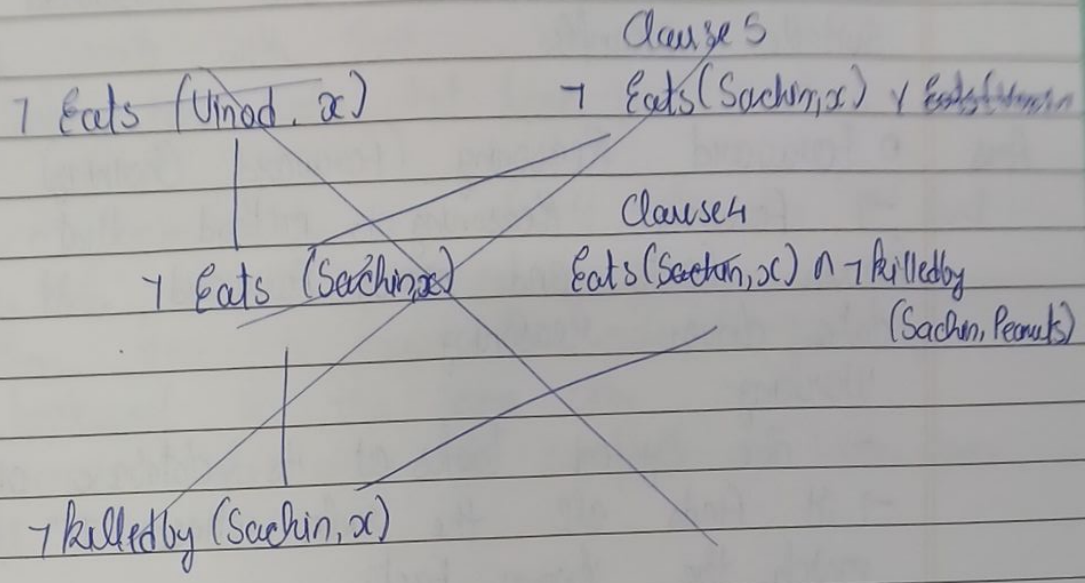
4) $Eats(Sachin, Peanuts) \wedge \neg Killedby(Sachin, Peanuts)$

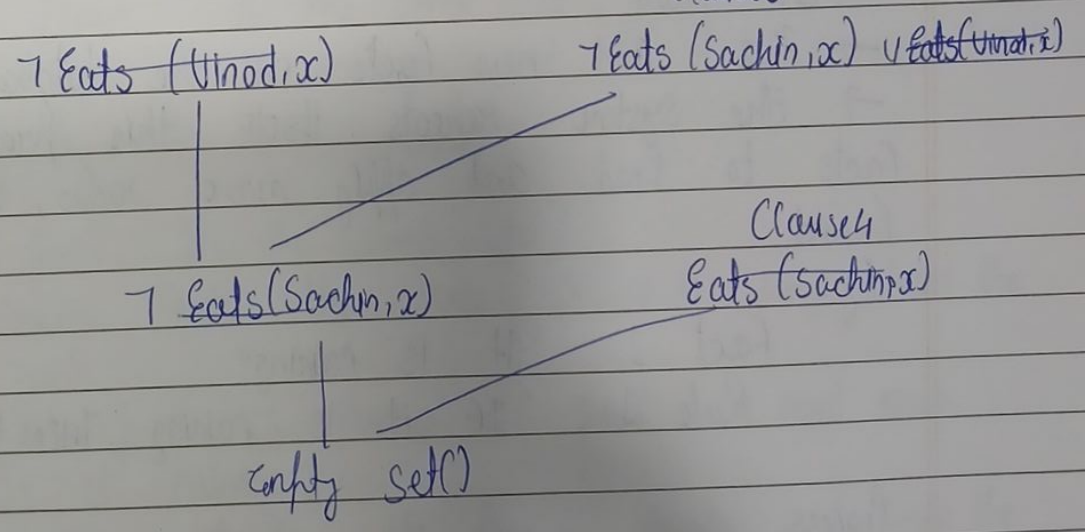
5) $\neg Eats(Sachin, x) \vee Eats(Vinod, x)$

Goal: Eats (Vinod, Peanuts)

A: $\neg$ Eats (Vinod, Peanuts)

x: Peanuts




Conclusion

Vinod will eats Peanuts

Q-14 Declarative Knowledge v/s Procedural Knowledge

Ans See Image

## 🆚 Quick Comparison

| Feature | Declarative Knowledge | Procedural Knowledge |
|---|---|---|
| Meaning | Knowing **what** | Knowing **how** |
| Expressed as | Facts, statements, rules | Skills, steps, procedures |
| Easy to explain? | ✅ Yes | ❌ Usually no |
| Learned by | Reading, listening | Practice and repetition |
| Stored as | Information | Action sequences |
| Example in AI | Knowledge base facts | Inference rules / algorithms |

Q-15 What is Forward and backward reasoning with suitable examples

Ans
- Forward Reasoning (Forward Chaining)
→ Forward Reasoning is method that starts with the known facts and moves forward. It is also known as data driven reasoning

Working:
→ The system looks at its database of known facts
→ It finds all the rules whose "IF" parts (conditions) match the known facts
→ It applies these rules, which means it performs the "THEN" part
→ This add new facts on the database
→ The system repeats itself this process, using the new facts to find and apply more rules, until goal is found

Eg:
Fact 1: "It is raining"
Rule 1: "IF it is raining, THEN the street is wet"

Process
1) The system starts with It is raining
2) It finds rule 1, and the "IF" part matches fact
3) It applies the rule and adds the "THEN" part as a new fact
4) NEW FACT 2: The street is wet (might be the goal)

- Backward Reasoning (Backward Chaining)
→ Backward chaining is a method that starts with the goal and works backward. Goal driven reasoning.

Working
→ System starts with goal
→ It looks for rules that have goal in their "THEN" part
→ Which it finds such a rule, it takes the "IF" part of that rule as its new sub goal
→ It then tries to prove this new sub-goal by working backward in the same way
→ The process stops when all sub-goals are broken down into original known facts

Example
Goal to prove: "Is the street wet"?
Known facts: It is raining and IF it is raining, THEN the street is wet

Process
1) Start goal : Is the street wet?
2) The system finds the rule: IF it is raining, THEN the street is wet
3) New Sub-goal: To prove the goal, it must now prove : Is it raining?
4) The system checks its known facts and finds the fact : It is raining
5) The sub-goal is proven so the original goal is also proven

Q-16 Difference between forward & backward reasoning

Ans See Image

## ✅ Forward vs Backward Reasoning (Aspect-wise Table)

| Aspect | Forward Reasoning | Backward Reasoning |
|---|---|---|
| **Starting Point** | Begins with known **facts** | Begins with a **goal/hypothesis** |
| **Direction** | Facts → Conclusion | Conclusion → Supporting facts |
| **Control Strategy** | **Data-driven** | **Goal-driven** |
| **Rule Matching** | Uses rules whose **IF** part matches facts | Uses rules whose **THEN** part matches goal |
| **Search Space** | Broad — explores many outcomes | Narrow — checks only relevant rules |
| **Best Used When** | Goal is **unknown** | Goal is **known** and specific |
| **Typical Application** | Diagnosis, monitoring systems | Prolog, query answering, theorem proving |
| **Efficiency** | Can be **slower** due to many rule firings | Usually **faster** for single goals |
| **Memory Usage** | **High** (stores new derived facts) | **Low** (recursive proof checking) |
| **Example** | Symptoms → identify disease | Verify disease → check symptoms |

www.youtube.com – to exit full screen, press esc

# Forward & Backward Chaining

- Data driven
  - Data is available
- Goal driven
  - Goal state is given

| data | Rules | Conclusion |
| --- | --- | --- |
| X = 1 | If (X==1 & Y==2) | a = 4 |
| Y = 2 | Then Z=3; | |
| | If (Z==3) | |
| | then a=4 | |

# Ch-4 Symbolic Reasoning under Uncertainty

Q-1 What is non monotonic reasoning? Explain logics for nonmonotic reasoning

Ans It is a type of reasoning where you can take back a conclusion if you learn new information

Eg:

Step: 1 You know 2 facts
Fact 1: "Birds can fly"
Fact 2: "Alex is bird"

Step: 2 You make a default conclusion: "Alex can fly"

Step: 3 You learn a new fact: "Alex is a penguin"

Step: 4 : This new fact contradicts your default assumption. Your reasoning is "non monotonic" because you take back the old conclusion

Step: 5 Your new, correct conclusion is: "Alex cannot fly"

Logics for Non-Monotic Reasoning
1) Default logic
2) Non Monotonic logic

1) Default logic
→ This is a very common way to handle non-monotonic reasoning
→ It creates a new type of "default rule" that looks like this

A : B|C

⇒ A ( Prerequisite ) : IF A is true

⇒ B ( Justification ) : AND it is consistent (believable) to assume B is true

⇒ C ( Consequent ) : THEN you can conclude C is true

Eg :

bird (x) : Flies(x) : Flies(x)

"If x is a bird, AND it is consistent to assume x flies, THEN conclude x flies"

2) Non-monotonic Logic (NML)

→ This logic extends regular logic with a special modal operator M

→ Symbol M means "is consistent with everything we know".

Eg :

$\forall x : Car(x) \wedge M \neg Broken \rightarrow Starts(x)$

Meaning :

→ For any car x, IF x is a car, AND it is consistent to assume x is NOT broken.

→ THEN we can conclude that x starts"

If we later add Broken (myCar)

then it contradicts with above rule and it means we must take it back the conclusion starts (myCar)

# Ch-5 Probabilistic Reasoning

Q.1 Explain Probability and Bayes theorem

Ans Probability:

It is simply a way to measure how likely something is to happen

It's a number between 0 to 1

⇒ If P=0, event is impossible

⇒ If P=1, event is certain to happen

Types

→ Marginal Probability: the probability of a single event happening

Eg: Probability of rolling an 6 on die, written P(roll a 6)

→ Joint Probability: the P of two or more events happening at same time

Eg: getting a tail & rolling a dice

→ Conditional Probability: The Probability of one event (A) happening, given that another event B has already happened

→ Key idea of Bayes theorem

→ It is written as P(A|B) which you red as read as "the probability of A given B"

Q Baye's Theorem

Its main job is to update a belief based on new evidence. It connects a conditional probability (like P(A|B) with its reverse (like P(B|A))

In simple terms
"What is the probability that my belief (A) is true, given that I just saw this new evidence (B)?"

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

Q. A: Bg I: 4W and 6Black balls
B: Bag 2: 4W & 3 Black balls
E: the event that the ball we drew is black
$P(A) = 1/2$
$P(B) = 1/2$

ball drawn from random at from any one bag

$$P(A|E) = \frac{P(E|A) \times P(A)}{P(E)}$$

$P(E|A)$ = (Probability of black, given bag I) = 6/10

$P(E|B) = 3/7$

$$P(E) = P(E|A) \times P(A) + P(E|B) * P(B)$$
$$= \frac{6}{10} \times \frac{1}{2} + \frac{3}{7} \times \frac{1}{2}$$
$$= \frac{6}{20} + \frac{3}{14}$$
$$P(E) = \frac{36}{70} = \frac{18}{35}$$

$$P(A|E) = \frac{6/10 \times 1/2}{36/70}$$
$$= \frac{6^3 \times 70}{20 \times 36\ 12}$$
$$P(A|E) = \frac{7}{12} = 58.3\%$$
$$= 0.583$$

Q-2 State the Baye's theorem. Illustrate how a Bayesian Network can be used to represent causality between relationship among attributes

Ans Bayesian Network

A Baysian Network is a diagram that is used to model uncertainity

→ Its a probabilistic tool model that shows a set of variables (events) and their conditional dependencies (how they influence each other)

→ It's a powerful tool for reasoning about a situation when you don't have all the facts.

How Bayesian Network Represent Causality (Cause and effect)

→ A Bayesian network uses a diagram with nodes (circles) and one-way arrows to show cause-and-effect relationships

- Nodes (Circles): Each node represents a variable or event (eg. "Rain", "Grass Wet")
- Arrows (Edges): An arrow from one node to another means the first node directly causes or influences the second node

The graph shows "Conditional dependencies" which means a node is only dependent on its direct parents (the nodes pointing to it)

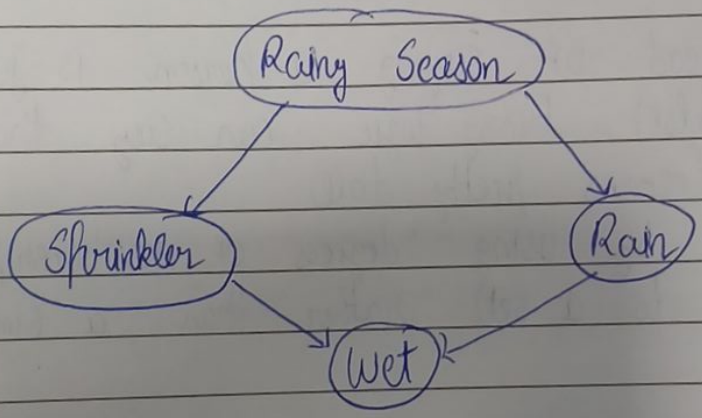
Eg: The "Wet Grass" Problem

- Events:
  - → Is it the "Rainy Season"?
  - → Did the "Sprinkler" turn on?
  - → Did it "Rain"?

→ Is the "Grass Wet"?

- Causality (Cause-and-Effect)
  1) The "Rainy Season" influences the probability of both rain & sprinkler use.
  (eg if it's the rainy season, it's more likely to rain and less likely you'll use the sprinkler)

2) Both "Rain" and the "Sprinkler"




Q. Discuss Bayesian network and its application

Ans
1) Medical Diagnosis
2) Sham Filtering
3) Anomaly Detection
4) Forecasting
5) Diagnostics

See Image

Q.4 What is Certainty Factor ?

Ans
MB (Measure of belief)
MD (Measure of Disbelief)
CF: MB - MD

**Applications of Bayesian Networks**

Bayesian Networks are used in many real-world systems that need to reason with uncertain information:

- **Medical Diagnosis:** Calculating the probability of a patient having a specific disease given their symptoms.
- **Spam Filtering:** Determining the probability that an email is spam based on the words it contains.
- **Anomaly Detection:** Identifying unusual activity (like fraud) that doesn't fit normal patterns.
- **Forecasting:** Predicting things like stock market movements or weather.
- **Diagnostics:** Figuring out why a machine or system has failed.

Q. What is certainty factor? **3M W21**

---

A **Certainty Factor (CF)** is a number that represents a numerical estimate of the belief or disbelief in a conclusion, especially when there is a set of evidence.

- It is a way for expert systems (like MYCIN) to handle uncertainty without using complex probabilities.
- It is defined by two components:
  - **MB (Measure of Belief):** A number (from 0 to 1) that shows how much the evidence supports the hypothesis.
  - **MD (Measure of Disbelief):** A number (from 0 to 1) that shows how much the evidence supports the *opposite* (negation) of the hypothesis.
- The final Certainty Factor is calculated by subtracting the disbelief from the belief:
  - **CF = MB - MD**

0.2 - 0.1

=

Q-5 Explain Fuzzy logic

Ans Fuzzy logic is a method of reasoning that is not limited to just "True" or "False"

→ Traditional logic (Crisp logic) uses exactly two^ values : 1(true) or 0(false)

→ Fuzzy logic can use any real number between 0 and 1

→ This allows it to handle the concept of "partial truth"

→ Eg: Instead of saying a person is just "Tall" (True) or "Not tall" (False), Fuzzy logic can say that they are 0.8 "tall" (means pretty tall)

→ It works by using "degrees of membership" (how much something belongs to a set) rather than a simple yes-or-no membership

Q-6 Compare Fuzzy logic and Crisp logic

Ans See Image

Q-7 Explain Membership Function & Linguistic Variable

Ans → A linguistic variable is a variable whose values are words or sentences from a natural language, instead of just numbers

→ A Membership Function is a rule or curve that defines what a linguistic value (like "Tall" or "Warm") actually means

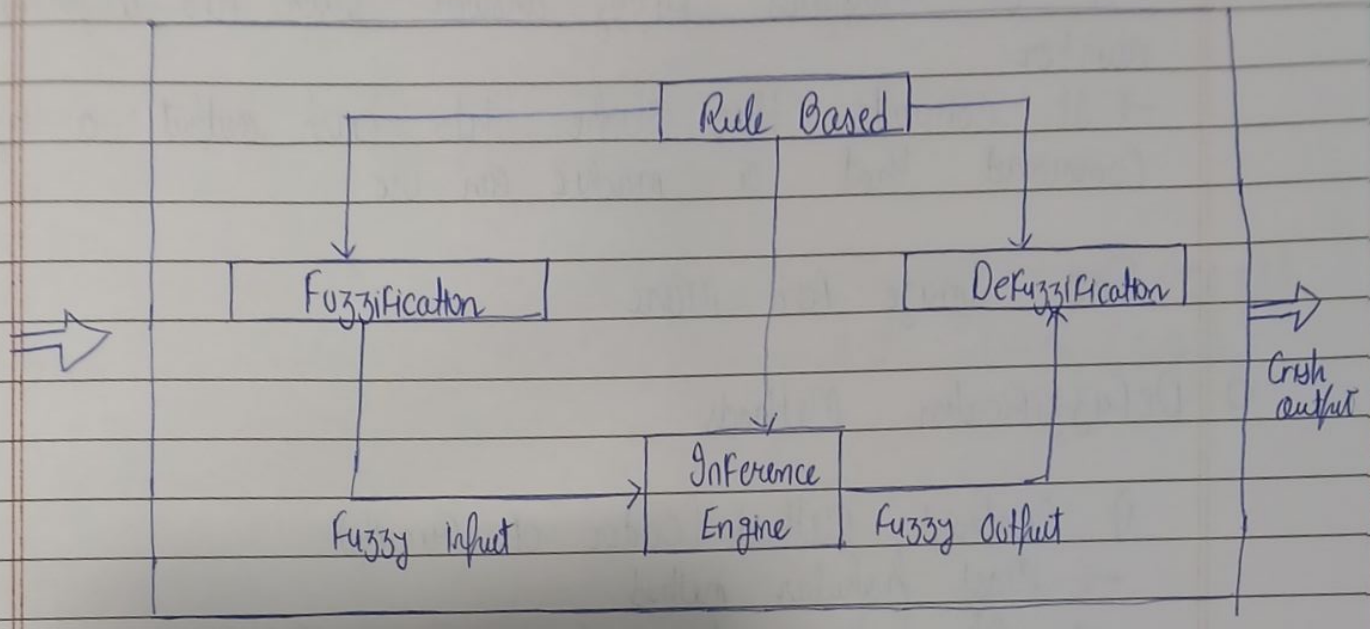
See Image For More

## ✅ Difference Between Fuzzy Logic and Crisp Logic

| Aspect | Crisp Logic | Fuzzy Logic |
|---|---|---|
| **Nature of Values** | Only **0 or 1** (true/false) | Values range from **0 to 1** (degrees of truth) |
| **Type of Reasoning** | **Binary / exact** | **Approximate / flexible** |
| **Output** | **Definite** yes/no result | **Partial**/gradual result |
| **Handling Uncertainty** | Cannot handle **imprecision** | Designed to handle **uncertainty & vagueness** |
| **Membership** | **Crisp sets:** either in or out | **Fuzzy sets:** partial membership |
| **Example Statement** | "Temperature is hot" → **true or false** | "Temperature is hot" → **0.7 true**, 0.3 false |
| **Decision Making** | Strict rules only | Human-like reasoning |
| **Mathematical Basis** | Classical Boolean logic | Fuzzy set theory |
| **Real-World Use** | Digital circuits, database queries | ACs, washing machines, AI control systems |
| **Suitability** | Works for **precise systems** | Works for **real-world uncertain systems** |

Q.8 Explain Different defuzzification methods

Ans




Architecture of Fuzzy logic System

The 4 Components of architecture

1) Fuzzification (Input)
   Crisp Input ⟹ Fuzzy input

2) Rule Base
   → Knowledge base
   → Collection of "IF-THEN" rules given by human expert

3) Inference Engine
   → Thinking Part that uses rules
   → It takes Fuzzy input (from step 1) and applies it to Rule Base (Step 2)
   → It combines the results of all rules into Fuzzy output. thus Fuzzy output is not a number, but a final combined SHAPE that represents the System's total Fuzzy decision

4) Defuzzification Output
→ Final step
→ It translates fuzzy answer back into a precise number
→ It converts this shape into crisp output: a single, precise command that a machine can use.

See Image for more

○ Defuzzification Methods

1) Centroid Method (Center of Gravity)
→ Most popular method
→ It looks at final fuzzy output shape and finds its "balance point"
→ The crisp number is the x-axis value of that balance point

2) Max Membership Method (Height method)
→ The simple method finds the "highest peak" of fuzzy output shape.
→ The crisp no. is the x-axis value at peak

3) Mean of Maxima Method
→ Use this if highest peak is a "flat top"
→ It takes all the x-values on that flat top and finds their average
→ The crisp no. is middle point

4) Weighted Average Method
→ This is a faster method
→ This does not combine the shapes

**Membership Function and Linguistic Variable**

**Linguistic Variable**

- A linguistic variable is a variable whose values are **words or sentences** from a natural language, instead of just numbers.
- *Example 1:* A variable named **"Temperature"** could have the values { "Cold", "Warm", "Hot" }.
- *Example 2:* A variable named **"Height"** could have the values { "Short", "Average", "Tall" }.

## Membership Function

- A membership function is a **rule or curve** that defines what a linguistic value (like "Tall" or "Warm") actually means.
- It takes a precise input number (like a specific height or temperature) and gives a fuzzy value (a "degree of membership" between 0 and 1).

- *Example:*
  - A membership function for **"Tall"** would define "how tall" a person is.
  - It might map a height of 152 cm to a value of **0.0** (not tall at all).
  - It might map a height of 181 cm to a value of **0.8** (pretty tall).
  - It might map a height of 208 cm to a value of **1.0** (definitely tall).

---

**The 4 Components of the Architecture**

- **1. Fuzzification (Input)**
  - This step takes a **Crisp Input** (a precise number) and translates it into a **Fuzzy Input** (an unclear concept).
  - **Crisp Input:** A sensor reads a real-world, precise number.
    - *Example:* Temperature is 29°C.
  - **Fuzzy Input:** The system checks its rules and decides how much that number belongs to an unclear group.
    - *Example:* 29°C is 0.7 "Hot" and 0.1 "Warm".
- **2. Rule Base**
  - This is the "knowledge base" of the system.
  - It is just a collection of "IF-THEN" rules given by a human expert.
  - *Example Rule:* IF temperature IS "Hot" THEN fan_speed IS "Fast".

- **3. Inference Engine (Thinking)**
  - This is the "thinking" part that uses the rules.
  - It takes the **Fuzzy Input** (from Step 1) and applies it to the **Rule Base** (Step 2).

---

  - It combines the results of all the rules into a **Fuzzy Output**. This **Fuzzy Output** is not a number, but a **final, combined SHAPE** that represents the system's total fuzzy decision.
  - *Example:* It combines the shape for 0.7 "Fast" with the shape for 0.1 "Medium" to make one new, lumpy **shape**.

- **4. Defuzzification (Output)**
  - This is the final step. It translates the fuzzy answer back into a precise number.
  - It takes the **Fuzzy Output SHAPE** (from Step 3).
  - It converts this **shape** into a **Crisp Output**: a single, precise command that a machine can use.
  - **Crisp Output:** A machine (like an AC fan) needs a precise command, not a fuzzy **shape**.
  - *Example:* The system finds the "balance point" of the fuzzy **shape** and outputs a single number: 85 (meaning "Set the fan speed to 85%").

→ It just takes the center point of each individual rules' output shape and calculates a weighted average based on each rule's "height"

Q-9 S-22 4M Sum

See Image For Question

$I' = \{(F, 0.4), (E, 0.3), (X, 0.1), (Y, 0.1), (I, 0.9), (T, 0.8)\}$

$F' = \{(F, 0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.5), (T, 0.5)\}$

Find

1. $I' \cup F'$

Union of max values

means $I' \cup F' = \{(F, 0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.9), (T, 0.8)\}$

2) $I' - F'$ (Mean Difference)

means Not $F'$

1-..

NOT $F' = \{(F, 0.01), (E, 0.2), (X, 0.9), (Y, 0.8), (I, 0.5), (T, 0.5)\}$

See Image For more

**Topic: Fuzzy Set Sum**

---

Q. The task is to recognize English alphabetical characters (F, E, X, Y, I, T) in an image processing system. Define two fuzzy sets I' and F' to represent the identification of characters I and F.

I'={(F,0.4), (E, 0.3), (X, 0.1), (Y, 0.1), (I, 0.9), (T, 0.8)},

F'={(F,0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.5), (T, 0.5)}.

Find the following.

1. I' U F'
2. I' – F' **4M S22**

---

First, let's establish the rules for these operations:

- **Fuzzy Union (U):** The Union of two fuzzy sets is found by taking the **MAXIMUM** membership value for each element from both sets.
- **Fuzzy Difference (-):** The Difference (I' - F') is found by finding the intersection of I' and the complement of F'.

First, let's establish the rules for these operations:

- **Fuzzy Union (U):** The Union of two fuzzy sets is found by taking the **MAXIMUM** membership value for each element from both sets.
- **Fuzzy Difference (-):** The Difference (I' - F') is found by finding the intersection of I' and the complement of F'.
    - Step 1: Find the complement of F' (which is 1 - degree(F')).
    - Step 2: Find the **MINIMUM** membership value between I' and the complement of F' for each element.

---

**1. I' U F' (Fuzzy Union)**

We take the **MAXIMUM** value for each character.

- **F:** MAX(0.4, 0.99) = **0.99**
- **E:** MAX(0.3, 0.8) = **0.8**
- **X:** MAX(0.1, 0.1) = **0.1**

- **I:** MAX(0.9, 0.5) = **0.9**
- **T:** MAX(0.8, 0.5) = **0.8**

Answer:

I' U F' = {(F, 0.99), (E, 0.8), (X, 0.1), (Y, 0.2), (I, 0.9), (T, 0.8)}

---

**2. I' – F' (Fuzzy Difference)**

First, we must find the complement of F' (which we can call "NOT F'").

The rule is Degree(NOT F') = 1 - Degree(F').

- NOT F' = {
  - (F, 1 - 0.99) = (F, **0.01**)
  - (E, 1 - 0.8) = (E, **0.2**)
  - (X, 1 - 0.1) = (X, **0.9**)
  - (Y, 1 - 0.2) = (Y, **0.8**)
  - (I, 1 - 0.5) = (I, **0.5**)
  - (T, 1 - 0.5) = (T, **0.5**)
  - }

# Ch-6 Game Playing

Q.1 Explain MiniMax Search Procedure

Ans Minimax is a search algorithm used in AI for making decisions in two-player games where the players have opposing goals (like Tic-Tac-Toe, chess or checkers)

→ It called "Minimax" because it an involves two players
- MAX (Maximizer): This is our AI. It always tries to make a move that leads to MAXimum possible score (a win)
- MIN (Minimizer): This is the opponent. We assume the opponent plays perfectly and will always try to make a move that leads to the MINimum possible scores (a loss for our AI).

MINIMAX Working

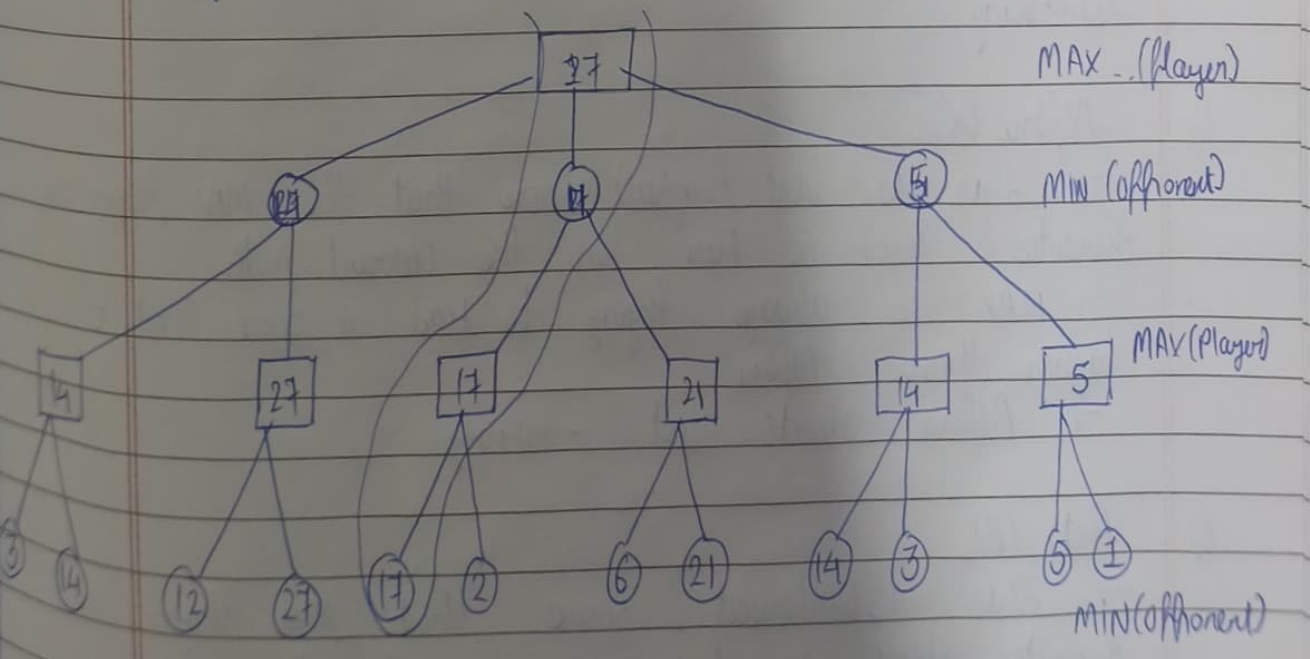
1) Generate the Game tree
→ Algorithm starts from current board state and explores all possible moves, then all possible replies to those moves and so on, down to depth or until game ends

2) Use a Scoring Function.
→ When it reaches end of search, It uses a static evaluation function (a scoring function) to give that board state a numerical score

3) Back Up the Scores
→ Most Important

The algo. works its way back up the tree from bottom
→ At MIN level (Opponent's Turn): The MIN player will choose the move that leads to lowest (minimum) score. So, score for this node is minimum of all its children scores
→ At MAX level (AI's turn): The MAX player will choose the move that leads to highest (maximum) scores. So, the score for this node is maximum of all its children's scores

4) Choose the Best Move: This process continues all the way back to the top (the current state). The AI then picks the move that leads to the node with the highest final "backed-up" score



Q-7 Explain Alpha Beta Cutoffs (Alpha Beta Pruning) with example

Ans Alpha Beta Cut-offs (or pruning) is an optimization technique that makes the Minimax algorithm much faster

Significance

Its purpose is to prune (cut off) large branches of the game tree that it knows are not good, without even looking at them

→ It can give the exact same answer as MiniMax but by searching only a fraction of game states

Working:

1) Alpha ($\alpha$):
→ This is best (highest) score that the MAX player can guarantee itself so far on the current path
→ MAX is always trying to find a score that is higher than Alpha
→ Alpha starts at -infinity

2) Beta ($\beta$)
→ This is best(lowest) score that MIN player can guarantee itself so far on current path
→ MIN is always trying to find a score that is lower than $\beta$
→ Beta starts at +infinity

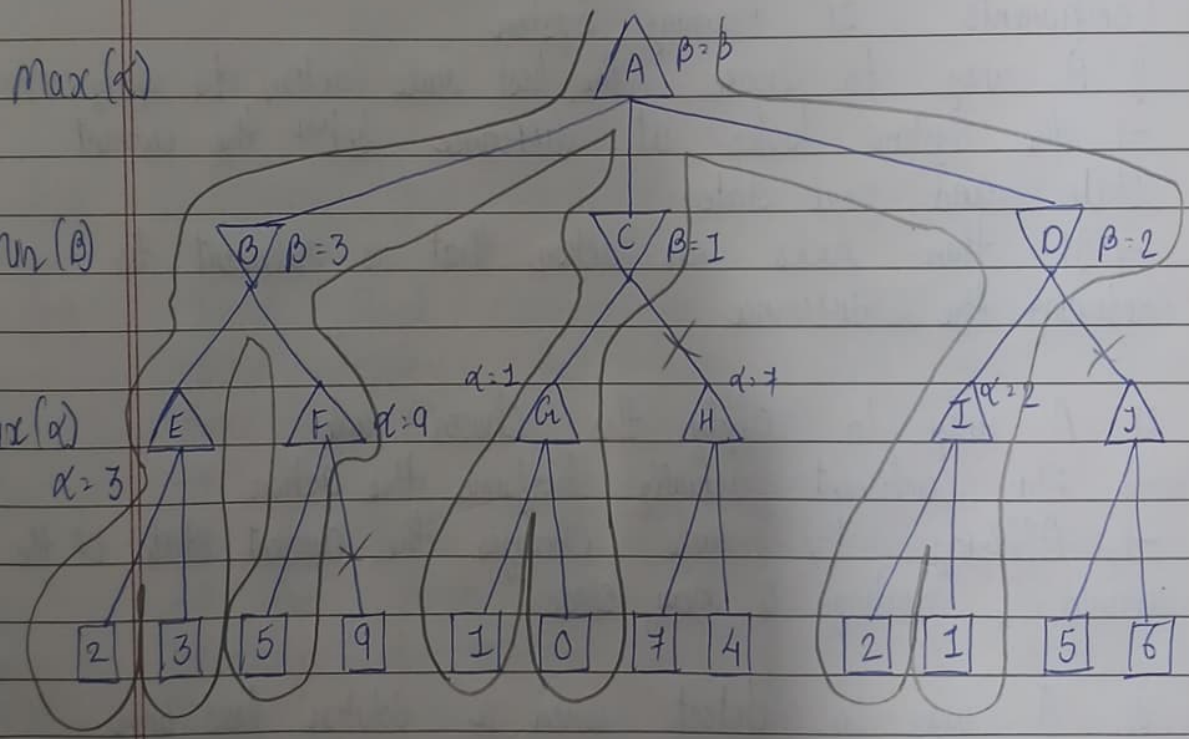
Cut-off Rule

Search stops exploring a node and all of its children (prunes the branch) as soon as:

$\alpha >= \beta$ (Alpha is greater than or equal to Beta)

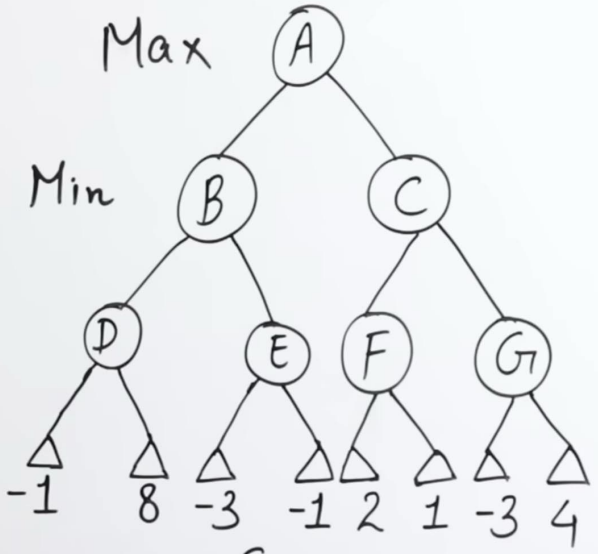
For more see Image

Example



# "Minimax Algorithm"

- ⇒ Backtracking algorithm
- ⇒ Best move strategy used
- ⇒ Max will try to maximize its utility ( Best Move)
- Min will try to minimize utility ( Worst move)




Game Tree

Like & Subscribe

on the current path.

- MIN is always trying to find a score that is **lower** than Beta.
- Beta starts at **+infinity**. +∞, ↓✓

**The Cut-off Rule**

The search stops exploring a node and all of its children (prunes the branch) as soon as:

**α >= β (Alpha is greater than or equal to Beta)**

1

3

- **Why? (MIN's perspective):** If MIN is exploring a move and finds a score that is *lower* than (or equal to) what MAX is already guaranteed (Alpha), MIN knows that MAX will *never* choose this path. So, MIN doesn't need to check any other moves from this node.
- **Ex. MIN's Perspective (The β <= α cut-off at Node C)**
- **A (MAX)** knows it can get a score of at least **3** (this is Alpha).
- **C (MIN)** finds a path that gives a score of **1** (this is Beta).
- **Cut-off:** C stops searching because its parent, A, will *never* choose this path (which gives a score of 1) over the path it already has (which gives a score of 3).

# Ch-7 Planning

Date
Page

Q-1 Explain Components of a Planning System

Ans A Planning problem in AI is task of finding a sequence of actions (called a plan) that will lead an AI from a starting state to a specific goal state

Components of Planning System

1) A way to choose the best rule (action) to apply next
→ The system looks at difference betwn the current state and goal state
→ It then picks an action that is relevant to reducing the difference

2) A way to apply the chosen rule
→ This component actually performs the action
→ Applying the rule changes the current state of the world, creating a new state

3) A way to detect when a solution has been found
→ The system needs to be able to check if the current state matches the goal state
→ This is how the system knows it has successfully finished the plan

4) A way to detect dead ends
→ The system must be able to realize when it is on path that can never lead to the solution
→ This allows the system to stop, backtrack and try a different path, saving time

5) A way to detect an almost correct solution

→ In complex problems, the system might solve smaller sub-problems separately.

→ Thus component checks if the solutions to the sub-problems can be combined without interfering with each other

Q-2 Difference between Search Planning and Search Procedure

Ans See Image

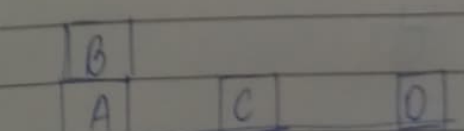
Q-3 Discuss goal Stack Planning in brief

Ans Goal Stack Planning is a method used by AI to solve problems that have multiple goals (e.g. "Goal A AND Goal B").

→ It uses a "stack" (a to-do list) to keep track of all the goals and actions it needs to finish.

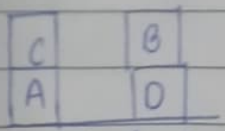
→ The stack works as "Last-In-First-Out (LIFO) list, meaning it always works on the last item that was added

Eg: Blocks World Problem




Initial State

ON(B,A) ∧ ONTABLE(C) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(C) ∧ CLEAR(B) ∧ ARMEMPTY



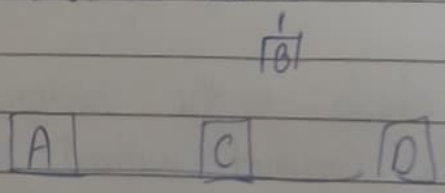

Goal State

ON(C,A) ∧ ON(B,D) ∧ ONTABLE(A) ∧ ONTABLE(D) ∧ CLEAR(C) ∧ CLEAR(B) ∧ ARMEMPTY

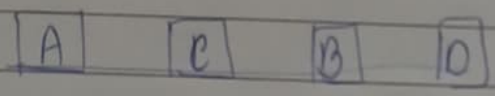
## ✅ Difference Between Planning and Search Procedure

| Aspect | Planning | Search Procedure |
|---|---|---|
| **Definition** | Generates a **sequence of actions** to achieve a goal | Explores **state space** to find a solution/path |
| **Focus** | **What actions** to perform and in what order | **Which state** to move to next |
| **Input** | Initial state, goal state, **actions with preconditions & effects** | Initial state, goal test, successor function, cost |
| **Output** | A **plan** (ordered steps) | A **solution path** from start to goal |
| **Representation** | Uses **planning languages** like STRIPS, PDDL | Uses **search trees/graphs** |
| **State Expansion** | Expands based on **applicable actions** | Expands based on **available successors** |
| **Knowledge Use** | Involves **domain knowledge** (preconditions, effects) | May or may not use knowledge (uninformed/informed) |
| **Complexity Handling** | Better for **large, structured problems** | Can become huge without heuristics |
| **Examples** | Robotics task planning, assembly scheduling | BFS, DFS, A*, hill climbing |
| **Goal Nature** | Often **multi-step, long-term** goals | Often **single goal reaching** |

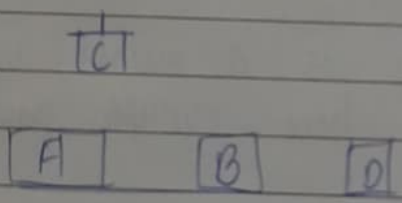
1) Unstack (B, A)



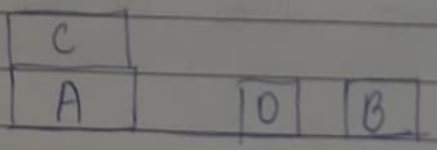

2) PUTDOWN (B)



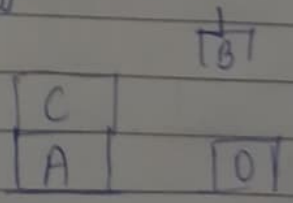

3) PICKUP (C)



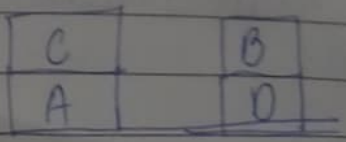

4) STACK (C, A)




5) PICKUP (B)




6) Stack (B, D)



Q-4 Explain Hierarchical Planning in brief

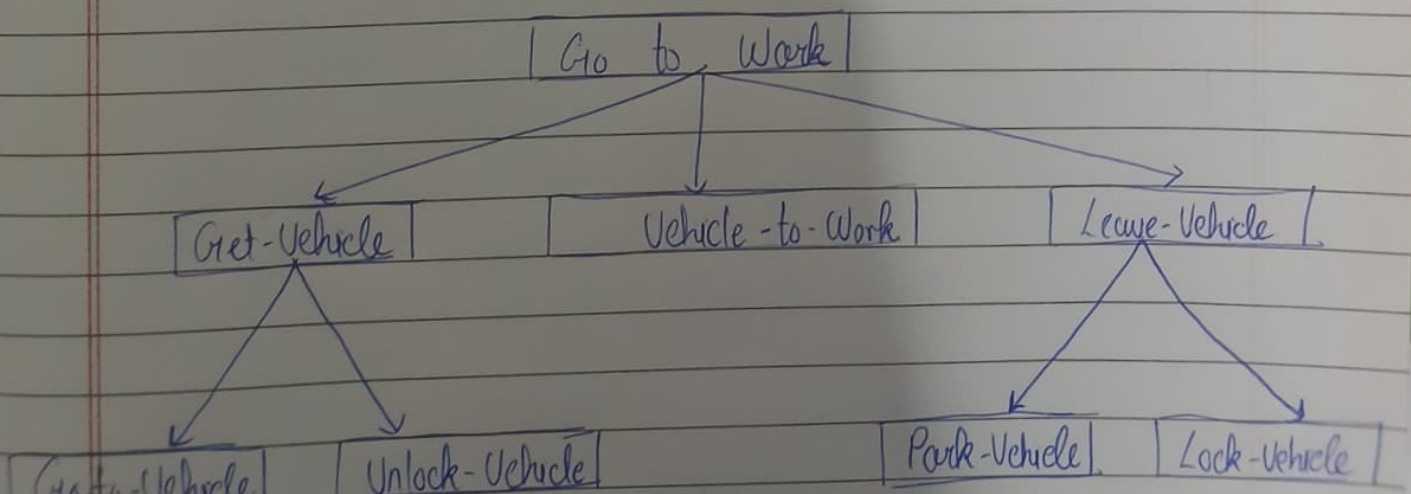
Ans Hierarchical Planning is a technique used to solve complex problems by first creating a simple, high-level plan and then filling the details later

Working:

→ The system first focuses only on most imp or "major-steps" needed to solve problem, ignoring small details this creates an abstract plan.

→ Once this high-level plan is complete, the system goes back to each major step

→ It then refines each major step by breaking it down into sequence of smaller, more detailed "minor steps"



# Ch-8 NLP

Q-1 What is NLP? Discuss applications of NLP in brief

Ans NLP is field of CS that bridges the gap between human language and computer understanding. It allows machines to process and analyze large volumes of text and speech data, enabling tasks like

→ Understanding intent and sentiment
→ Extracting information
→ Generating text
→ Translating text
→ Answering questions

Applications in NLP

1) Machine Translation
2) Chatbots and Virtual Assistant
3) Sentiment Analysis
4) Text Summarization
5) Information Extraction

Q-2 Phases of NLP

Ans See Image

Check NLP Full Package

1) Morphological Analysis
2) Syntactic " "
3) Semantic " "
4) Discourse Integration
5) Pragmatic Analysis

Q-3 State the factors which may make understanding of natural language difficult for a computer

Ans Factors

1) Ambiguity
   - → Word ambiguity
   - → Referential ambiguity
   - → Grammar Ambiguity

2) Lack of Context and "Common Sense"
   1) "Common Sense" Problem
   2) Context (memory) Problem

3) Pragmatics (Intended Meaning)

4) Speech - Specific Problems

See Image for more

Q-1 Explain applications of NLP in brief

Ans NLP is a branch of AI that enables computers to understand, interpret and generate human language

o Applications of NLP

1) Machine Translation
→ Used in systems like Google Translate to automatically translate text or speech from one language to another while preserving meaning and grammar

2) Chatbots and Virtual Assistants
→ NLP powers assistants like Siri, Alexa and ChatGPT enabling them to understand user queries, respond conversationally and perform tasks based on voice or text input

3) Sentiment Analysis
→ Bussiness use NLP to analyze customer reviews or social media posts to determine the sentiment and improve services or products

4) Text Summarization
→ Automatically generating summaries of longer texts. Eg: Used in web browsers to summarize

4) Text Summarization
→ Automatically generating summaries of longer texts. Eg: Used in web browsers to summarize

Page No.
Date

news articles

Information Extraction
→ Extracting specific information from unstructured text data
Example: Extracting names, dates, phone numbers, or key facts from documents

Q-2 Explain the Components of NLP

→ For natural ~~prace~~ language communication to take place, following two things are necessary

- Language Understanding
- Language Generation

→ Natural language understanding, it means to understand the context, and Natural language generation relates to sensible response to the context.

1) NLU (Natural Language Understanding)
   → NLU helps the machine to understand and analyze human language
   → NLU used in business applications to understand the customer's problem in both spoken and written language
   - NLU involves two tasks:-
     → Used to map the given input into useful representation
     → Used to analyze different aspects of the language

Q.5 State Phases of NLP. Explain each step for following statement "GTU is the winner of Robocon 2023"

Ans 1) Lexical Analysis
This initial phase focuses on breaking down the input text into smaller units called tokens (words, punctuation) etc. It also identifies and categorizes these tokens, often including tasks like stemming and lemmatization

tokens: GTU | is | the | winner | of | | 2023
winner: → "win" + "-ner"
win: B

(2) Syntatic Analysis: Also known as parsing, this phase analyzes the grammatical structure of sentences. It determines how words relate to each other, identifies phrases, and checks for grammatical correctness
Eg:
- Subject: GTU
- Verb: is
- Predicate: the winner of

(3) Semantic Analysis: This phase delves into the meaning of text, focusing on the literal meaning of words, phrases and sentences. It aims to understand the relationships between words and how they contribute to the overall meaning

Page No
Date

Eg: Understands that "GTU" is the organisation and "winner" indicates victory or achievement in "Robocon 2023"

4) Discourse Integration : This phase examines how sentences relate to each other within a larger context. It considers the impact of previous sentences on the understanding of the current sentence, involving resolving pronoun references

For single sentence
- No previous sentence → treated as independent info.
- "GTU" is recognized as a new entity introduced in discourse

5) Pragmatic Analysis
The final phase focuses on understanding the intended meaning and effect of the text, considering factors like context, speaker intent and real world knowledge. This phase interprets nuances like sarcasm, politeness or the overall tone

Eg:
Understands that "Robocon" is a robotics competition and statement expresses an achievement by GTU in that event

Here are some of the main factors that make it difficult for a computer to understand natural language:

- **Ambiguity:** This is the biggest problem. Language is often unclear and can be interpreted in multiple ways.
    - **Word Ambiguity (Lexical):** Many words have more than one meaning. For example, "print" can mean "print a file" or "a footprint." The computer must figure out the correct sense from the context.
    - **Referential Ambiguity:** It's hard to know what pronouns (like "he," "she," "it," "they") or general nouns refer to. In the sentence, "Bill had a balloon. John wanted it," the computer must figure out that "it" refers to the "balloon."
    - **Grammar Ambiguity (Syntactic):** A sentence's structure can be understood in more than one way. For example, "I saw the man on the hill with a telescope." The computer doesn't know if I have the telescope or if the man on the hill has it.

- **Lack of Context and "Common Sense":**
  - The meaning of a sentence often depends on the sentences that came before it (the discourse context).
  - **1. "Common Sense" Problem:**
  - **Sentence:** "I put the cake on the table. It collapsed."
  - **Problem:** A computer doesn't know *what* "it" is. It doesn't have the common sense to know that a cake is soft and a table is hard.
  - **2. Context (Memory) Problem:**
  - **Sentence 1:** "Mary has a red ball."
  - **Sentence 2:** "John saw it."
  - **Problem:** A computer reads "John saw it" and doesn't know what "it" is, because it forgot the first sentence (the context).

- **Pragmatics (Intended Meaning):**
  - A computer may understand the *literal* meaning of a sentence but not the *intended* meaning.
  - For example, if someone says, "I want to print Bill's file," they aren't just stating a fact. They are giving an implied **command** for the computer to perform an action.
- **Speech-Specific Problems:**
  - When processing spoken language, extra problems arise, such as handling different accents, slang words, or background noise.
  - **Speech-Specific Problems** I
  - This is why it's hard for a computer to *hear* you.

- **Speech-Specific Problems:**
  - When processing spoken language, extra problems arise, such as handling different accents, slang words, or background noise.
  - **Speech-Specific Problems**
  - This is why it's hard for a computer to *hear* you.

---

  - **Accents:** A person with a British accent says, "Pass me the **wa'er bo'le**," but the computer hears "**water battle**" and gets confused.
  - **Slang:** When you say, "That movie was *sick*," the computer thinks you mean "unwell," not "awesome."
  - **Background Noise:** If you say, "Play the song..." and a dog barks (...BARK...), the computer hears "Play the... bark..." and fails.

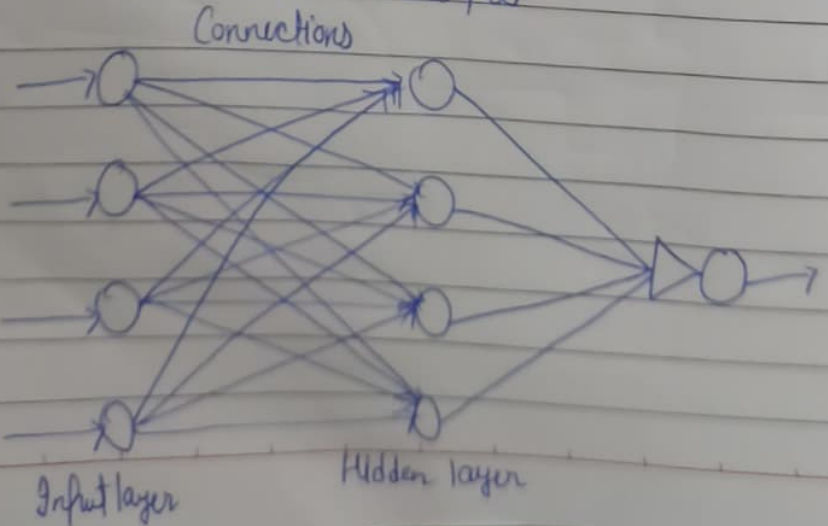
Ch-9

Q) Explain Artificial Neural Network (ANN)

Ans An ANN is a computing system that is inspired by the way the human brain works. It is not programmed with step-by-step instructions. Instead it "learns" to solve problems by processing many examples.

Main Features of ANN are

1) Many Simple Units: It is a built from a very large number of simple processing units, which acts like the "neurons" of in a brain.
2) Weighted Connections: These units are all connected to each other by a large number of links. Each link has a "weight", which is a number that represents the strength of connection
3) Knowledge is in the weights: The knowledge of entire network is stored in these connection weights
4) Parallel Processing: They are highly parallel meaning many units can process information at same time, much like the brain
5) Learning (learning from data): The network learns automatically adjusting these weights as it is shown more and more examples



Working

- Artificial Neuron (Perceptron):

1) Inputs: The neuron recieves one or more input values (like data from sensor)

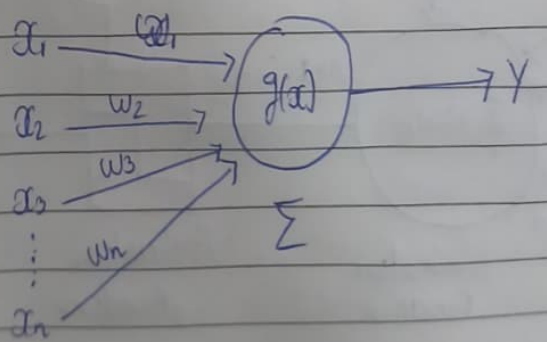
2) Weights: Each input is multiplied by a "weight". This weight represents the strength and importance of that connection. The network's entire knowledge is stored in these weights

3) Summing: The neuron adds up all the weighted inputs

4) Activation Function: The total sum is passed through an "activation function". This is a simple rule (like threshold) that decides what the neuron's final output should be (eg fire (1) or don't fire (0))

$$g(x) = \sum_{K=0}^{n} W_K X_K$$

$$\text{Output}(x) = \begin{cases} 1 & \text{if } g(x) > 0 \\ 0 & \text{if } g(x) < 0 \end{cases}$$



The Network Structure

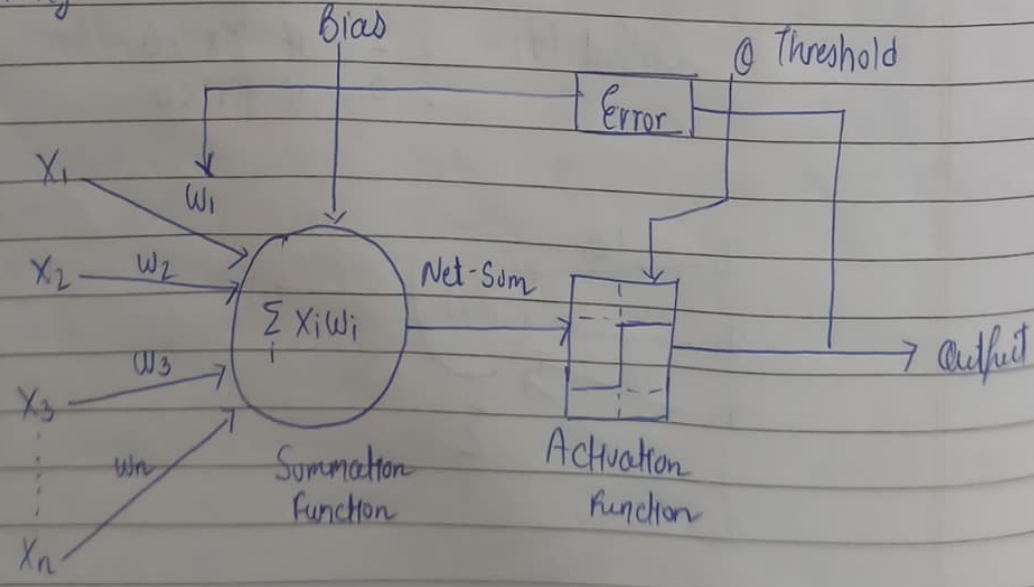
1) Input layer : First layer. It takes in the raw data
2) Hidden layer : These are layers in middle. They are the "brain" of the network, where complex patterns are recognized. A network can have one or many hidden layers
3) Output layer: This is last layer, which produces the final answer

Q-2 What is Perceptron

Ans It is one of the earliest and simplest models of an artificial neuron. It's the basic building block of a simple, single layer nural network

→ It is a linear classifier. This means it learns to make a decision or classify an input into one of two categories (eg "Yes" or "No", "1" or "0")

Working



1) Recieves Inputs
2) Calculates Weighted Sum
3) Adds the bias
4) Why bias is important
5) Applies activation function

See Image for more

Q-3 What is learning ? Explain various learning techniques . Also write difference between them

Ans In simple terms, learning is the process an AI system uses to adapt or improve itself
→ Instead of being perfectly programmed for every task , the system is given data or interacts with an environment
→ It uses this data to adjust the internal structure (like weights in a neural network) so it can get better at a task , like making predictions or finding patterns

Types
1) Superwised learning
2) Unsuperwised learning
3) Reinforcement learning

1) Superwised learning (No teacher or Answer Key)
→ The system is given "Labeled data" - examples where the correct answer is already provided (eg. Shape of square is labelled "Square")
→ The system goes through "Model training" where it tries to learn the rules that connect examples to their answer.

- **What it is:** A Perceptron is one of the earliest and simplest models of an artificial neuron. It's the basic building block of a simple, single-layer neural network.
- **What it does:** It is a **linear classifier**. This means it learns to make a decision or classify an input into one of two categories (e.g., "Yes" or "No", "1" or "0").

**How it Works (Figure 13.5)**

1. **Receives Inputs:** It takes several inputs (features), shown as x1, x2, ... xp.
2. **Calculates Weighted Sum:** Each input x is multiplied by its "weight" w (like wk1, wk2). The weights show how important each input is.
3. **Adds the Bias:**
    - There is a special input, x0, which is always a fixed value of **+1**.
    - This +1 input also has its own weight, wk0 (which is labeled bk (bias)).
    - This **bias weight** is added to the total sum just like all the other weighted inputs.
4. **Why is the bias important?**
    - The bias gives the perceptron flexibility.
    - If you think of the perceptron's decision as a line on a graph (like in Figure 13.3), the bias allows that line to **shift left or right**, so it doesn't have to pass

inputs.

4. **Why is the bias important?**
    - The bias gives the perceptron flexibility.
    - If you think of the perceptron's decision as a line on a graph (like in Figure 13.3), the bias allows that line to **shift left or right**, so it doesn't have to pass through the center (0,0).
    - This lets it solve many more problems. The "threshold" is just another way of thinking about this bias.
5. **Applies Activation Function:** The total sum (weighted inputs + bias) is passed to an activation function. This function makes the final decision:
    - If the total sum is greater than 0, the perceptron "fires" and outputs **1**.
    - If the total sum is 0 or less, it outputs **0**.

1) Recieves Inputs
2) Calculates Weighted Sum
3) Adds the bias
4) Why bias is important
5) Applies activation function

See Image for more

Q-3 What is learning ? Explain various learning techniques . Also write difference between them

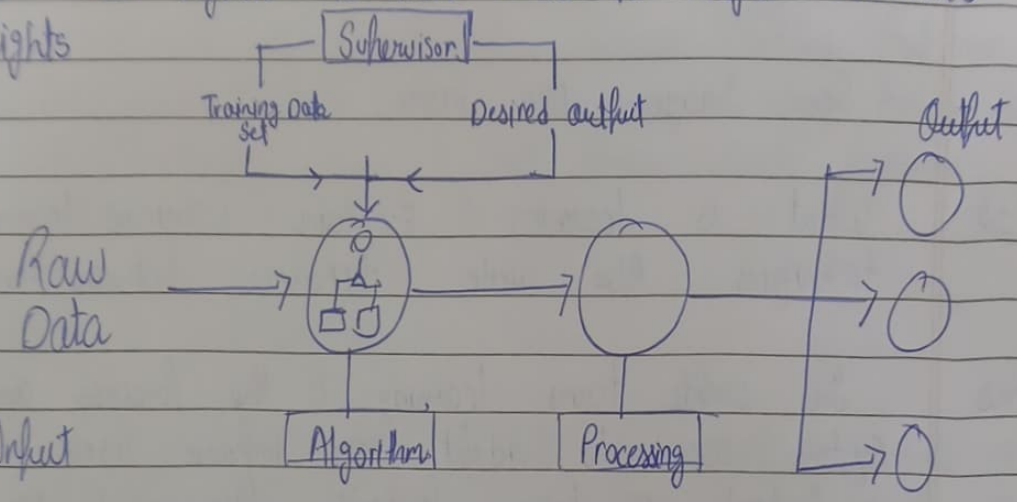
Ans In simple terms, learning is the process an AI system uses to adapt or improve itself
→ Instead of being perfectly programmed for every task , the system is given data or interacts with an environment
→ It uses this data to adjust the internal structure (like weights in a neural network) so it can get better at a task , like making predictions or finding patterns

Types
1) Superwised learning
2) Unsuperwised learning
3) Rein Forcement learning

1) Superwised learning (No teacher or Answer Key)
→ The system is given "Labeled data" - examples where the correct answer is already provided (eg. Shape of square is labelled "Square")
→ The system goes through "Model training" where it tries to learn the rules that connect examples to their answer.

→ The goal is to make accurate "Predictions" on new, unseen "Test Data" that it hasn't been trained on

→ If the prediction is wrong, an "Error signal is created" by comparing that wrong output to correct answer, and this signal is then used to adjust network's weights




2) Unsupervised learning

This is learning without a teacher or an "answer key"

→ The system is given "Unlabelled data" - just a mix of items with no labels or correct answers

→ The algorithm's job is not to be "correct", but to find hidden patterns or structures in the data all by itself

→ As both images show, the most common result is clustering, where the algorithm sorts the jumbled data into organized groups (eg. all the dogs in one group, all the cats in another)

→ Same Diagram
Except the
Supervisor stuff

3) Reinforcement Learning

Idea: This is like learning by trial and error to get a reward

Working

→ the AI, called an "agent" is placed in an "Environment" (like a game)

→ the agent performs an "Action"

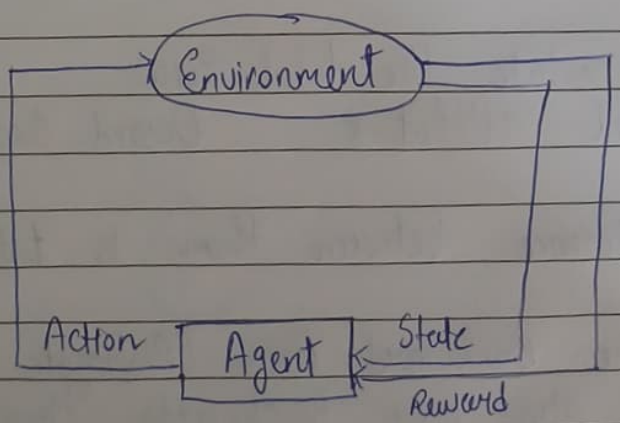
→ the agent is not told which actions are right or wrong. Instead, it must discover which actions are good by trying them

→ when it performs good action, it recieves a "reward" (like points)

→ when bad, penalty (like losing points)

→ the AI's goal is to learn a strategy (called a "policy") that maximizes its total reward over time

→ the Environment responds by giving agent a "Reward" (a positive score for a good action) or penalty (bad action) and a new State (the new situation)




Difference between learning techniques

See Image

## ✅ Differences Between Learning Types

| Aspect | Supervised Learning | Unsupervised Learning | Reinforcement Learning |
|---|---|---|---|
| **Type of Data** | Learns from **labeled data** (input + correct output) | Learns from **unlabeled data** (input only) | Learns by **rewards or penalties** from environment |
| **Goal** | To **predict the correct output** for new inputs | To **find hidden patterns** or form groups (clusters) | To learn a **strategy (policy)** that gets maximum rewards |
| **Teacher Availability** | Has a **teacher** (error signal guides learning) | **No teacher** | **No teacher**, but guided by rewards/penalties |
| **Feedback Type** | Gets an **error signal** (how wrong the output was) | **No feedback** | Gets a **reward signal** (score) |
| **Output Type** | Classification or regression results | Clusters, associations, patterns | Optimal actions or decisions |
| **Example Tasks** | Spam detection, price prediction | Customer segmentation, anomaly detection | Game playing, robot navigation |
| **Learning Style** | Learning from **examples** | Learning from **data structure** | Learning from **trial and error** |
| **Data Requirement** | Needs **large labeled datasets** | Works with **unlabeled datasets** | Needs **interaction** with environment |

Q-4 Explain Back Prophagation Algorithm in Neural Network

Ans → Most Important
→ It is a superwised learning algorithm/method, meaning it learns from data that includes the correct answers

GOAL:-
The algorithm's goal is to figure out how much each and every "weight" (connection strength) in the network contributed to the final error, and then adjust all the weights to make the network's guess more accurate next time

Steps:

Step 1: The Forward Pass (make a guess)

→ Image of 2
→ Final guess : 8 (actual output

Step 2: Calculate the total Error.
Actual output: 8 Desired Output = 2

Difference between them is total error

Step 3: The Backward Pass (Prophogate the Error)
→ The algo now backwards through the network starting from end

The '1' neuron was very wrong

Step 4: Update the weights

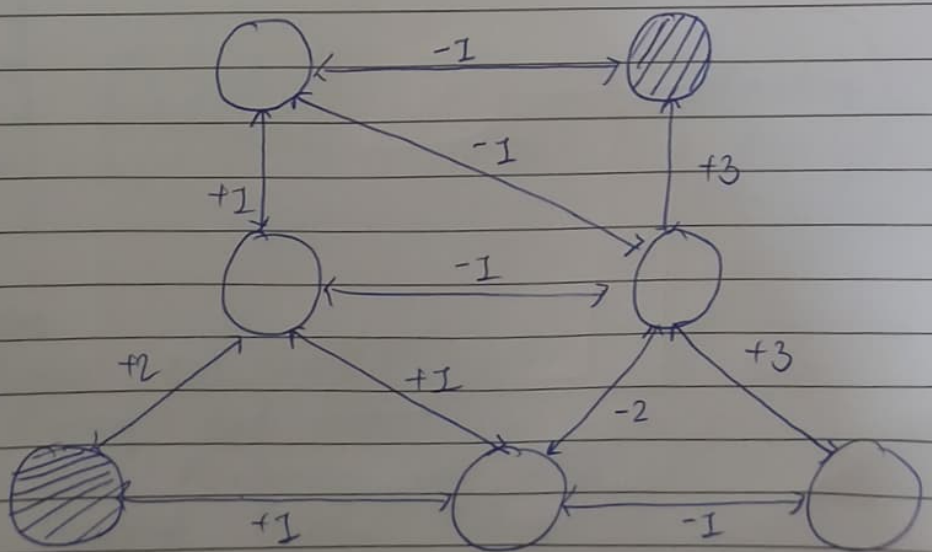
Step 5: Repeat

See Image for more

Q-5 Explain Hopfield Network in brief

Ans A Hopfield Network is a special type of "connectionist model" (or neural network). Its main purpose is to work like a content-addressable memory
→ Content-addressable means it can retrieve a full, complete memory (a "pattern") even if you give it a partial or "noisy" (corrupted) version of that pattern.

o Working
Structure: It has only one layer of simple processing units (neurons). These units are all connected to each other in a feedback loop.




Black ● → active & White ○ → inactive

The process is a loop that repeats for every piece of training data (like every image or data row).

**Step 1: The Forward Pass (Make a Guess)**

- You feed an input (like an image of a '2') into the **input layer** of the network.
- The data flows forward through the **hidden layers** to the **output layer**.
- The network makes a final guess (e.g., it might output '8'). This is the **"actual output"**.

**Step 2: Calculate the Total Error**

- You compare the network's guess (the "actual output", '8') with the **"desired output"** (the correct answer, which was '2').
- The difference between them is the **total error**.

**Step 3: The Backward Pass (Propagate the Error)**

- This is the most important part. The algorithm now works **backward** through the network, starting from the end.
- **At the Output Layer:** It calculates how much the output neurons contributed to the error. (e.g., the '8' neuron was "very wrong").
- **At the Hidden Layers:** It takes this error and "propagates" (sends) it backward to the last hidden layer. Each neuron in that hidden layer gets a "share" of the blame for the error.

error. (e.g., the '8' neuron was "very wrong").

- **At the Hidden Layers:** It takes this error and "propagates" (sends) it backward to the last hidden layer. Each neuron in that hidden layer gets a "share" of the blame for the error.
- This process continues, passing the blame backward from layer to layer, until it reaches the first hidden layer. This step figures out exactly how much every single weight in the network was responsible for the final mistake.

**Step 4: Update the Weights**

- Now that the algorithm knows how much each weight contributed to the error, it adjusts all of them slightly.
- Weights that were *very responsible* for the error are changed a lot.
- Weights that were *mostly correct* are changed very little.

- Weights that were *mostly correct* are changed very little.

- The adjustment is made to *reduce* the error.

**Step 5: Repeat**

- The entire process (Steps 1-4) is repeated with the next piece of training data (e.g., an image of a '5', then a '3', etc.).
- The network does this thousands of times, and with each pass, its weights get a little better, and the overall error gets lower.

Step 4: Update the weights

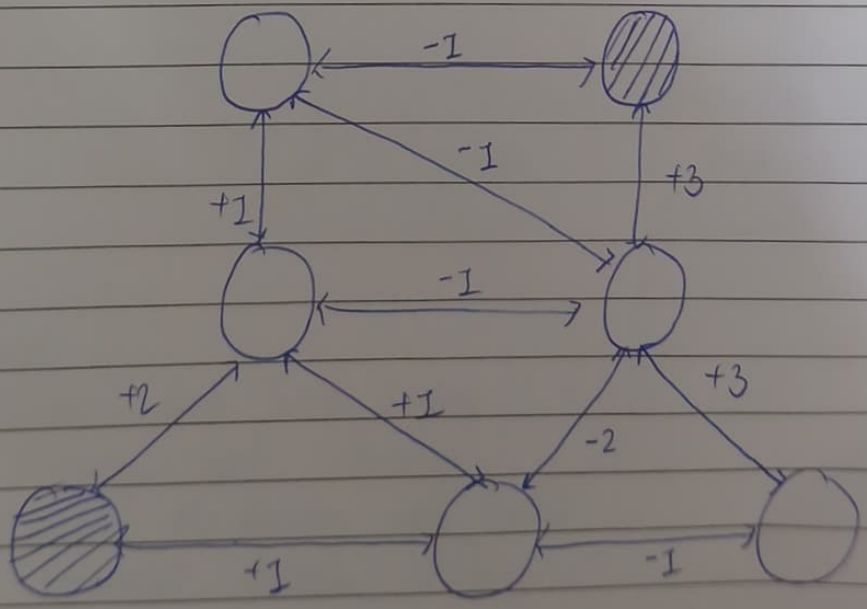
Step 5: Repeat

See Image for more

Q.5 Explain Hopfield Network in brief

Ans A Hopfield Network is a special type of "connectionist model" (or neural network). Its main purpose is to work like a content-addressable memory
→ Content-addressable means it can retrieve a full, complete memory (a "pattern") even if you give it a partial or "noisy" (corrupted) version of that pattern.

o Working
Structure: It has only one layer of simple processing units (neurons). These units are all connected to each other in a feedback loop.




Black ● → active & White ○ → inactive.

States
Active (+1)
Inactive (0)

Weights: the connection between units have weights (numbers)
A positive weight means two unit tend to activate each other, and if negative then deactivate

Setting Down

1) A random unit (neuron) is chosen
2) It looks at all its active neighbours
3) It sums up the weights from these active neighbours
4) If the total sum is positive, the unit turns ON.
5) If the total sum is negative, the unit turns OFF
6) The network repeats this process, picking another random unit and updating its state

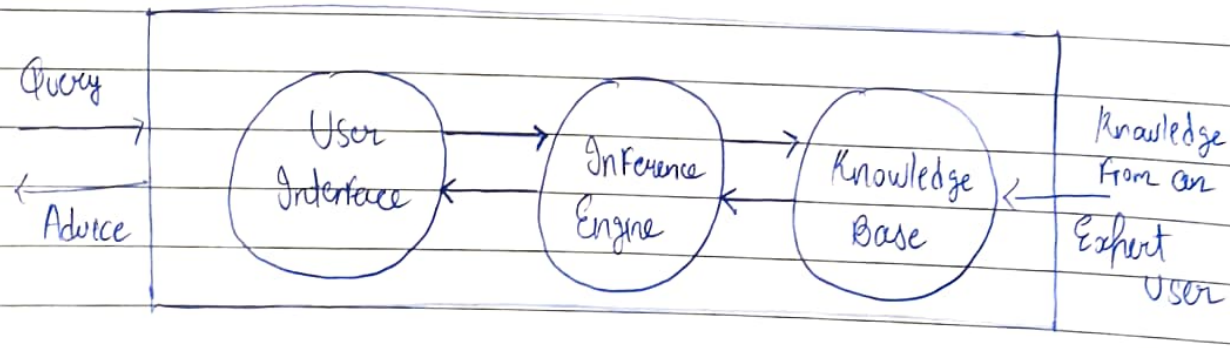
Ch 10

Date
Page

2. What do you mean by Expert Systems? List out its four applications

Ans An Expert System is a computer program that is designed to solve complex problems and provide decision making ability, just like a human expert in a specific field.

It works by using two main components
1) A Knowledge Base
→ A database filled with the facts and "if-then" rules that a human expert would use
2) An Inference Engine
→ the "brain" of the system that uses the rules and facts to reason about a user's problem and come to a conclusion

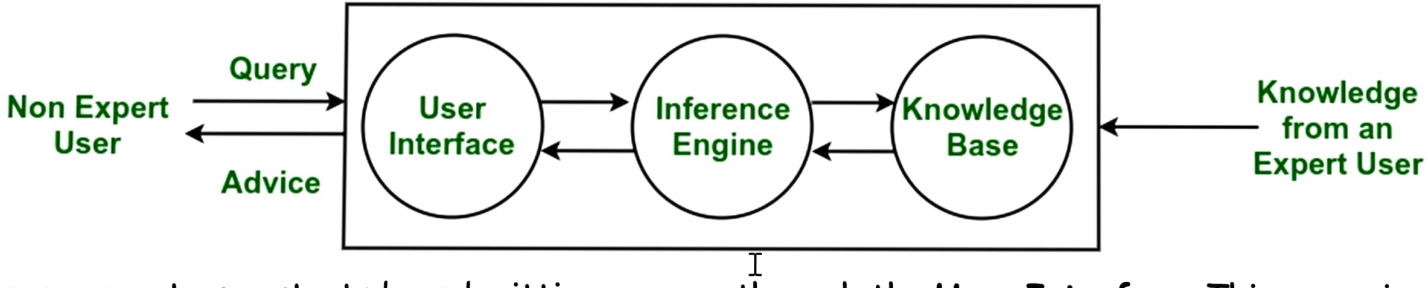



Applications
1) Diagnosis and Trouble Shooting
2) Planning and Scheduling
3) Process Monitoring and Control
4) Financial Decision Making

Q. Draw and explain Architecture of Expert System

**What is an Expert System?**

- An **Expert System** is a computer program that is designed to solve complex problems and provide decision-making ability, just like a human expert in a specific field.
- It's a type of AI that models the behavior and knowledge of an expert in a narrow domain, such as medicine, engineering, or finance.
- It works by using two main components:
    1. A **Knowledge Base:** A database filled with the facts and "if-then" rules that a human expert would use.
    2. An **Inference Engine:** The "brain" of the system that uses the rules and facts to reason about a user's problem and come to a conclusion.



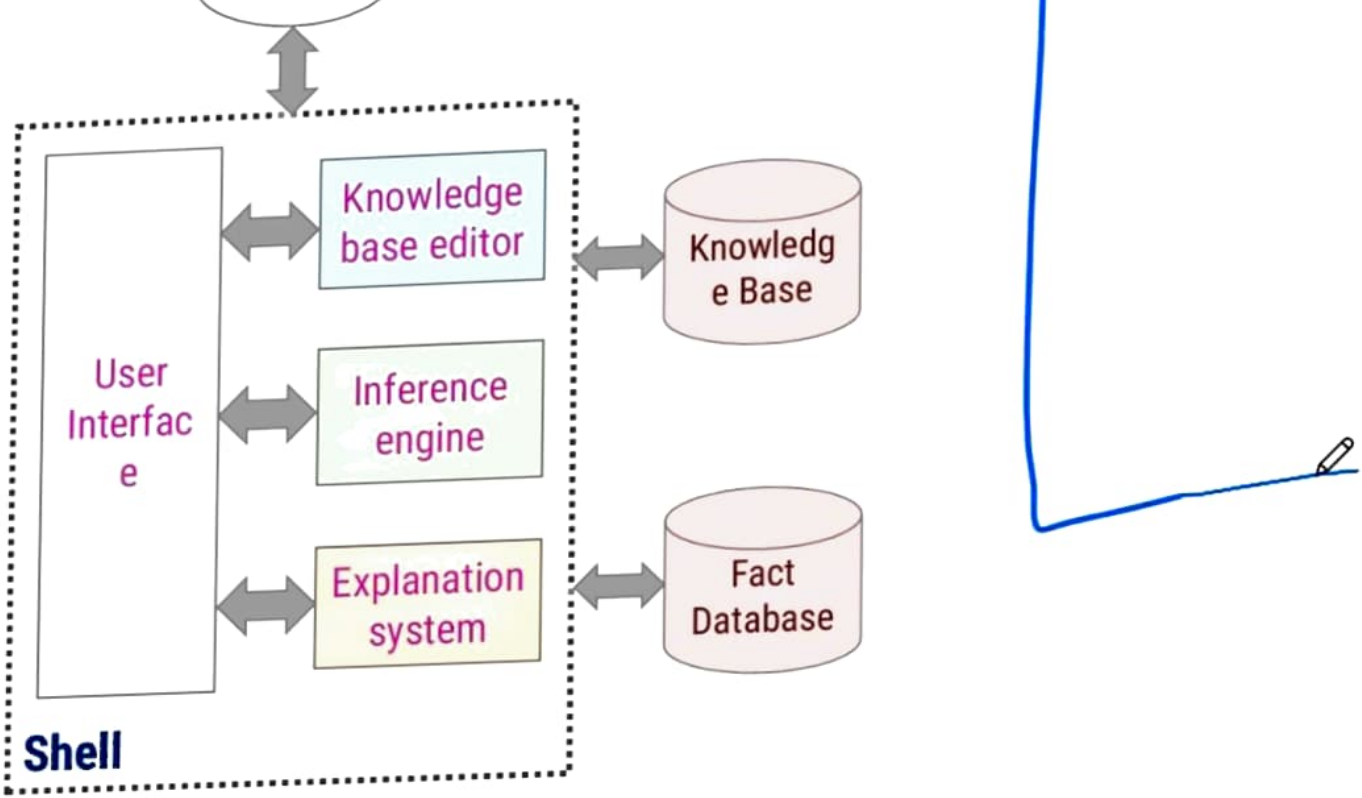

A non-expert user starts by submitting a **query** through the **User Interface**. This query is sent to the **Inference Engine**. The Inference Engine then asks the **Knowledge Base** for

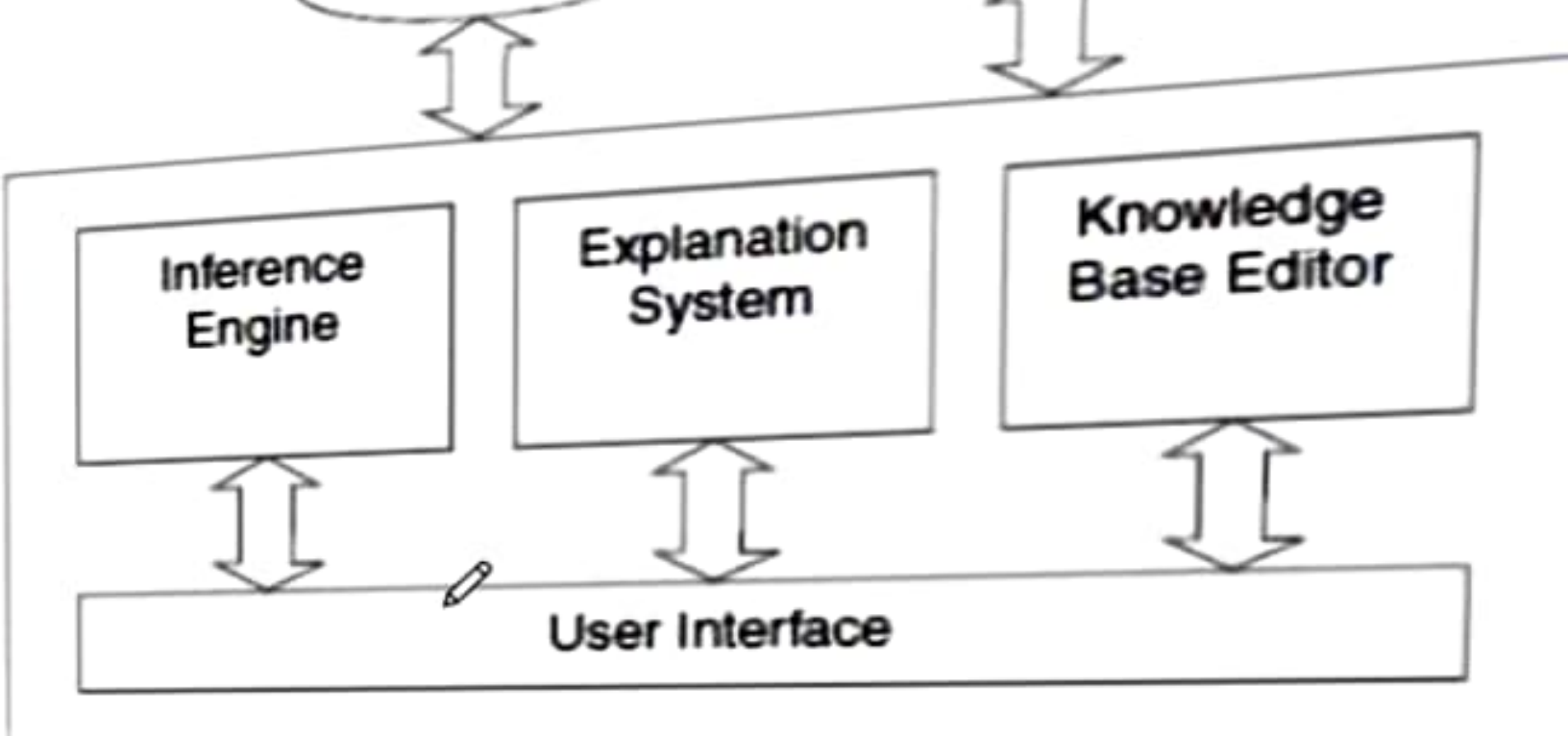
A non-expert user starts by submitting a **query** through the **User Interface**. This query is sent to the **Inference Engine**. The Inference Engine then asks the **Knowledge Base** for relevant facts and rules. The Knowledge Base **sends this information back** to the Inference Engine. Using these rules, the Inference Engine figures out a solution and sends this final **Advice** back to the User Interface for the user to see.

**Applications of Expert Systems**

Here are four common applications:

- **Diagnosis and Troubleshooting:** Used to identify problems in complex systems or to diagnose diseases. (e.g., MYCIN, a system to help doctors diagnose blood infections).
- **Planning and Scheduling:** Used to create complex plans and schedules. (e.g., helping airlines schedule their flights, crews, and gates).
- **Process Monitoring and Control:** Used to monitor real-time data from a factory or plant (like an oil refinery) to look for problems and control the system.
- **Financial Decision Making:** Help banks decide whether to give a loan and help insurance companies to check a customer's risk.


Knowledge base editor
Knowledg e Base
User Interfac e
Inference engine
Explanation system
Fact Database
Shell


Inference Engine
Explanation System
Knowledge Base Editor
User Interface

- **1. Knowledge Base:**
  - This is the "brain" or "library" of the system.
  - It contains the permanent, specialized knowledge of a human expert, stored in the form of **"IF-THEN" rules**.
- **2. Inference Engine:**
  - This is the "reasoning" part of the system. It's the component that "thinks."
  - It takes the rules from the Knowledge Base and applies them to the current facts of the problem (which are in the Fact Database).
  - It uses reasoning methods (like forward or backward chaining) to deduce new information and arrive at a final conclusion.
- **3. Fact Database (or Working Memory):**
  - This is the system's temporary memory for the *current problem*.
  - It holds the specific facts given by the user (e.g., "The patient has a fever").
  - The Inference Engine constantly updates this database as it discovers new facts.

- It holds the specific facts given by the user (e.g., "The patient has a fever").
- The Inference Engine constantly updates this database as it discovers new facts.

- **4. User Interface:**
  - This is the part of the program that the **end-user** (the person getting advice) interacts with.
  - It allows the user to enter their problem, ask questions, and receive the final answer.
- **5. Explanation System:**
  - This is a very important component that builds trust.
  - It allows the user to ask "Why?" or "How?"
  - The system can then explain *how* it reached its conclusion by showing the chain of rules it used.
- **6. Knowledge-base Editor:**
  - This is the tool for the **programmer or human expert** (the "knowledge engineer").
  - It is used to add new rules, delete old rules, and modify existing rules in the Knowledge Base.

**Expert System Shells**

- **What it is:** An **Expert System Shell** is a software package that provides the complete architecture of an expert system, but with an **empty Knowledge Base**.
- **The Concept:** The "Shell" contains the Inference Engine, the User Interface, the Explanation System, and the Knowledge-base Editor. It is a general-purpose toolkit.
- **How it's used:** A developer can buy an "empty" shell and then use the Knowledge-base Editor to add their own specific rules. This allows them to create a new, custom expert system without having to build the complicated parts (like the inference engine) from scratch.

**Example:**

**Example:**

- A famous early expert system was **MYCIN**, which was designed to diagnose blood infections. It had a very good inference engine and explanation system.
- Researchers realized they could **re-use** the main parts of MYCIN *without* the medical rules.
- This empty version was called **EMYCIN** (for "Empty MYCIN").
- This **EMYCIN shell** could then be used by other people. For example, a geologist could add rules about rocks to EMYCIN to create a new expert system for identifying minerals, all while using the same inference engine and user interface that MYCIN used.

See Image for more

Q.3 Describe the Expert System Development Procedure
OR
Explain Knowledge acquisition in expert system

Ans The development of an expert system involves several key people
1) the Domain Expert: the human expert who has the specialized knowledge (eg engineer, doctor, geologist)
2) the Knowledge Engineer: The AI specialist who designs the system, interviews the expert and writes the rules
3) the End-User: The person who will eventually use the system to get advice

The development procedure generally follows these steps:-
1) Knowledge Acquisition
→ First and often most difficult step
→ Knowledge Engineer works closely with Domain Expert to get the expert's knowledge
→ Done by observing expert work, asking them questions about how they solve problems and learning what rules they use to make decisions.
→ Knowledge can be gathered from textbooks, manuals and databases

2) Knowledge Representation (Encoding)
→ Knowledge engineer must take the expert's knowledge and translate into a formal language that computer can understand
→ Writing expert logic as set of "IF-THEN" rules

3 Building the System (Implementation)
→ The encoded rules are located into Knowledge Base of an Expert System shell
→ The shell provides all the other necessary components
→ This saves the engineer from having to build the whole system from stratch

4) Testing and Refinement
→ The expert system is tested with sample problems
→ The Domain experts check the system's answer to see if they are correct
→ The Knowledge Engineer then "refines" the system by fixing bad rules, adding missing rules and updating the knowledge base
→ This is not a one time step: the knowledge is tested and refined continually throughout the life of system to keep it accurate and up-to date.

Ch-11

Q-1 Explain about the basic operators in genetic algorithms

Ans There are three basic operators

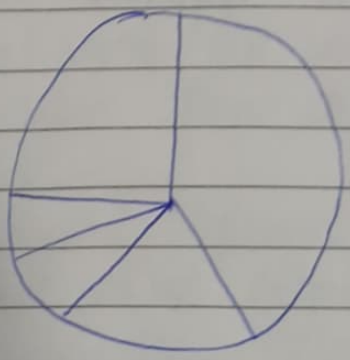
1) Selection
2) Crossover (Recombination)
3) Mutation

1) Selection
→ "Survival of the fittest" operator. It decides which solutions (chromosomes) from the current population get to be "parents" for the next generation
→ It uses fitness score of each solution to make this choice. Solutions with a better fitness score are more likely to be selected
→ Significance: This operator is what it guides the algorithm towards a good answer. It gives preference to better solutions, allowing them to pass their "genes" to next generation, while bad solutions (with low scores) die out.

Example
1) Roulette-Wheel Selection
1) Assign Slices: Each of solution in the population is given a slice of the wheel.
2) Size Slices by Fitness: The size of each solution's slice is proportional to its fitness score. A solution with a very high score gets a big slice, and a solution with a low score gets a tiny slice.
3) Spin the Wheel: The algorithm "spins" the wheel. The solution where the ball lands is "selected" to be a parent.

4) Repeat: The process is repeated until you have enough parents to create the new generation

Roulette Wheel Selection



| Chromosome | Fitness Value |
|---|---|
| A | 8.2 |
| B | 3.2 |
| C | 1.4 |
| D | 1.2 |
| E | 4.2 |
| F | 0.3 |

2) Tournament Selection

See Image

2. Crossover (Recombination)
→ "Reproduction operator"
→ It takes two "parent" solutions (chosen by Selection) and combines them to make new "offspring" solutions
→ It swaps parts of the "genes" (the solution string) from the two parents
→ Crossover is how the algorithm explores new solutions. It creates off-spring that are a mix of the best traits from their parents based on the idea that combining two good solutions might create a better choice

## 2. Tournament Selection

This is a way to choose the "parents" (the best solutions) for the next generation. It works like a mini-contest.

1. **Pick Randomly:** The algorithm picks 2 solutions randomly from the population.
2. **Compare Fitness:** It compares their fitness scores.
3. **Winner is Chosen:** The solution with the **best** score wins the "tournament" and is chosen to be a parent.
4. **Repeat:** The process is repeated until you have enough parents.

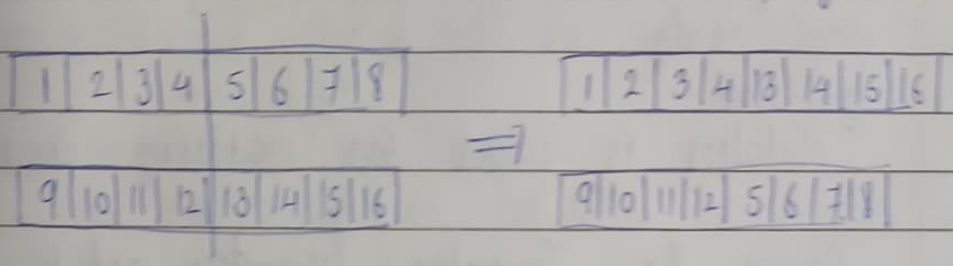
**Explanation of the Example**

- The diagram shows one single "tournament" happening.
- **1. Pick Randomly:** From the whole list (A-G), the algorithm randomly picked two solutions:
  - **B** (which has a fitness score of 70)
  - **F** (which has a fitness score of 40)
- **2. Compare:** It compares the two scores: 70 is better than 40.
- **3. Winner:** **B** wins the tournament and is selected to be a parent.

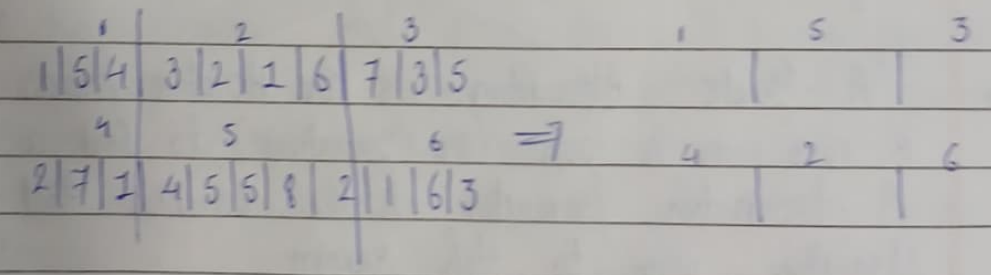
▸ Tournament Selection (2 Players)

Q. Explain types of Cross Over Operators in Genetic Algorithm

Ans 1) Single Point Crossover

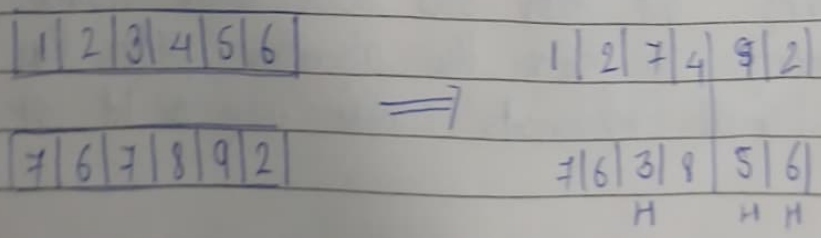



2) Two Point Crossover




3) Uniform Crossover

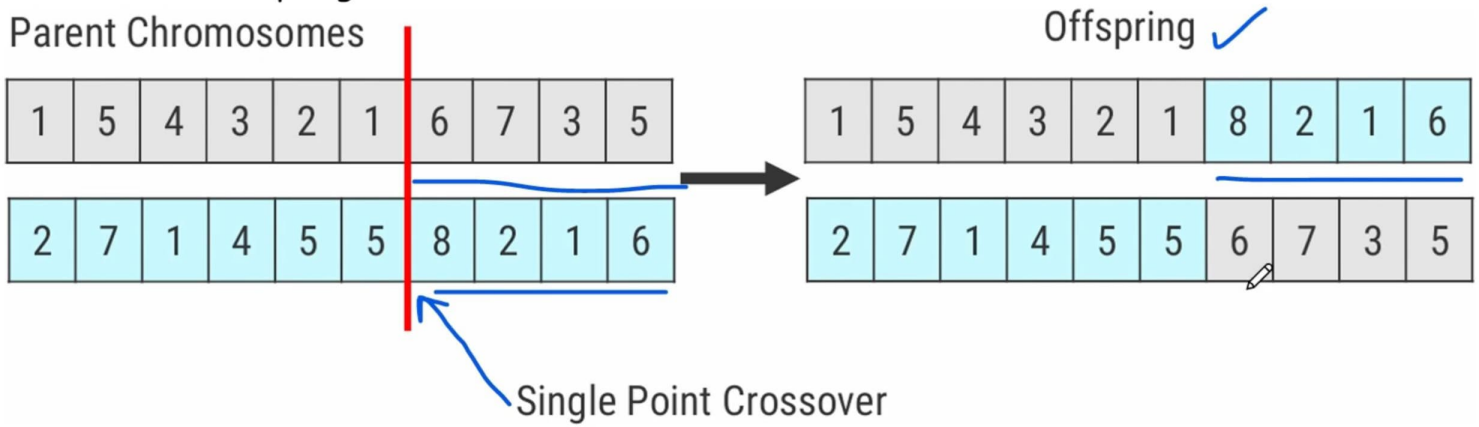
→ Each gene (each position in string) is looked at one by one
→ For each gene, a "coin is flipped" (or probability is used)
→ Based on the coin flip, the offspring gets the gene from either parent 1 or 2
If H=1 bits swapped



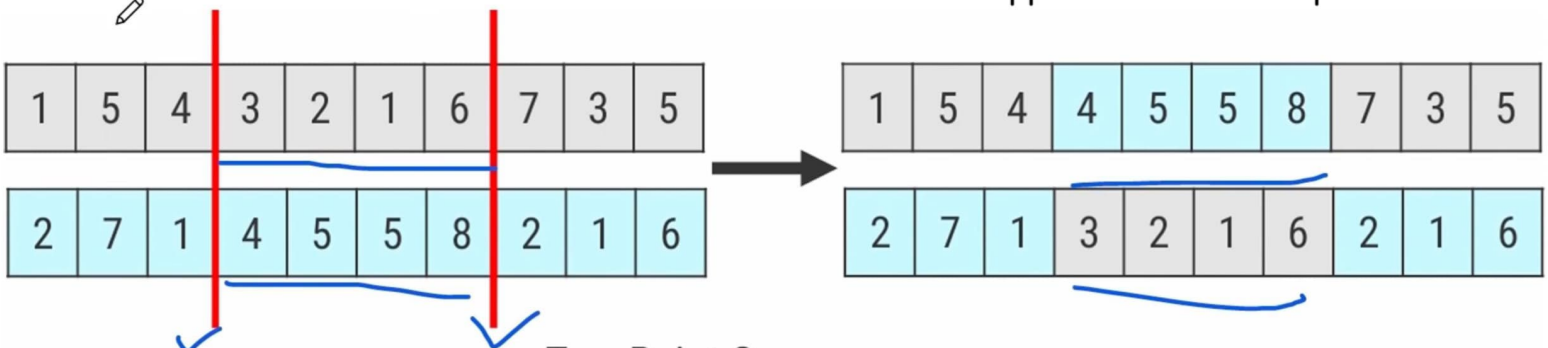
**Various Types of Crossover Operators**

There are several ways to swap the genes:

- **1. Single-Point Crossover:**
  - A single "crossover point" is chosen randomly on the chromosome string.
  - Both parent strings are "cut" at this same point.
  - The second halves of the two strings are then swapped to create two new offspring.



- **2. Two-Point Crossover:**
  - Two random points are chosen on the string.
  - The strings are "cut" at both points.
  - The **middle section** between the two cuts is swapped between the parents.




Two Point Crossover

- **3. Uniform Crossover:**
  - Each gene (each position in the string) is looked at one by one.
  - For each gene, a "coin is flipped" (or a probability is used).
  - Based on the coin flip, the offspring gets the gene from either Parent 1 or Parent 2.

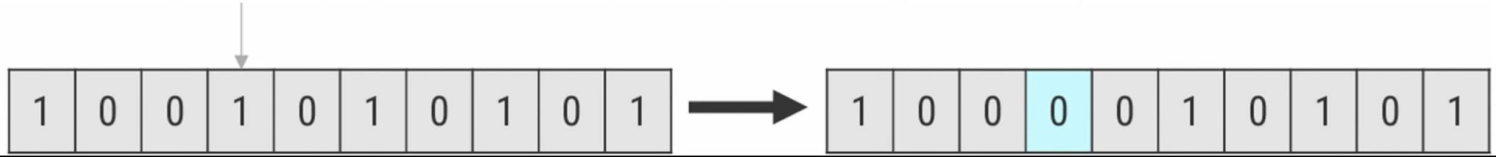
3) Mutation
→ This is a small, random change that is made to a new offspring solution
→ It randomly changes one of the "genes". For example, it might flip a 1 to a 0 in binary string, or randomly swap two cities in a route. This happens very rarely
Significance:
Mutation is crucial for maintaining diversity

Q-2 Discuss the Termination parameters used in genetic algorithms detail

Ans A Genetic Algorithm is an iterative process, meaning it runs in a loop (or "generations") to find a solution
A termination parameter is the condition that tells the algorithm when to stop running

Importance of Termination Parameters
We need a stopping condition because
→ An algorithm that runs forever isn't useful
→ We want to stop when the solution is "good enough" or close to the optimal answer
→ Genetic Algorithms often progress very fast at beginning but then slow down later (called saturation) Continuing to run the algorithm after this point gives very small improvements while using a lot of computer time

3. **Mutation**
   - **What it is:** This is a small, **random change** that is made to a new offspring solution.
   - **How it works:** It randomly changes one of the "genes." For example, it might flip a 1 to a 0 in a binary string, or randomly swap two cities in a route. This happens very rarely.
   - **Significance (Why it's important):** Mutation is crucial for **maintaining diversity**. Sometimes, Selection and Crossover can cause the algorithm to get "stuck" on a good solution that isn't the *best* solution (a local optimum). Mutation introduces brand new genetic material, allowing the algorithm to "jump" to a new part of the search space and potentially find a better solution that it would have missed.



Q-2 Discuss the Termination parameters used in genetic algorithms detail

Ans A Genetic Algorithm is an iterative process, meaning it runs in a loop (or "generations") to find a solution. A termination parameter is the condition that tells the algorithm when to stop running.

Importance of Termination Parameters

We need a stopping condition because

→ An algorithm that runs forever isn't useful

→ We want to stop when the solution is "good enough" or close to the optimal answer

→ Genetic Algorithms often progress very fast at beginning but then slow down later (called saturation) Continuing to run the algorithm after this point gives very small improvements while using a lot of computer time

Common Termination Parameters (Stopping Methods)

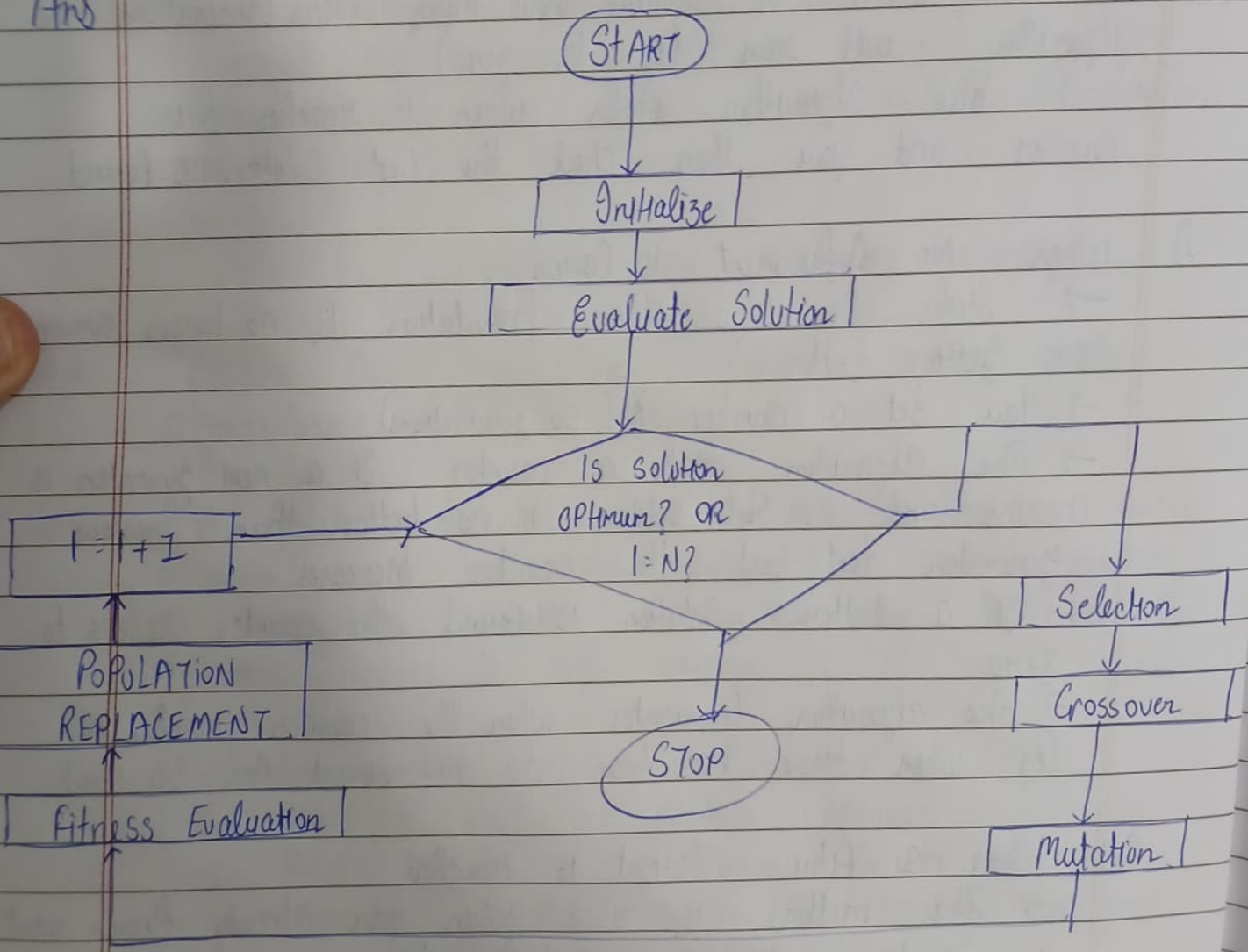
1) After a Pre-Specified Number of Generations
→ Simplest method
→ You decide in advance how many loops (generations) the algorithm will run (eg 100 gen.)
→ The algorithm stops when it reaches this number, and you then check the best solution is found

2) When No Improvement is Found
→ Stops the algo when population is no longer 'evolving' or getting better
→ You set a number N (50 generations)
→ The algorithm keeps a counter. If a new generation is created and its best solution is no better than previous generation test best, the counter increases
→ If a better solution is found, the counter resets to zero.
→ The algorithm terminates when the counter reaches N (eg. when there has been no improvement for 50 gen)

3) When a Fitness Target is Reached
→ This method is used when you already known what a "good enough" answer looks like.
→ You set a target value for the objective function (the fitness score)
→ The algorithm runs until it produces a solution whose fitness score meets or exceeds this target value

Q-3 Describe Working Principle of Genetic Algorithm
OR Describe the Phases of genetic Algorithm

Ans



**Explanation of the Flowchart Steps**

Here is what each part of the flowchart means:

- **1. START**
  - This is the beginning of the algorithm.
  - Here, we set the generation counter I = 0 and decide on the maximum generations, N (e.g., N=100).
- **2. Initialize Population**
  - The algorithm creates a large group (a "population") of many **random** solutions.
  - These are the "first generation" of solutions. They are usually not very good.
- **3. Evaluate solution (Fitness Evaluation)**
  - The algorithm checks every single solution in the current population.

- **3. Evaluate solution (Fitness Evaluation)**
  - The algorithm checks every single solution in the current population.

---

  - It uses a **"Fitness Function"** (a judge) to give each solution a **score**.
  - This score tells us how good or "fit" the solution is.
- **4. Is solution optimum? Or I = N? (The "Stop" Check)**
  - This is a "YES/NO" question to see if the algorithm should stop.
  - It asks two things:

- **4. Is solution optimum? Or I = N? (The "Stop" Check)**
  - This is a "YES/NO" question to see if the algorithm should stop.
  - It asks two things:
    1. "Have we found a perfect (optimum) solution?"
    2. "Have we run out of time (is the generation counter I equal to N?)"
  - If the answer is **YES** to *either* question, the algorithm follows the "Y" arrow and goes to STOP.
  - If the answer is **NO**, the algorithm follows the "N" arrow and continues, to try and "evolve" a better solution.
- **5. Selection**
  - This is the "survival of the fittest" step.
  - The algorithm looks at all the solutions and their scores (from Step 3).
  - It **selects** the best (fittest) solutions to be "parents" for the next generation.

- **5. Selection**
  - This is the "survival of the fittest" step.
  - The algorithm looks at all the solutions and their scores (from Step 3).
  - It **selects** the best (fittest) solutions to be "parents" for the next generation.
  - Bad solutions (with low scores) are discarded and "die off."
- **6. Crossover (Recombination)**
  - This is the "reproduction" step.
  - The "parent" solutions (from Step 5) are paired up.
  - Their "genes" (parts of their solutions) are swapped and combined.
  - This creates new "offspring" (child) solutions that are a mix of their parents' best parts.

- **7. Mutation**
  - This step makes a small, **random change** to a few of the new offspring.
  - *Example:* It might flip one 0 to a 1 in a solution string.
  - This is important to add new, random ideas and prevent the algorithm from getting stuck.
- **8. Replacement**
  - The new population of "offspring" (created by Crossover and Mutation) now **replaces** the old, parent population.
- **9. I = I + 1**
  - The algorithm updates its generation counter.
  - It adds 1 to I to show that a new generation is complete.
  - After this, the flowchart loops back to the **Evaluate solution** step (Step 3) to start the whole process over again with the new generation.
- **10. STOP**
  - The algorithm ends.
  - The final, best solution found in the last generation is given as the answer to the problem.

# Ch-12 Introduction to Prolog

Q-1 What are the Applications, Features and Limitations of Prolog?

Ans → Prolog stands for "Programming in Logic"
→ Prolog is a declarative programming language
Tell computer two things
1) Facts : A database of things that are true (eg "Apples are Food")
2) Rules: Logical relationships between Facts (eg "John likes all things that are Food")

Working
After you give it facts and rules, you ask a question (called a "goal") the Prolog System that uses logical deduction to search through the facts and rules to find the answer

- Features of Prolog
  1) Declarative
  2) Based on logic
  3) Facts and Rules
  4) Backward Chaining
  5) Unification
  6) Backtracing
  7) Recursion

- Applications of Prolog
  1) AI
  2) NLP
  3) Expert System
  4) Automated Reasoning

Limitations of Prolog

1) Not For Numbers
2) Can be inefficient
3) "Black box" Reasoning
4) Limited Scope

Q-2 Explain with Example how recursive predicate is defined in Prolog ?

Ans How to define Recursive Predicate in Prolog.

Must define two parts

1) The Base Case (The "Stopping Condition")
   → This is simple fact that tells program when to stop
   → It must handle the simplest possible input (like 0, or an empty list)
   → If you forget this, you will create an infinite loop.

2) The Recursive Rule (The "Do-it-again" step)
   → This is the that calls itself
   → It must break the problem down into a smaller, simpler version that moves closer to Base Case

○ How Prolog does math

X = 5+1 X

X is 5+1 ✓

Q. What are the Applications, Features and Limitations of Prolog? **7M W23**

---

**What is Prolog?**

- **Name:** "Prolog" stands for **"Programming in Logic"**.
- **What it is:** Prolog is a **declarative programming language**.
- **Declarative vs. Procedural:**
    - Most languages (like C, Python, Java) are **procedural**. You must write step-by-step instructions telling the computer *how* to solve a problem.
    - Prolog is **declarative**. You *don't* write step-by-step instructions. Instead, you just tell the computer two things:
        1. **Facts:** A database of things that are true (e.g., "Apples are food.").
        2. **Rules:** Logical relationships between facts (e.g., "John likes all things that are food.").
- **How it works:** After you give it facts and rules, you **ask a question** (called a "goal"). The Prolog system then uses **logical deduction** to search through the facts and rules to find the answer.

**Applications of Prolog**

Because Prolog is so good at logic, rules, and language, it is mainly used in specific AI-related fields:

- **Artificial Intelligence (AI):** It is a classic AI language, perfect for problems involving logic.
- **Expert Systems:** Its "if-then" rule system is ideal for building expert systems that mimic a human expert's decision-making.
- **Natural Language Processing (NLP):** Prolog is very good at parsing sentences and applying the rules of grammar.
- **Automated Reasoning:** It is used in systems that need to prove mathematical theorems or solve complex logical puzzles.

**Limitations of Prolog**

Prolog is very powerful for its specific tasks, but it is not good for everything:

- **Not for Numbers:** It is very bad at heavy mathematical calculations (numeric processing). It is designed for *symbolic* logic, not arithmetic.
- **Can be Inefficient:** The automatic backtracking feature is helpful, but if the problem is large or the rules are written badly, it can make the program very slow.
- **"Black Box" Reasoning:** It can be hard to follow *how* Prolog is finding an answer. Because it's not step-by-step, debugging a program and understanding its "flow" can be difficult.
- **Limited Scope:** It is not a good language for making things like user interfaces (graphics), web applications, or controlling computer hardware. It is highly specialized.

Limitations of Prolog
1) Not For Numbers
2) Can be inefficient
3) "Black box" Reasoning
4) Limited Scope

Q-2 Explain with Example how recursive predicate is defined in Prolog?

Ans How to define Recursive Predicate in Prolog.
Must define two parts
1) The Base Case (The "Stopping Condition")
→ This is simple fact that tells program when to stop
→ It must handle the simplest possible input (like 0, or an empty list)
→ If you forget this, you will create an infinite loop.

2) The Recursive Rule (The "Do-it-again" step)
→ This is the that calls itself
→ It must break the problem down into a smaller, simpler version that moves closer to Base Case

oHow Prolog does math
X = 5+1 X
X is 5+1 ✓

Example
Find factorial of given number

Goal : Factorial (number, Result)
Base Case: The simplest case is factorial of 0, which is 1
Recursive Rule:
→ the factorial of any number N is N times the factorial of (N-1)
→ We make the problem smaller (N-1)
→ We call the rule on that smaller number to get its result

Prolog Program

```
% Base Case : Factorial of 0 is 1
factorial (0, 1).

% Recursive Rule
factorial (N, Result) :-
    N > 0,
    N1 is N - 1,
    factorial ( N1, Result1),
    Result is N * Result1.
```

Q-3 Demonstrate the use of Cut and Fail predicates in Prolog with example

Ans

- The Fail Predicate
  - → Fail is a special goal that always fail
  - → Its only job is to force backtracking to start immediately
  - → It is like telling Prolog "This path is a dead end. Go back and try a different path right now."
  - → Example : The fail predicate is like finding a sign in a maze that says "I will never see a shop" You immediately turn around (backtrack)

- The cut predicate (The ! symbol)

→ The cut is written as an exclamation mark: !

→ It is a special goal that always succeeds (the first time it's seen), but it's main job is to stop future backtracking

→ Think of cut as building a one-way gate behind you

1) When Prolog runs a rule and passes a !, it commits to all the choices it made so far in that rule

2) If the rule "fails" after the Prolog, Prolog is not allowed to go past the ! to try other rules. The entire goal will fail

Example : The cut is like a one-way gate with a sign that says "I will definitely see a River" Once you go through you are committed. You cannot go back to the start and try a different path, even if this path leads to a dead end

# The Cut and Fail Combination

How it works : !, Fail

What it means: It tells Prolog, "If you match this rule, stop all searching and Fail immediately"
→ This is the main way to create negation in Prolog. It's how you tell Prolog, "I know for a Fact this one thing should Fail"

Example
Program about what sports Tom likes. We want to like all sports, except soccer

Code:

```
% These are Facts
sport(tennis).
sport(basketball).
sport(soccer).

% These are the rules
% Rule 1: Tom hates Soccer
likes(tom, X):-
    X = soccer,
    !,
    fail.

% Rule 2: Tom likes other sports
likes(tom, X):-
    sport(X).
```

Query 1 : likes (tom, tennis)

1) Prolog tries Rule 1 : tennis = soccer ? Fails
2) Prolog backtracks and then Rule 2
3) Rule 2 true

Query 2: likes (tom, soccer)

1) Rule 1 true
2) Cut (!) Succeeds. Prolog is now committed to this rule. It is not allowed to backtrack and try Rule 2
3) next : Fail
4) Fail means Prolog tries to backtrack, but the cut (!) stops it. The entire goal likes (tom, soccer) Fails
5) Result Fails

Q-3 Demonstrate the use of repeat predicate in Prolog with Eg.

Ans Repeat is a simple program you write to create a loop
→ It works by forcing Prolog to backtrack over and over, until a final condition is met

How to write: Add these two lines
repeat.
repeat :- repeat.

Eg:

Program will keep asking the user for a number until they enter a positive one

```
% this is main goal to run
go :-
    repeat,
    write ('Enter a positive number'),
    read (X),
        X > 0,
        !.      % this cut! stops the 'repeat' loop

% these two lines create the 'repeat' predicate
repeat.
repeat :- repeat.
```

Write a program for
1) Find factorial
2) to find nth element of given list

Ans 1) See Q-2

2) Code

```
% base case : when N = 1, the head is answer
nth_element (1, [H|_], H).

% Recursive case: decrease N and move to tail
nth_element (N, [_|T], X) :-
        N > 1,
        N1 is N-1,
        nth element (N1, T, X)
```