ARTIFICIAL INTELLIGENCE

SEMESTER EXAM SERIES

IN 6 HOURS + NOTES

Artificial Intelligence

- **Chapter-1** (INTRODUCTION)
- **Chapter-2** (PROBLEM SOLVING METHODS)
- **Chapter-3** (KNOWLEDGE REPRESENTATION)
- **Chapter-4** (SOFTWARE AGENTS)
- **Chapter-5** (APPLICATIONS)

http://www.knowledgegate.in/gate

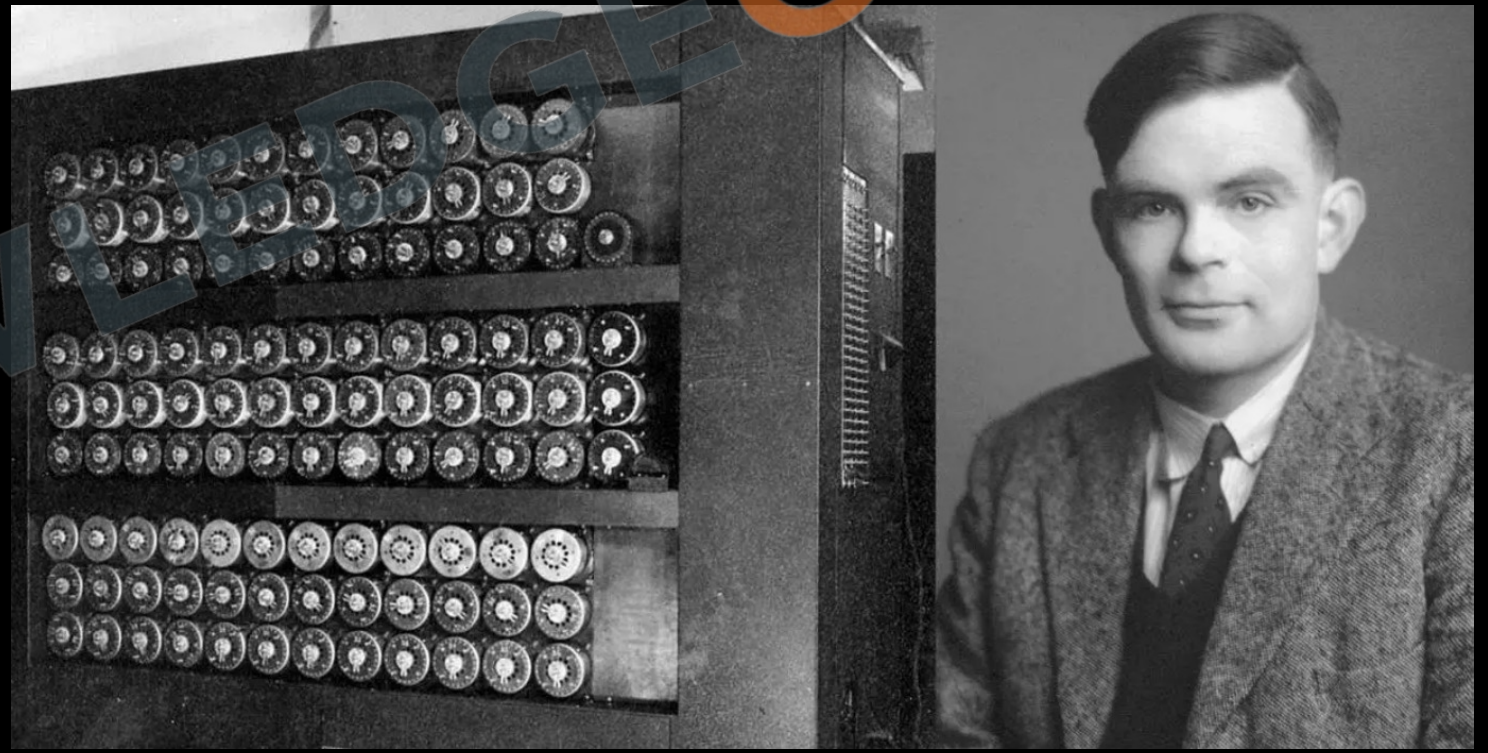
Video chapters

- **Chapter-1 (**INTRODUCTION**):** Introduction-Definition - Future of Artificial Intelligence - Characteristics of Intelligent Agents - Typical Intelligent Agents - Problem Solving Approach to Typical Al problems.
- **Chapter-2 (**PROBLEM SOLVING METHODS**):** Problem solving Methods - Search Strategies- Uninformed - Informed - Heuristics - Local Search Algorithms and Optimization Problems - Searching with Partial Observations - Constraint Satisfaction Problems - Constraint Propagation - Backtracking Search - Game Playing - Optimal Decisions in Games - Alpha - Beta Pruning - Stochastic Games.
- **Chapter-3 (**KNOWLEDGE REPRESENTATION**):** First Order Predicate Logic - Prolog Programming - Unification - Forward Chaining - Backward Chaining - Resolution - Knowledge Representation - Ontological Engineering-Categories and Objects - Events - Mental Events and Mental Objects - Reasoning Systems for Categories - Reasoning with Default Information.
- **Chapter-4 (**SOFTWARE AGENTS**):** Architecture for Intelligent Agents - Agent communication - Negotiation and Bargaining - Argumentation among Agents - Trust and Reputation in Multi-agent systems.
- **Chapter-5 (**APPLICATIONS**):** Al applications - Language Models - Information Retrieval - Information Extraction - Natural Language Processing - Machine Translation - Speech Recognition - Robot -Hardware – Perception – Planning – Moving.

- **Chapter-1 (**INTRODUCTION**):** Introduction-Definition - Future of Artificial Intelligence - Characteristics of Intelligent Agents - Typical Intelligent Agents - Problem Solving Approach to Typical AI problems.

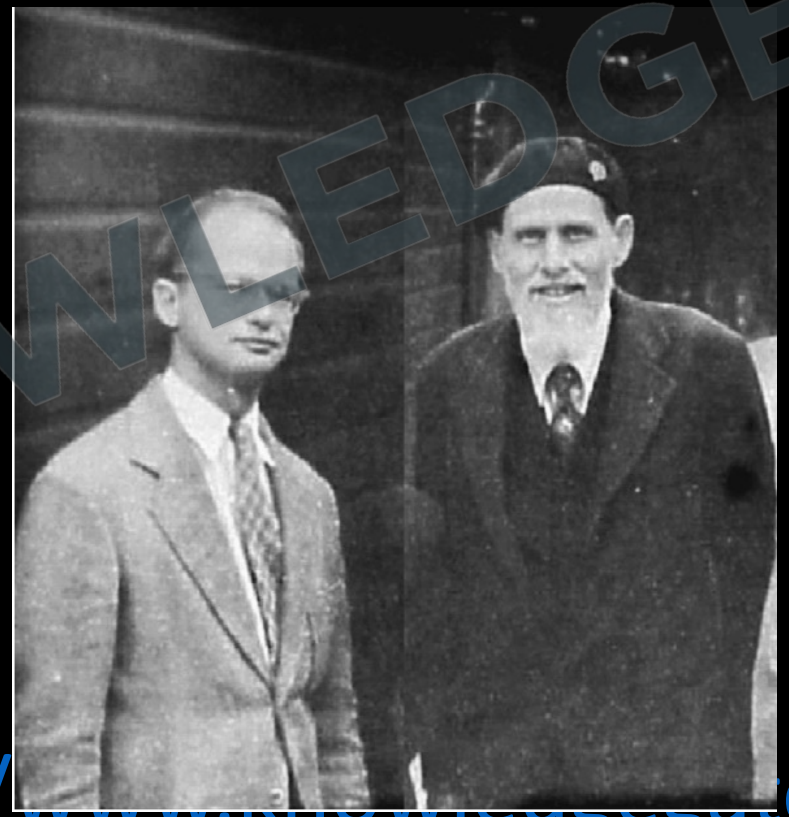
History of AI

- During the Second World War, British computer scientist **Alan Turing** worked to crack the 'Enigma' code which was used by German forces to send messages securely. Alan Turing and his team created the Bombe machine that was used to decipher Enigma's messages.
- The Enigma and Bombe Machines laid the foundations for Machine Learning.



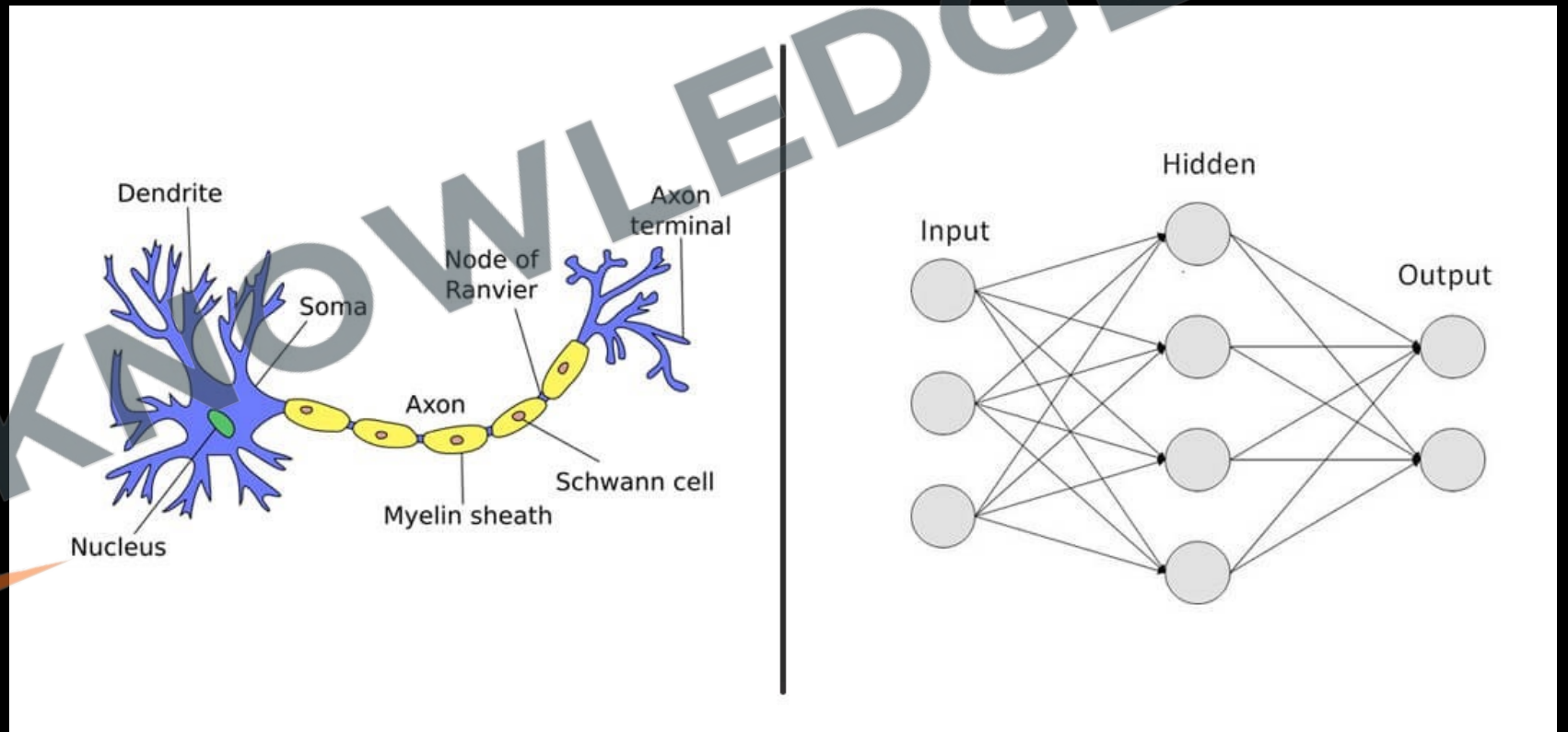
First work in AI

- The first work that is now generally recognized as AI was done by Warren McCulloh and Walter Pits in 1943. They drew on three sources:
  - **Knowledge of basic Physiology and Functions of Neurons in brain.**
  - **A formal analysis of propositional logic**
  - **Turing's Theory of Computation**



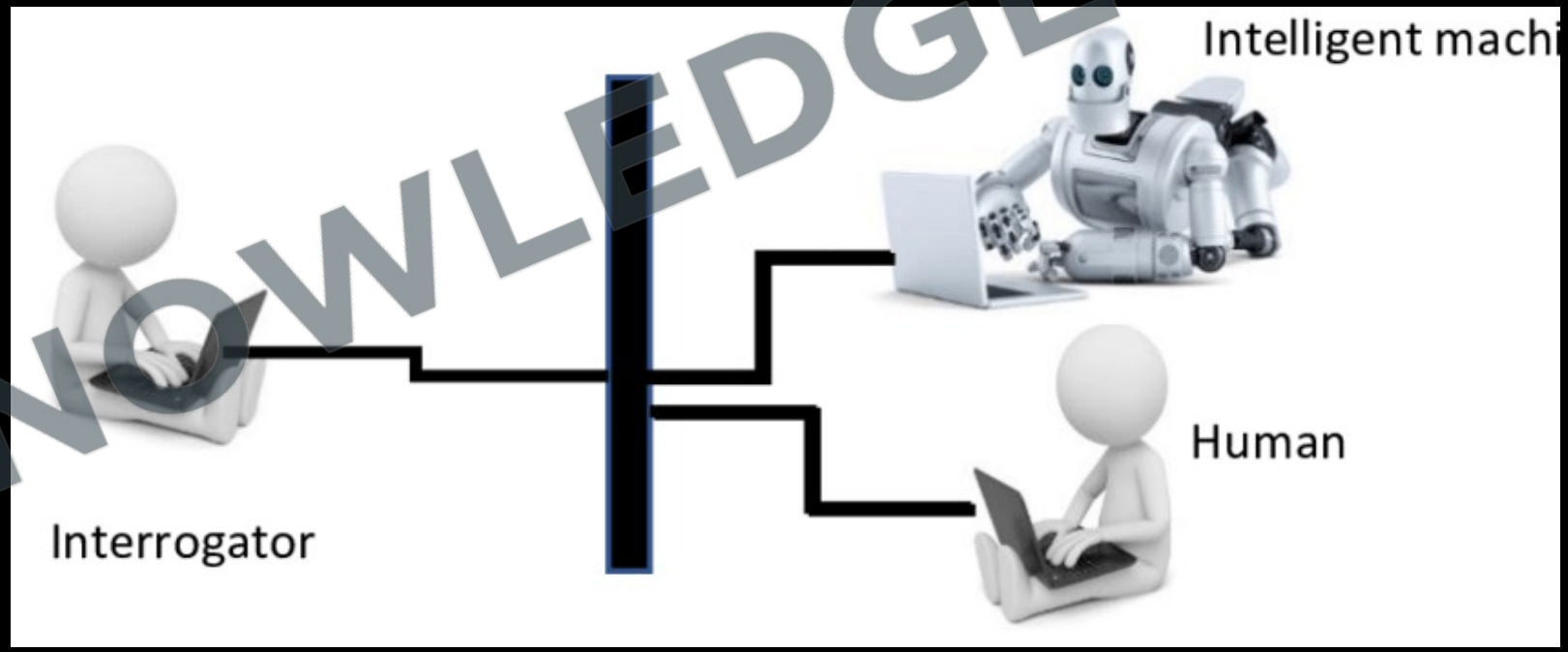
Key Findings

- They proposed a model of artificial neurons in which each neuron can be characterized as being **on** and **off**.
- They showed that any computable function could be computed by some network of connected neurons.
- They also suggested that suitably defined networks can also learn.



Acting humanly: The Turing Test approach

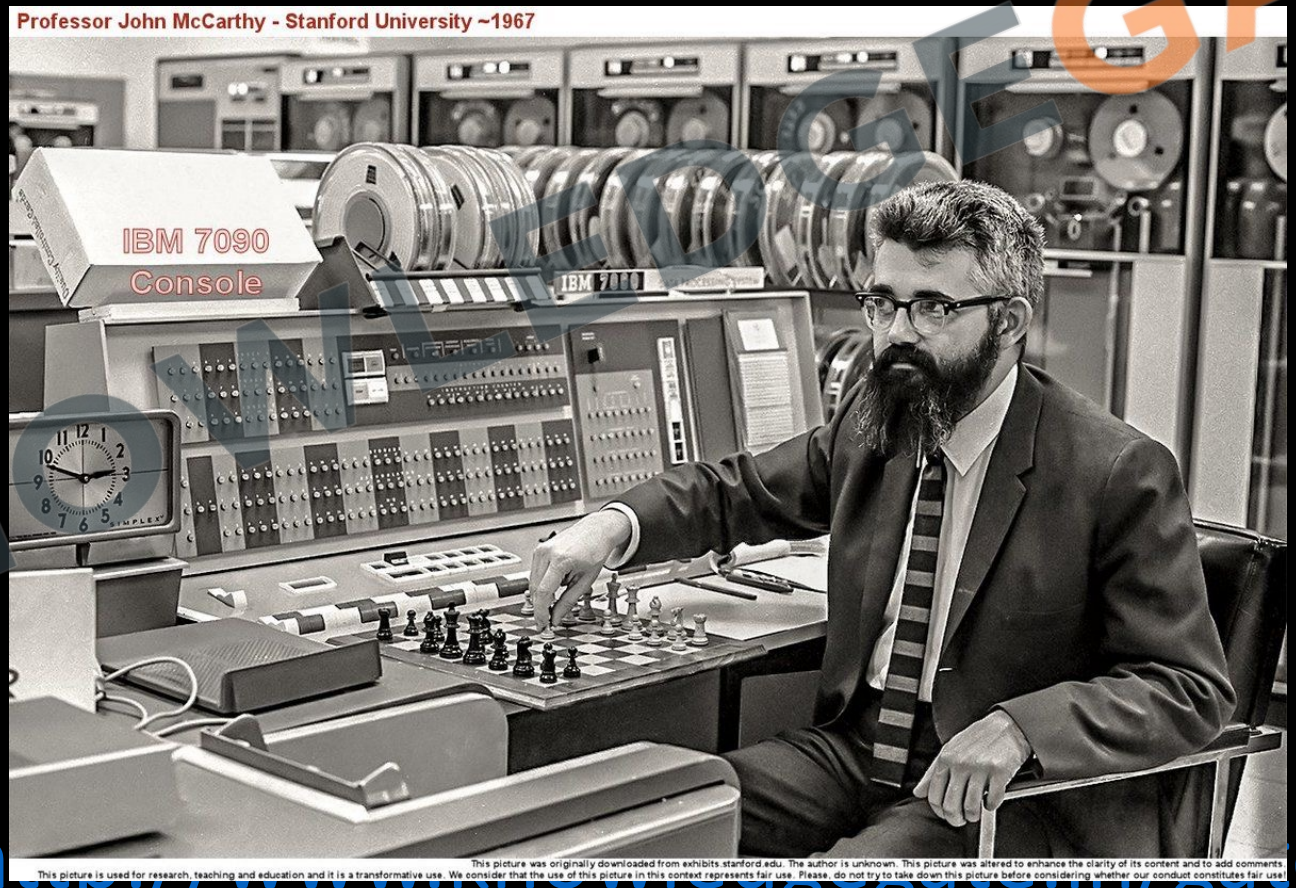
- The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence.
- A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.



- Marven Minsky and Dean Edmonds from Harvard built the first neural network computer in 1950, it was called "***The SNARC***".

The Birth of AI

- The term AI was coined in 1956 by a computer scientist “John McCarthy” in Dartmouth Conference organized by him and three other scientists. He is often known as “***Father of AI”.*** McCarthy in 1958 developed LISP. There were so many discoveries going on in the field of AI from the year 1952 to 1969.

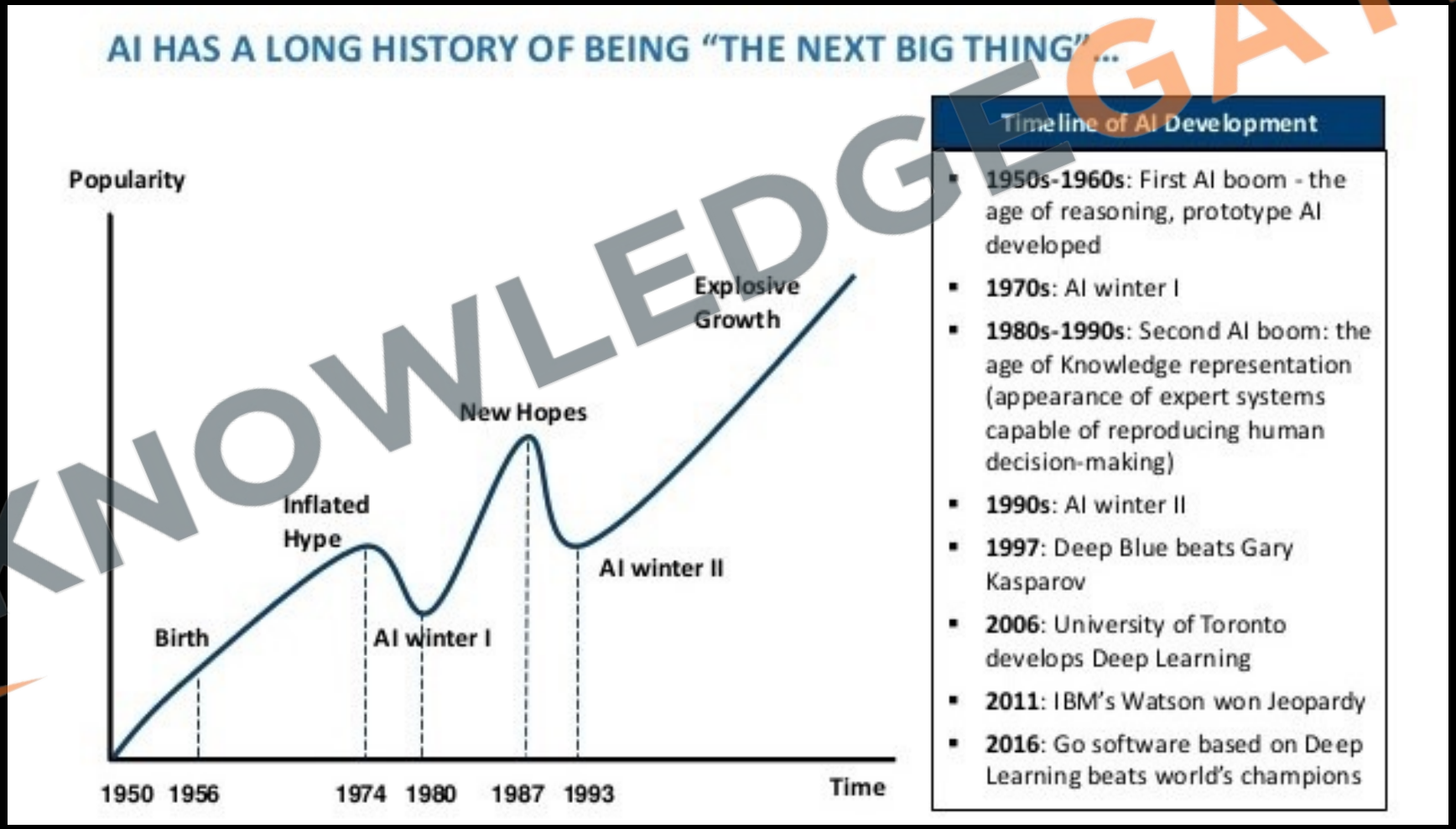


AI Winter (1974-1980)

- In 1970's AI was subjected to criticism and financial setbacks. The expectations from AI was quite high due to heavy optimism but the researchers had failed to materialize the promised results.

Boom in AI after 1980 (1980-87)

- In 1980's "Expert Systems" an AI program was adopted by many corporations. **The Expert system was a program that answers questions or solves problems from a particular domain using logical rules derived from the knowledge of experts.**

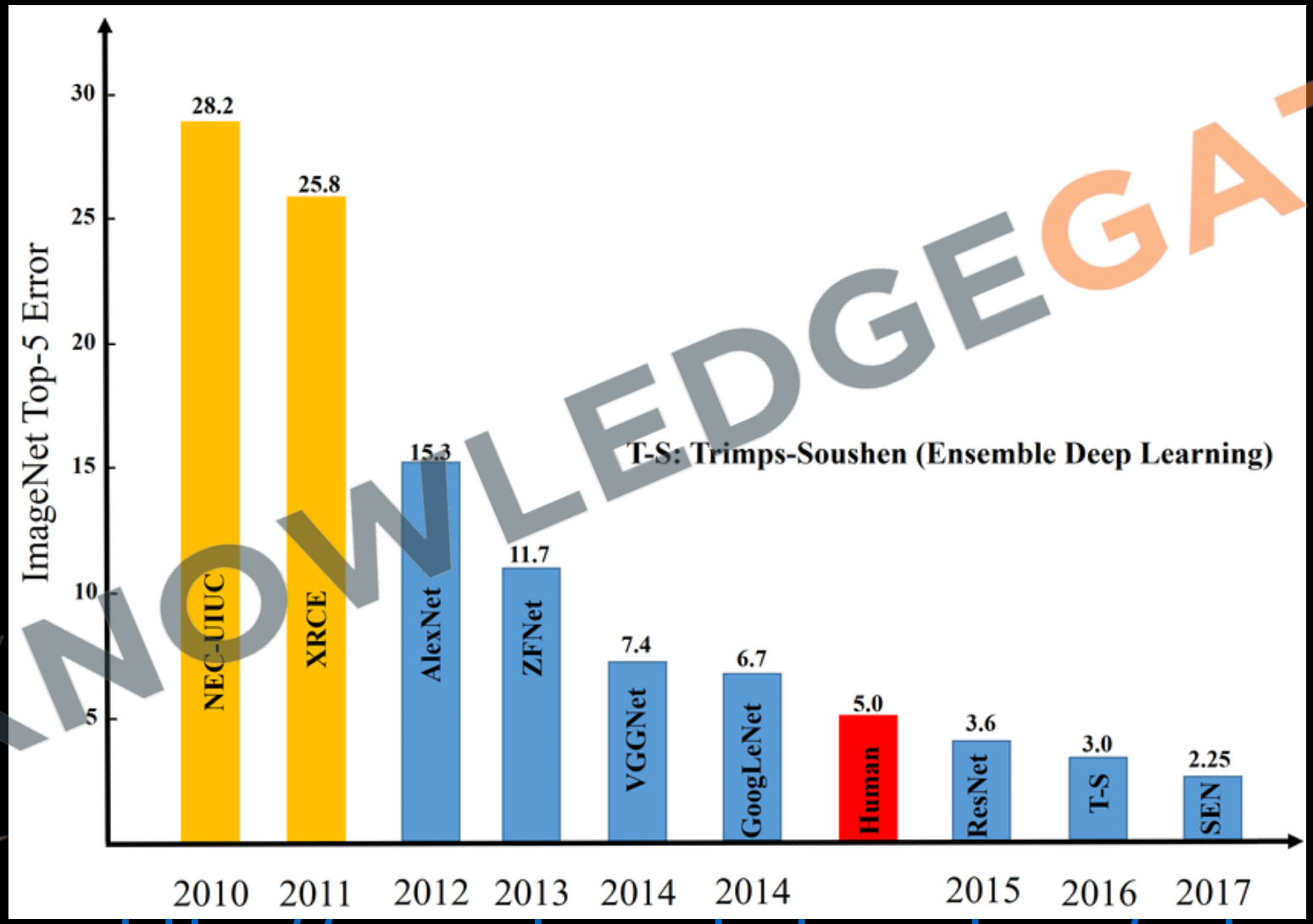


- **1997:** IBM's Deep Blue beats the world chess champion, Garry Kasparov, a significant milestone for AI in games.

- **2005:** Stanley, the self-driving car developed by Stanford University, wins the DARPA Grand Challenge.

- **2011:** IBM's Watson wins Jeopardy! against former winners, showing the capabilities of AI in natural language processing.

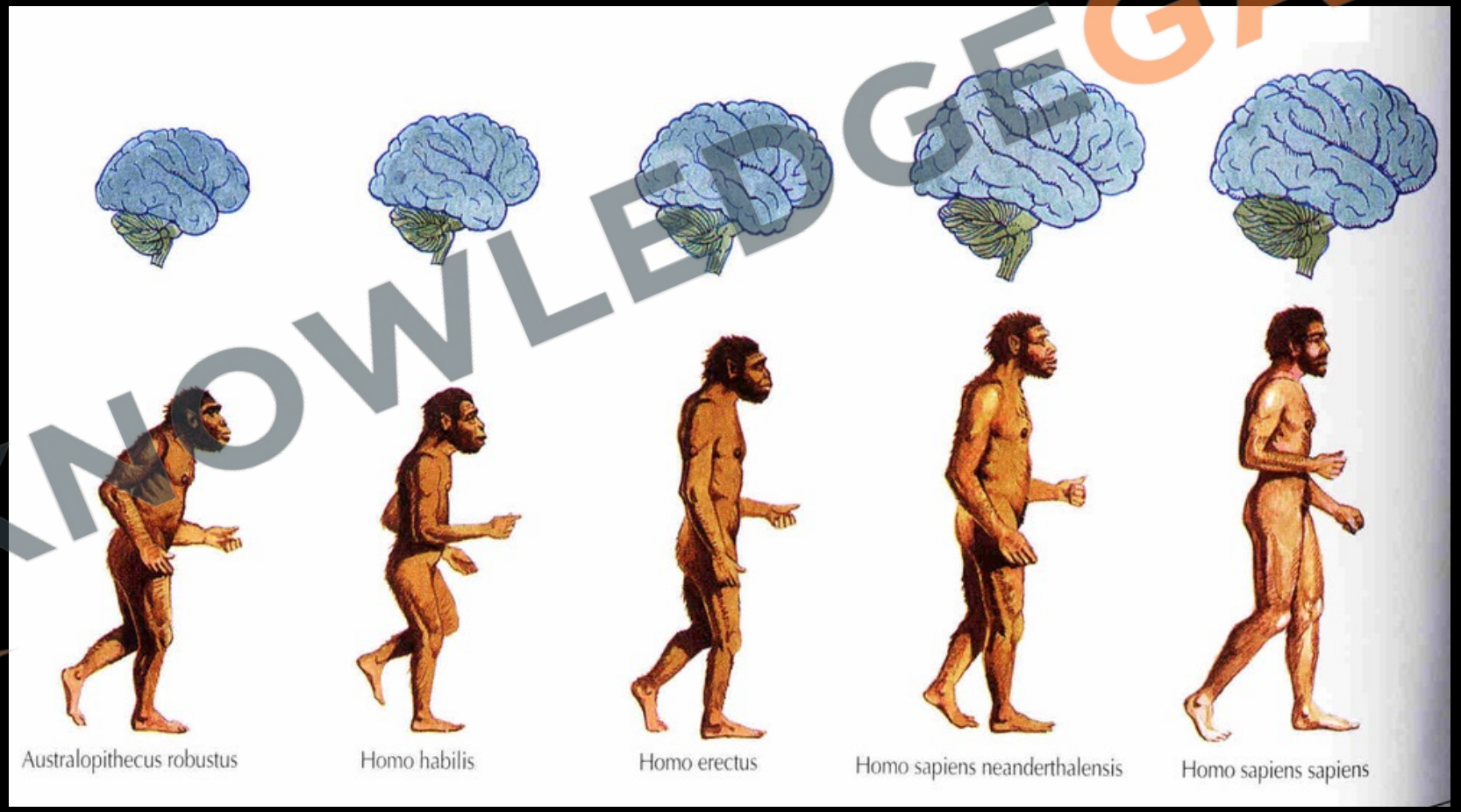
- **2015:** A breakthrough in deep learning occurs when a neural network wins the ImageNet competition, significantly improving the state of image recognition.



- **2016:** Google's AlphaGo defeats world champion Go player Lee Sedol, a significant milestone due to the complexity of the game.

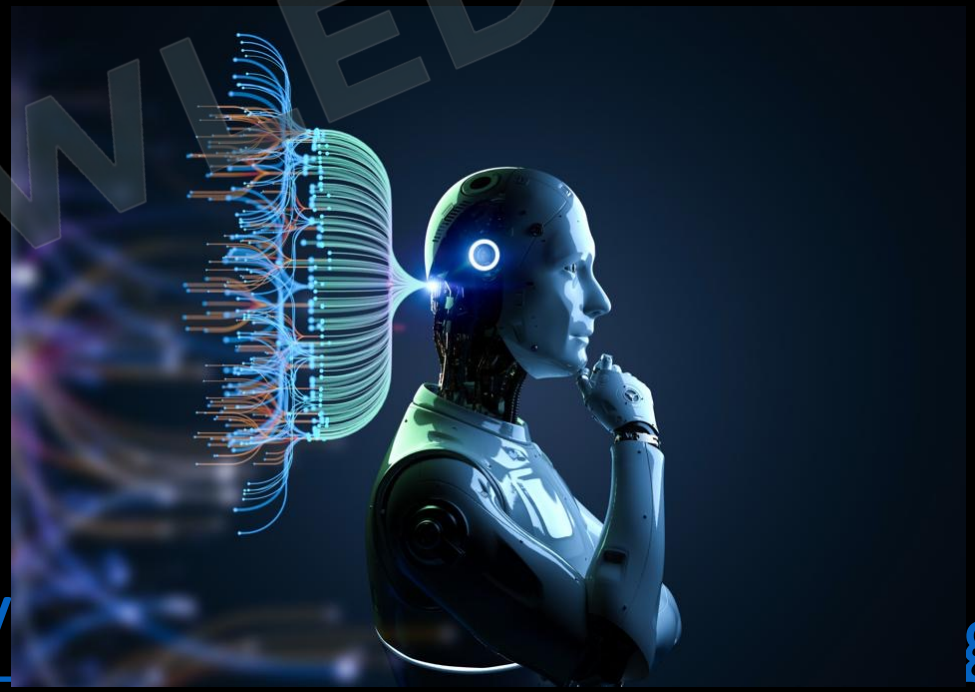
- **2018:** OpenAI's GPT showcases a leap in natural language processing capabilities.

- **Intelligence and AI:** Intelligence in general is the "**Ability to acquire and apply knowledge and skills".** We are known as Homo-Sapiens "Homo- ***Man*** and Sapiens - ***Wise***" it is because our intelligence is so important to us.
- For thousands of years we have tried to understand how our brain functions. The field of AI goes beyond this: It not only attempts to understand it but also tries to build intelligent entities.



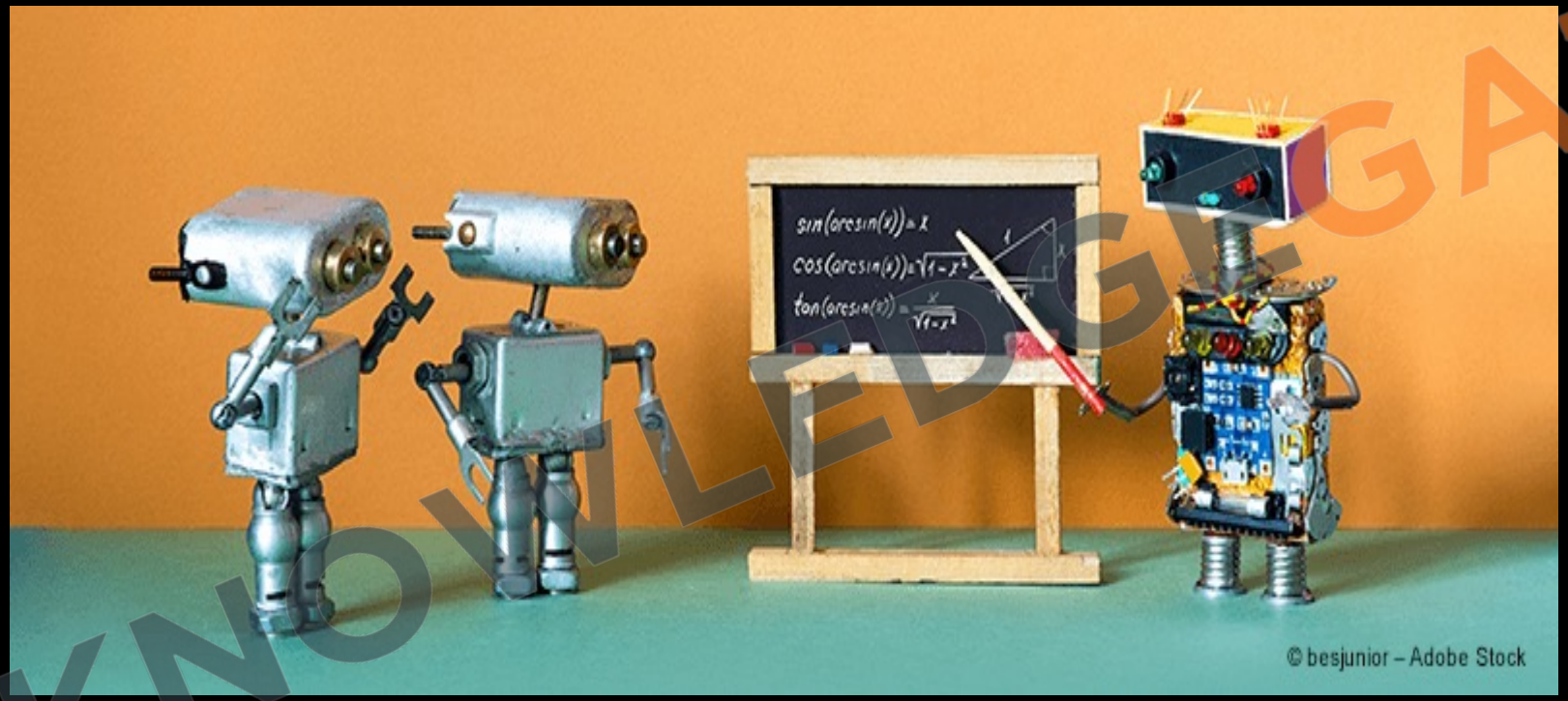
What is AI

- There is no universally accepted definition of AI in general refers to the simulation of human intelligence processes by machines, especially computer systems.
- **Sub-fields:** It includes various sub-fields like machine learning, where machines learn from experience, natural language processing, which is about interaction with human language, and computer vision, where AI interprets and understands visual data.

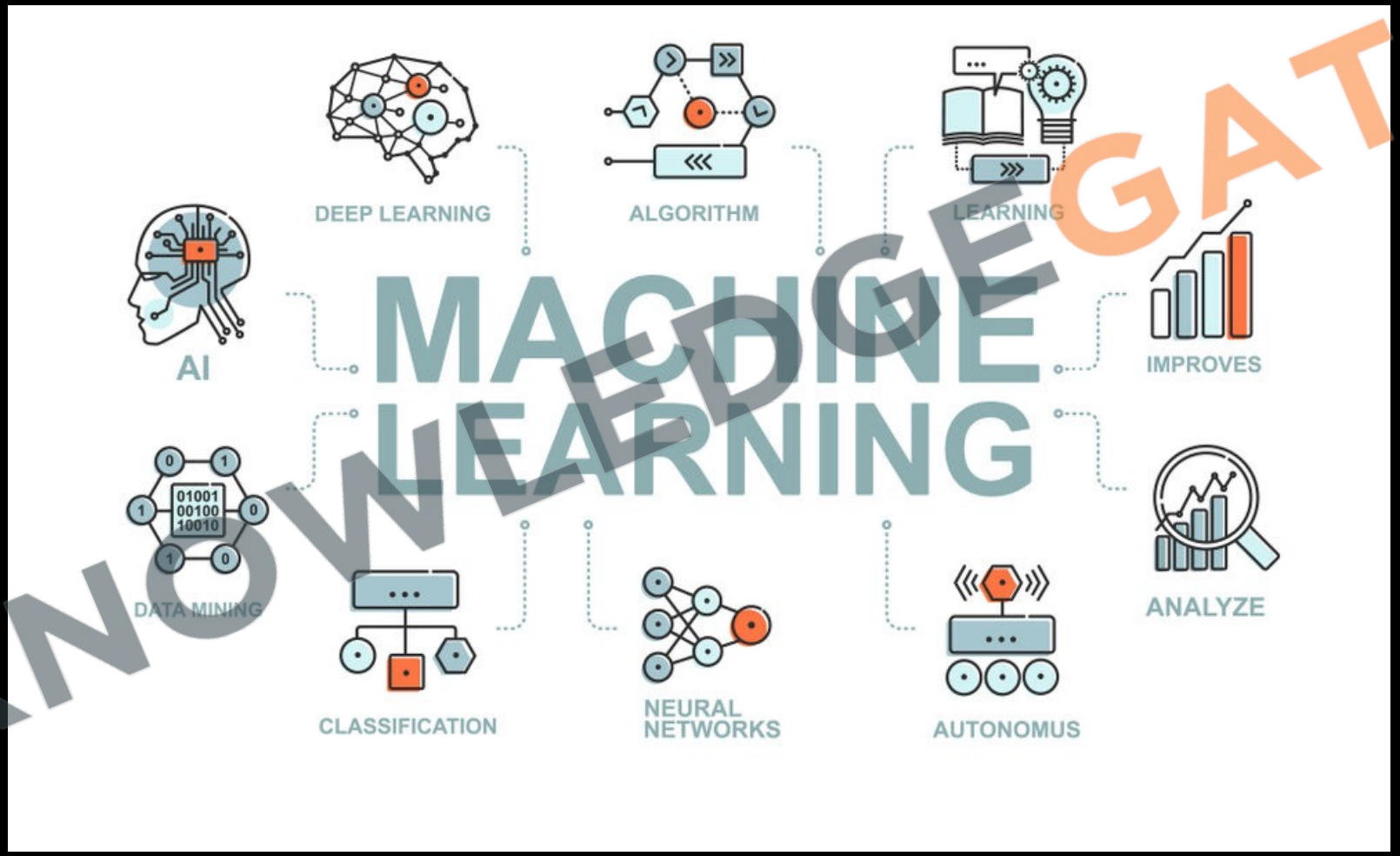


- **<u>Problem-solving</u>:** AI is used to solve complex problems and perform tasks that would usually require human intelligence. This could be anything from **recognizing speech or images, making decisions, or translating languages**.

- **Training:** AI systems are trained using large amounts of data. For instance, an AI used for recognizing speech would be trained on a large database of spoken language.

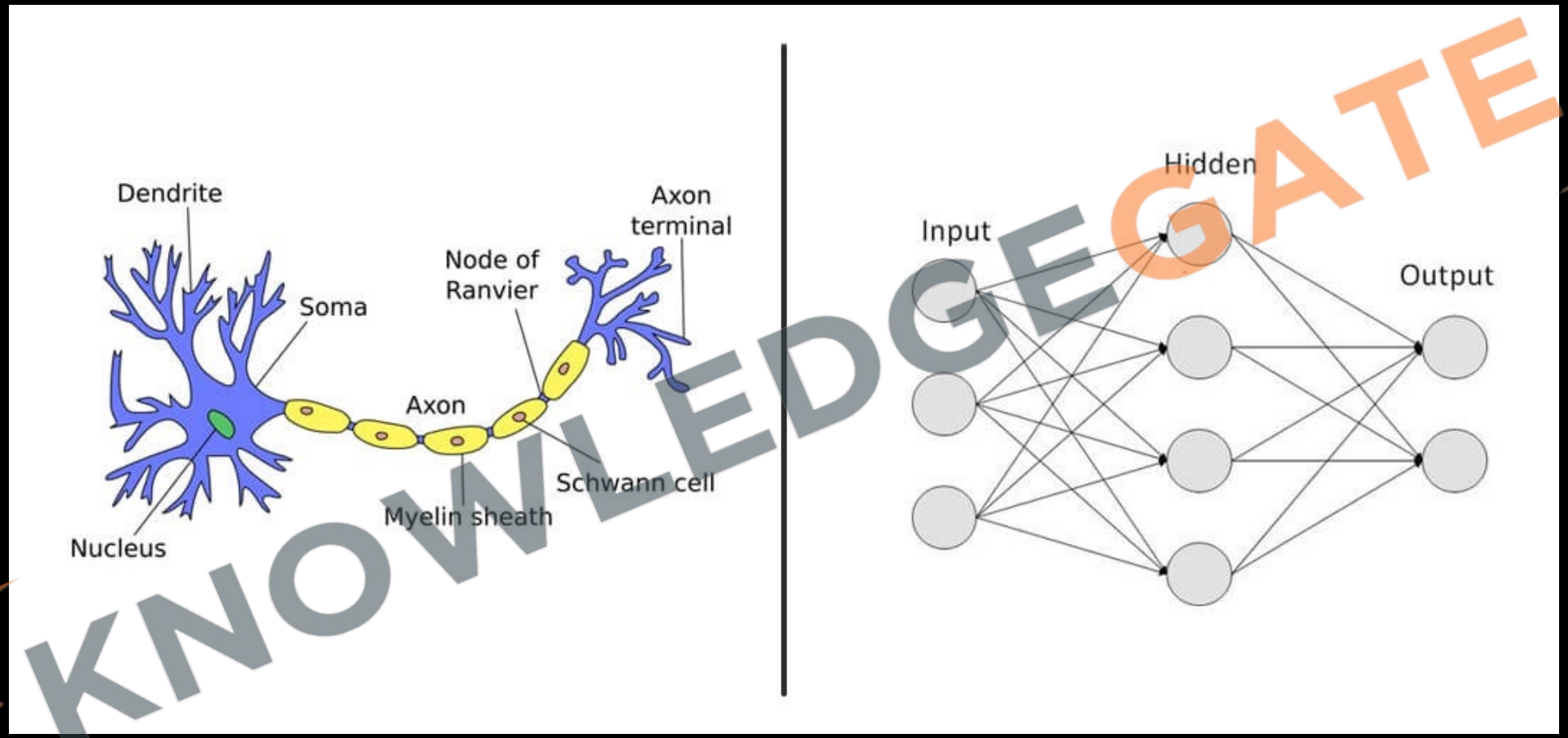


- **<u>Machine Learning</u>:** One of the most common types of AI in use today is based on machine learning, where AI gets smarter the more data it is given.



http://www.knowledgegate.in/gate

- **Neural Networks:** AI systems often use neural networks, which are designed to mimic the human brain's ability to recognize patterns and make decisions.



- **Benefits:** AI has many potential benefits, such as improving efficiency and productivity, making faster and more informed decisions, and performing tasks that are dangerous for humans. The future potential of AI is vast, with ongoing research in fields like self-driving cars, voice assistants, predictive healthcare, and many more.

Artificial Intelligence (AI) can be classified into four categories based on capabilities and functionalities:

- **Reactive Machines:**
  - These are the most basic type of AI.
  - They can react to specific situations or inputs but don't have memory or past experience to influence their decisions.
  - Example: IBM's Deep Blue, the chess-playing AI.

http gate

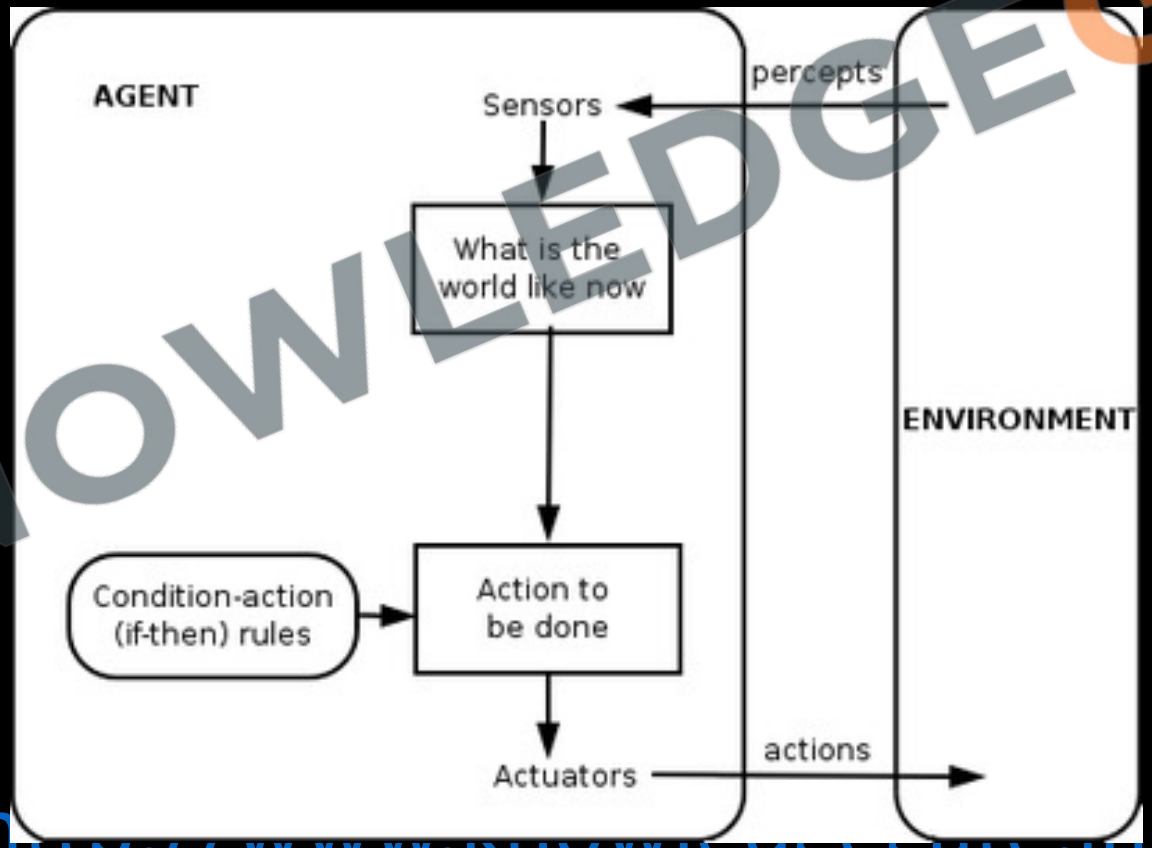
- **Limited Memory:**
  - These AI systems can use past experiences or historical data to make current decisions.
  - They have a temporary or limited memory to store past information and predictions.
  - Example: Self-driving cars.

- **Theory of Mind:**
  - This is a more advanced type of AI that researchers are still working to develop.
  - These systems would understand emotions, people, beliefs, and be able to interact socially.
  - Example: More advanced personal assistants that can understand human emotions and react accordingly.

- **Self-aware AI:**
  - This is the most advanced form of AI, which is still theoretical and not yet realized.
  - These systems would have their own consciousness, self-awareness, and emotions.
  - They would be smarter and more capable than the human mind.

AGENTS AND ENVIRONMENTS

- An agent is anything that can be viewed as perceiving its **environment** through sensors and acting upon that environment through **actuators.**
- A human agent has eyes, ears, and other organs for sensors are hands, legs, vocal tract, and so on for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.



- **Percept:** This refers to what an AI agent (like a robot or a computer program) senses or perceives at a specific moment.
- **Percept Sequence:** This is the total history or record of what the AI agent has sensed or perceived since it started operating. An AI agent's decisions or actions at any moment can be influenced by the entire percept sequence it has experienced so far. However, it can't base decisions on information it hasn't perceived or sensed.
- **Agent Function:** This is a rule or set of rules that determines what action an AI agent takes. based on its percept sequence.
- **Agent Program:** This is the practical, real-world implementation of the agent function.

Concept of Rationality in AI

- **Rational Agent:** An AI entity that always chooses the best action.
- **Action Selection:** Based on its percepts, a rational agent selects an action that's expected to maximize its performance measure, using its percept history and innate knowledge.
- **Performance Measure:** Not a fixed measure for all tasks and agents, it varies based on the specific task at hand.
- **Designing Performance Measures:** Ideally, these are designed based on desired outcomes in the environment, not preconceived ideas about the agent's behavior.

Understanding Task Environments and Rational Agents

- **Task Environment:** a task environment is essentially the setting or context in which an AI agent operates. This could be a physical environment, like a warehouse where a robot is sorting boxes, or a virtual environment, like a chess game where an AI is playing against a human opponent. each task environment presents a unique set of challenges or problems that the AI agent needs to navigate.
- the task environment of a chess game includes the rules of the game, the current state of the board, and the goal of checkmating the opponent's king. The "problem" in this case is determining the best move to make at each step to win the game. In the chess example, a rational agent would be an AI system that can analyze the state of the game, consider all possible moves, and choose the one that maximizes its chances of winning.
- A rational agent, then, is the "solution" to this problem: it's an AI system designed to make the best possible decisions given the circumstances of the task environment.

- Acronymically we call it **PEAS (P**erformance, **E**nvironment, **A**ctuators, **S**ensors**)** description. **PEAS for agent Taxi Driver:**

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Properties of task environments

- **Fully Observable vs. Partially Observable:**
  - **Fully Observable:** Every relevant aspect in the environment is visible to the agent.
  - **Partially Observable:** Not all aspects are visible. The agent doesn't have complete information about the environment's state because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.
  - **Unobservable:** If the agent has no sensors at all then the environment is **unobservable**.

Single Agent vs. Multi-Agent:

- **Single Agent:** Only one agent is acting in the environment, like playing solitaire.
- **Multi-Agent:** Multiple agents are interacting with the environment and potentially with each other, like playing chess.
- Driving taxi requires avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative** multiagent environment. It is also partially competitive because only one car can occupy a parking space.

Deterministic vs. Stochastic:

- **Deterministic:** The next state of the environment is entirely determined by the current state and the action executed by the agent.
- **Stochastic:** The next state cannot be entirely predicted and might depend on probability, like playing poker.

Episodic vs. Sequential:

- **Episodic:** The agent's experience is divided into episodes. Each episode is independent of others, like in image recognition tasks.
- **Sequential:** The current decision could influence all future decisions, like in a game of chess.

Static vs. Dynamic:

- **Static:** The environment doesn't change while the agent is deciding, like a game of tic-tac-toe.
- **Dynamic:** The environment can change while the agent is making a decision, like in stock trading.

Discrete vs. Continuous:

- **Discrete:** There are a finite number of actions the agent can take, like moves in a chess game.
- **Continuous:** The actions or states can vary in a continuous way, like a self-driving car controlling its speed.

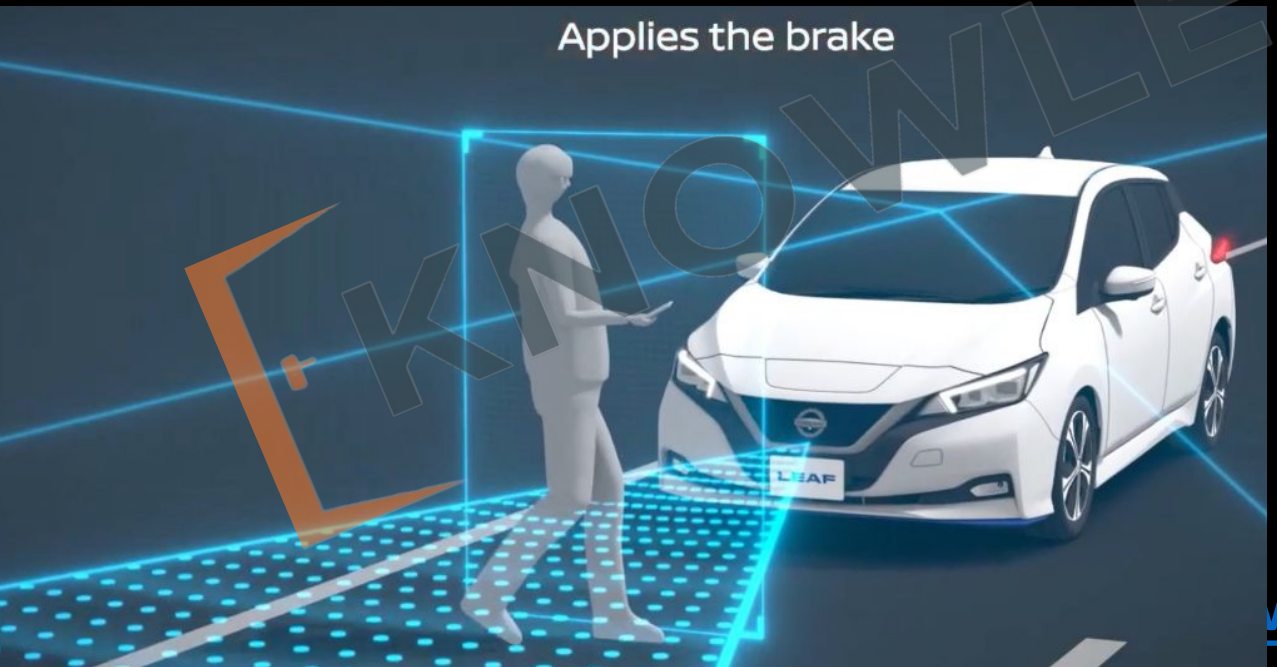
Known vs. Unknown:

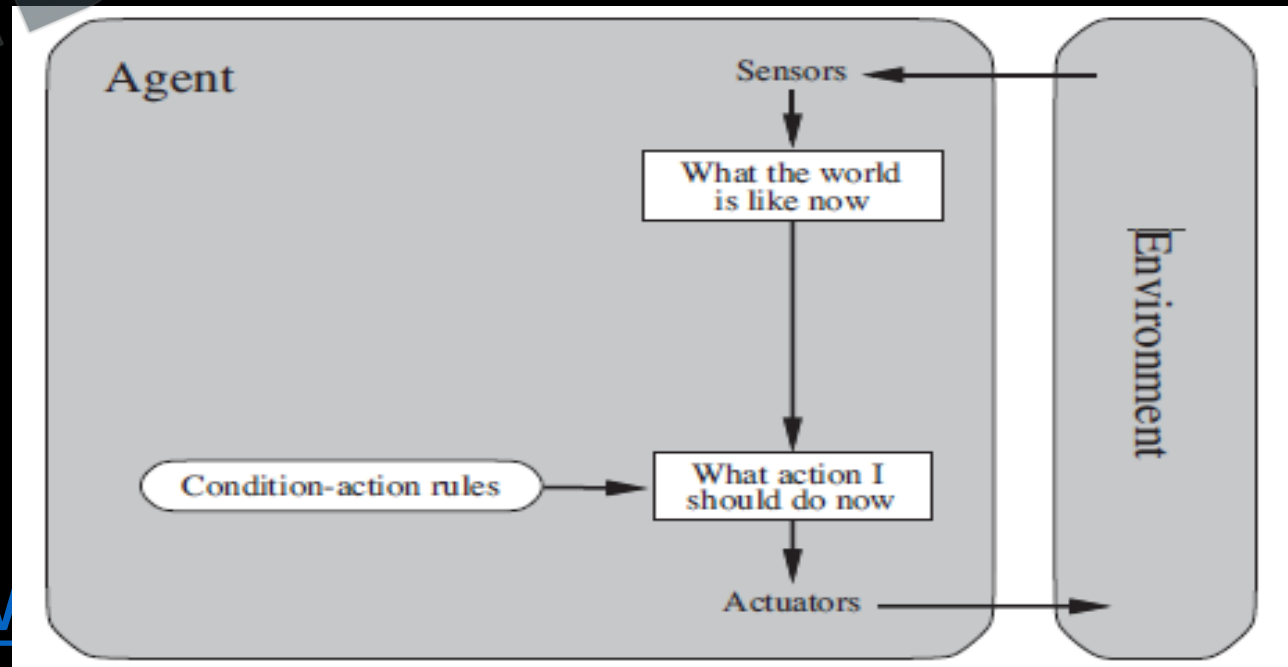
- **Known:** The agent knows the rules of the environment, like in a game of checkers.
- **Unknown:** The agent doesn't fully understand the rules of the environment and must learn them, like navigating a new city.

- Four basic kinds of agent programs that embody the principles underlying almost all intelligent systems are:
  - **Simple reflex agents;**
  - **Model-based reflex agents;**
  - **Goal-based agents; and**
  - **Utility-based agents.**

Simple reflex agents

- **Basic Functioning:** Simple reflex agents act based on current percepts, ignoring historical percepts.
- **Condition-Action Rule:** These agents follow "if-then" rules, like "if someone is in front of the car then initiate-braking."
- **Simplicity and Limitations:** While such agents are simple to design do not perform complex calculation, their intelligence and memory is limited.
- **Problems with Unobservability:** Issues arise if all necessary information isn't observable, like different configurations of car lights.

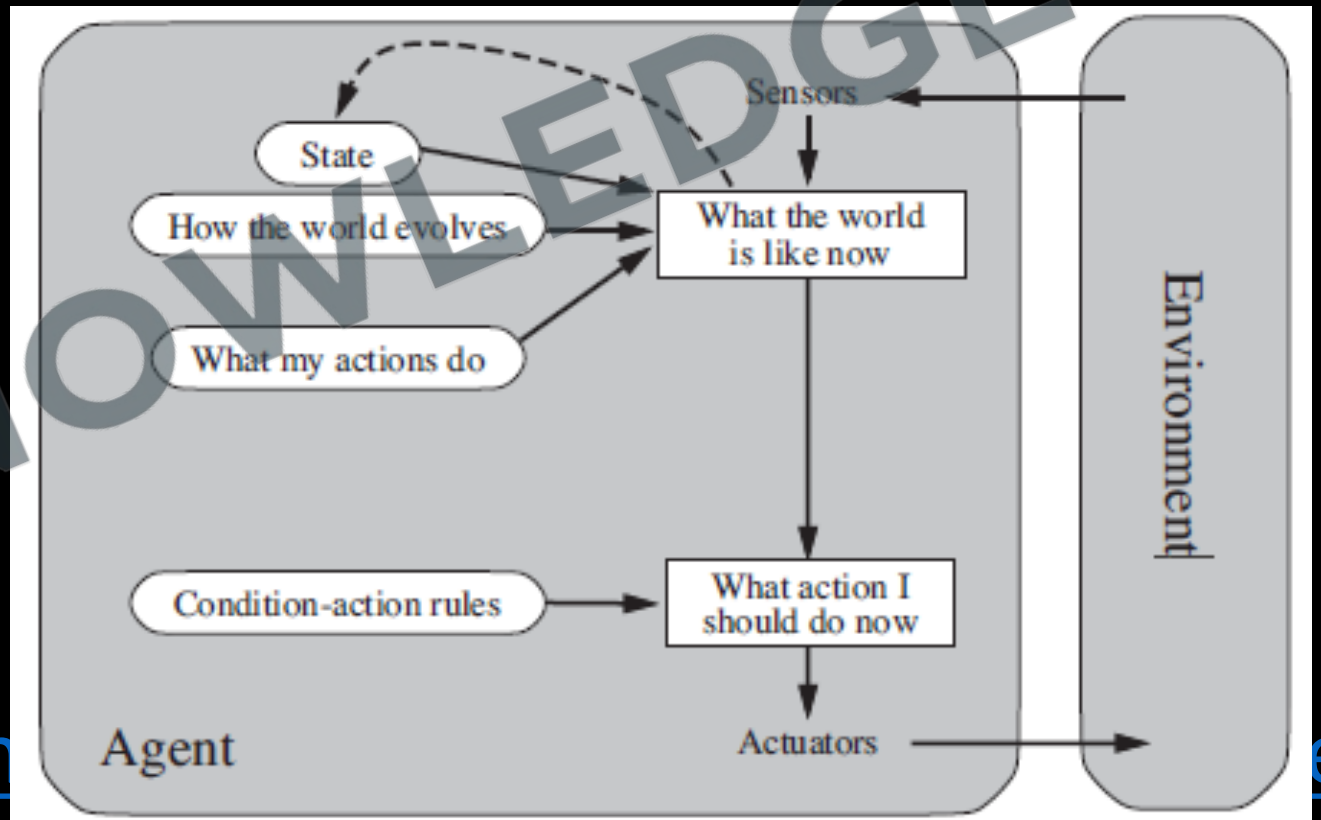




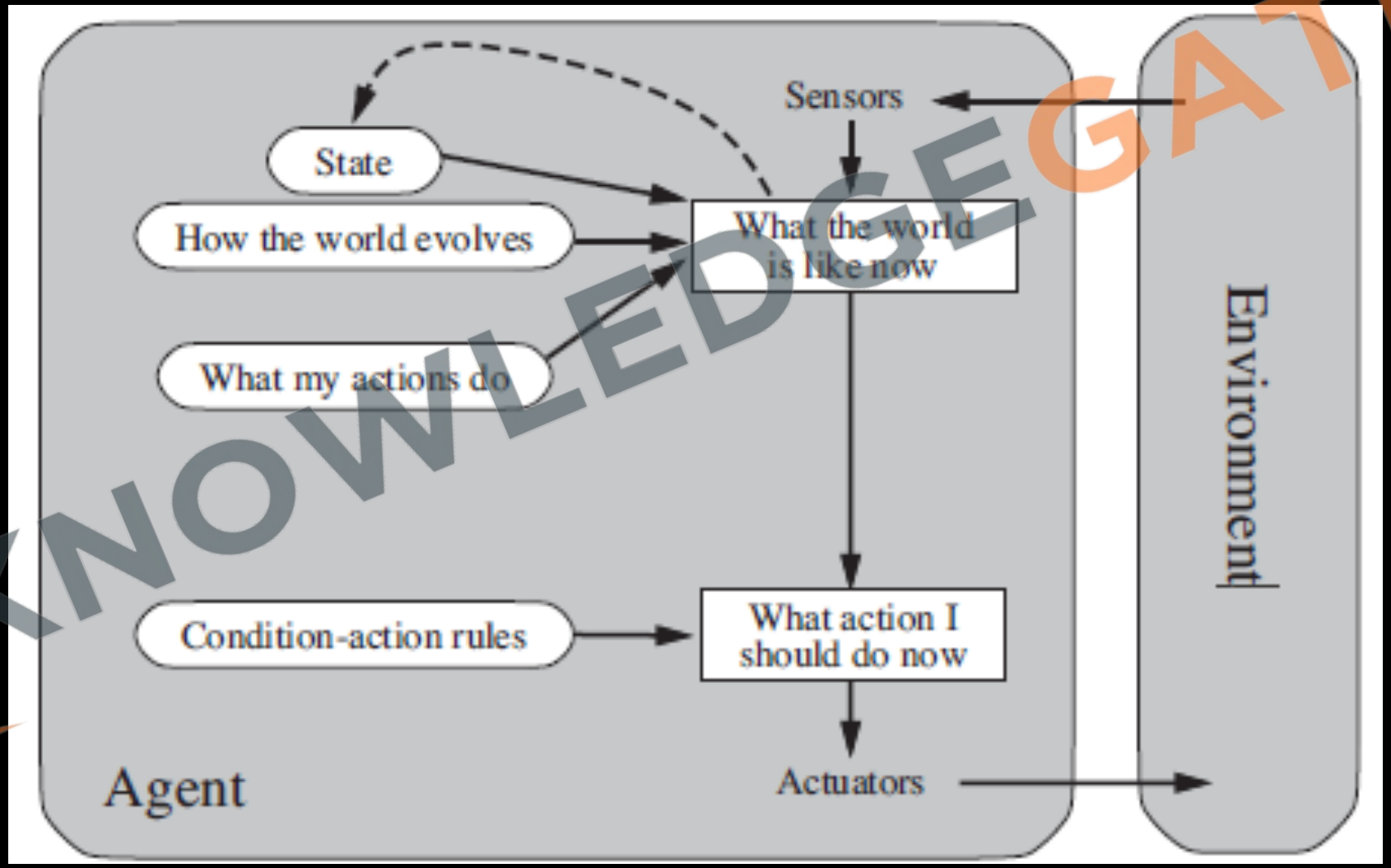
http://www.knowledgegate.in/gate

Model-based reflex agents

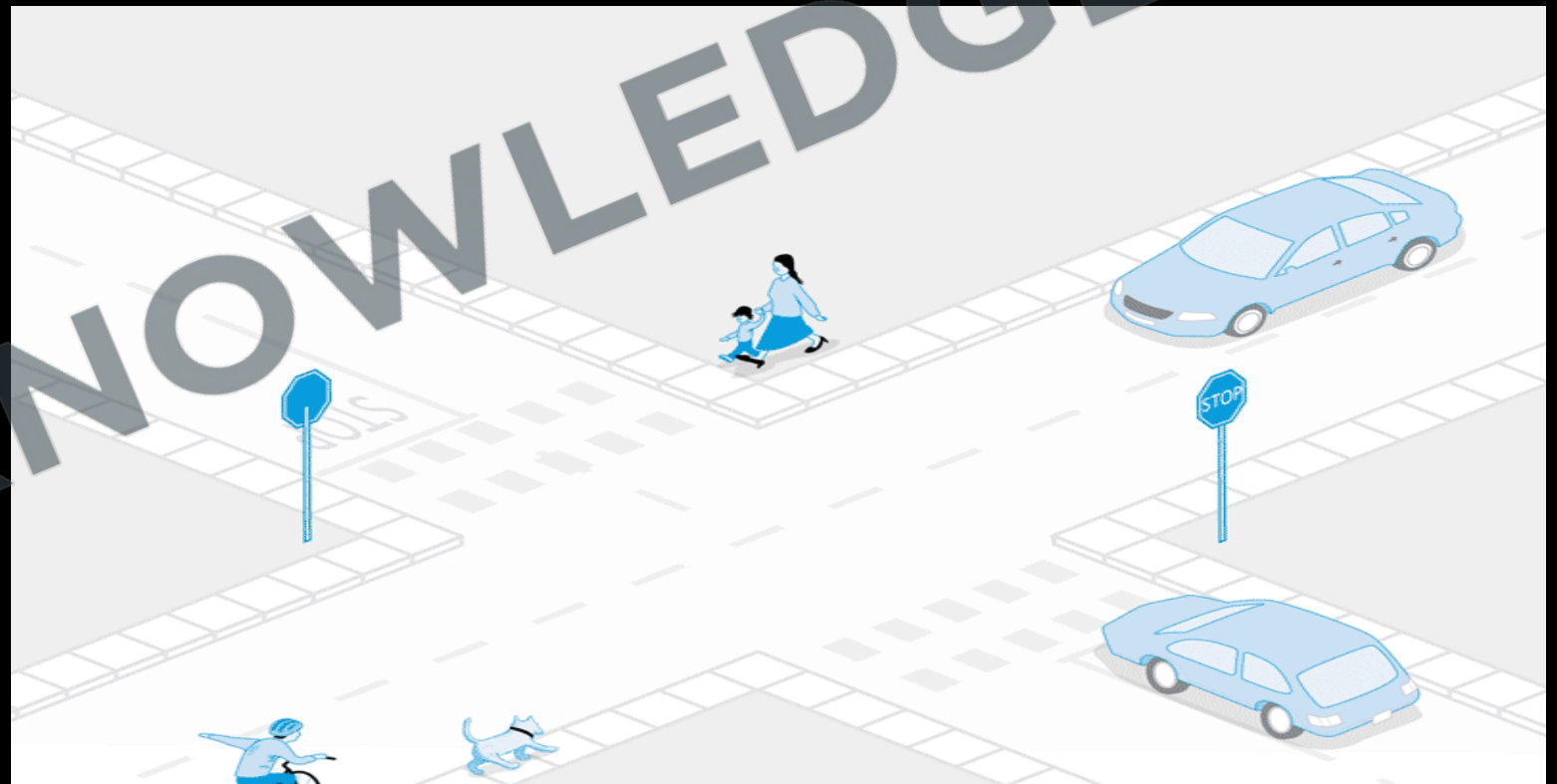
- **Handling Partial Observability:** Model-based agents is designed to work in partially observable environment.
- **Internal State:** These agents maintain an internal state that reflects unobserved aspects of the environment, based on percept history.
- **Updating Internal State:** This process requires knowledge of how the world evolves independently of the agent and how the agent's actions affect the world.



- **Model of the World:** Agents use this knowledge, whether simple or complex, to form a model of how the world works, and then take actions accordingly.

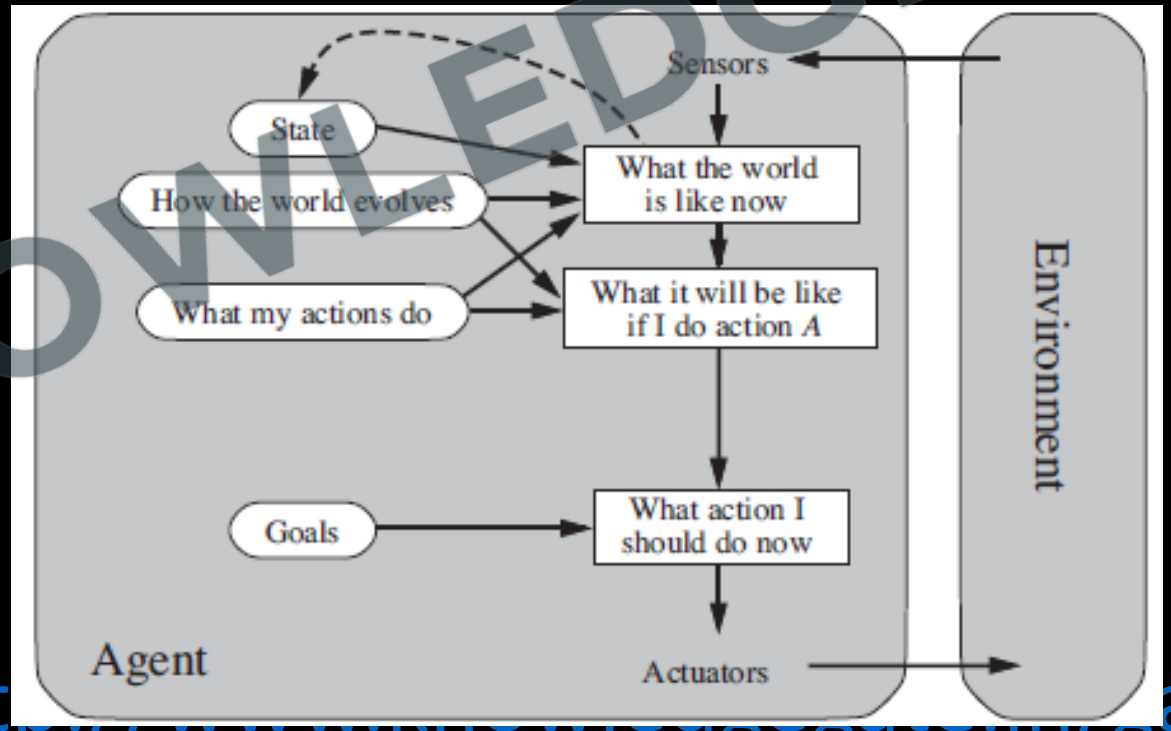


- **Internal Model**: The car has a map of the roads, traffic rules, and typical traffic patterns.
- **Sensors**: Uses cameras and sensors to perceive traffic lights, road signs, other vehicles, pedestrians, and road conditions.
- **Updating Knowledge**: It updates its model in real-time as it drives, learning current traffic conditions, construction work, or detours.
- **Decision Making**: If the car approaches a red light, it stops. When the light turns green and the path is clear, it proceeds.
- **Learning Over Time**: The car refines its internal model to improve predictions, like estimating the time it will take to reach destinations considering the current traffic.

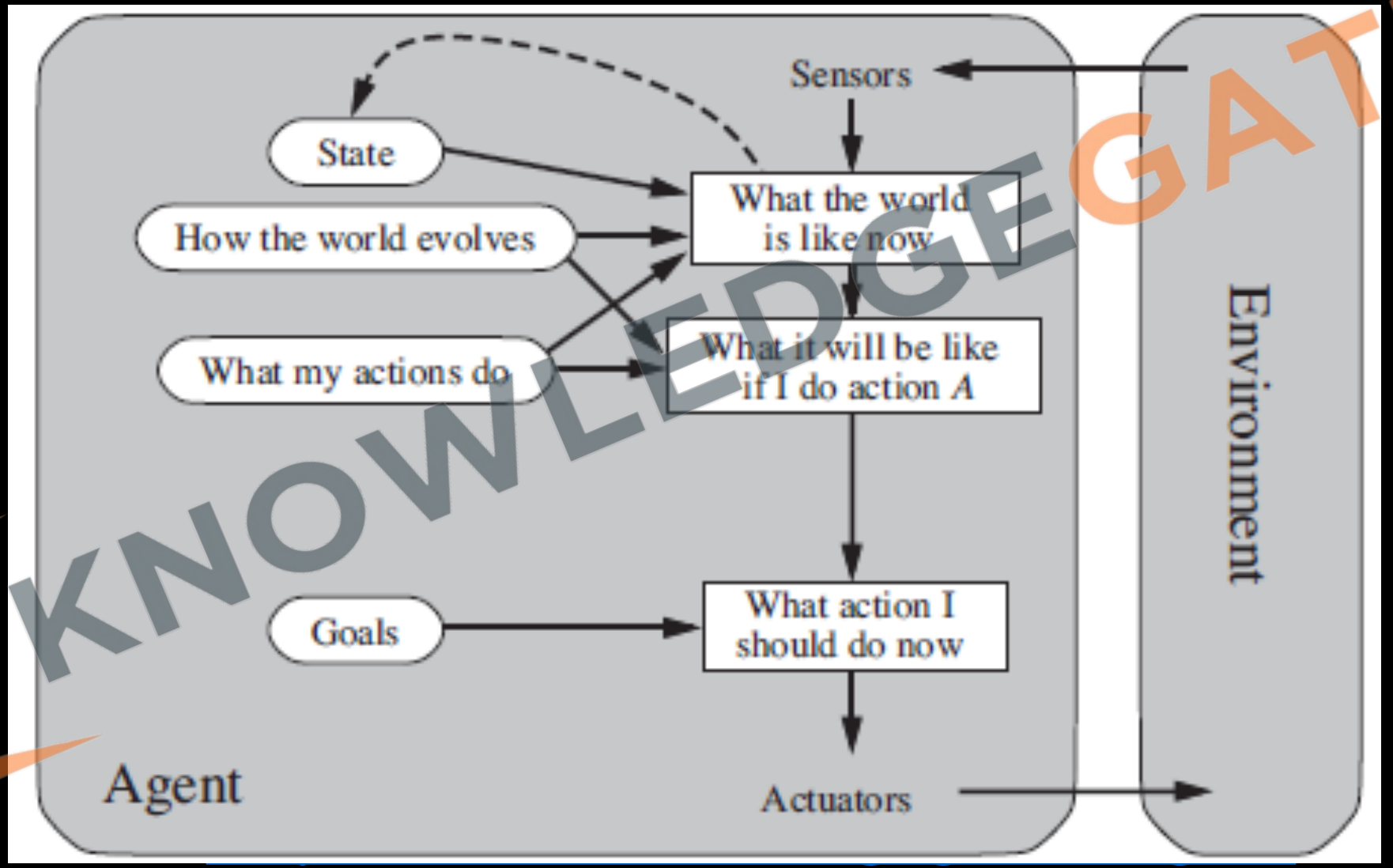


Goal-based agents

- **Goal Information:** It is an extension of model based reflex agent, here we have a goal information that describes desirable situations, guiding their actions.
- **Action Decision:** These agents use their internal model and goal information to decide actions. For example, a taxi at a junction chooses a direction based on its destination.
- **Action Selection:** Goal-based agents may need to consider sequences of actions to achieve the goal. This can be straightforward or complex depending on the goal.



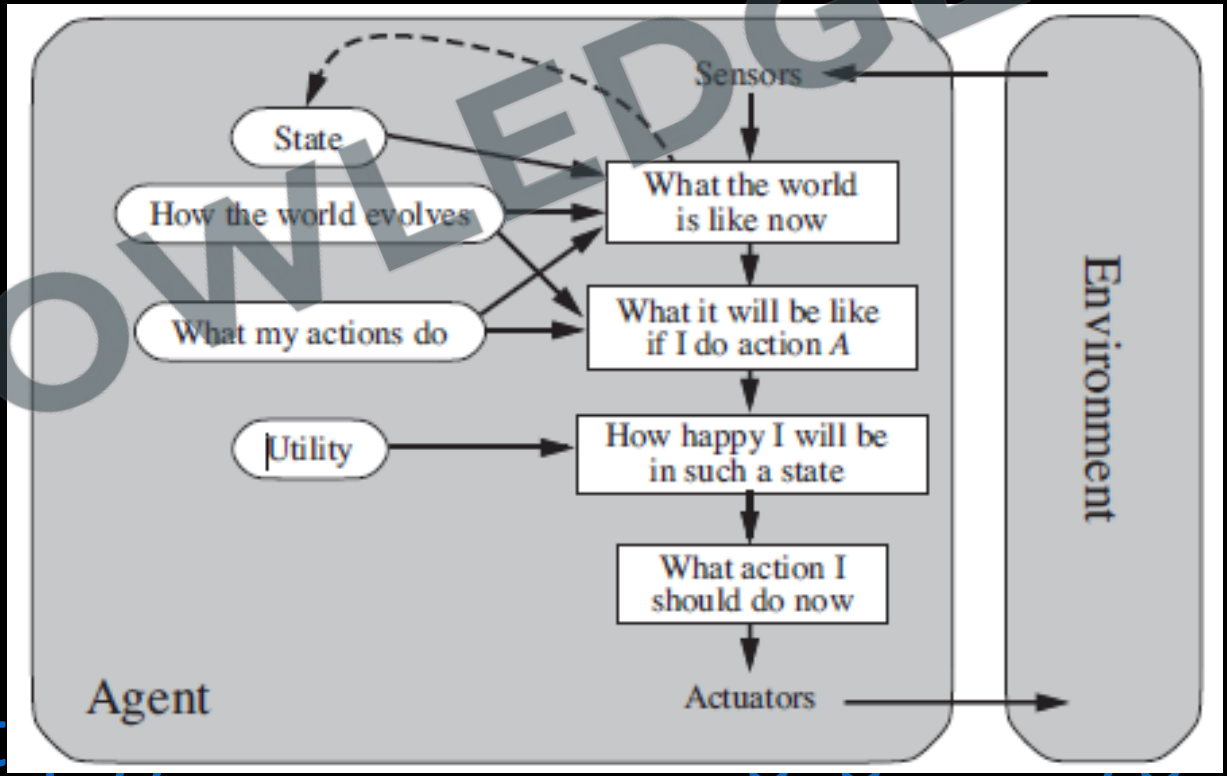
- **Example:** If it starts to rain, a goal-based agent can update its knowledge about how the brakes will operate, adjusting behavior accordingly. This is more flexible than a reflex agent that requires explicit rules for each new situation.



Example: Automated Vacuum Cleaner
Goal:- Clean the entire floor area.
Perception:- Detects location and whether a spot is clean or dirty.
Actions:
- **Move:** Changes position to cover the floor.
- **Clean:** Activates vacuum on dirty spots.
Decision-making:- Chooses actions based on its goal and what it perceives (e.g., cleans dirty spots, moves to new areas if current is clean).
Goal Achievement:- Stops operating once no dirty spots are detected, indicating the floor is clean.

Utility-based agents

- **Performance Measure:** Beyond having a goal, utility-based agents have a performance measure that compares different world states based on how "desirable" they are.
- **Utility:** To sound more scientific, the term "utility" is used to denote how happy or satisfied an agent is with a state.
- **Example:** Many routes can get a taxi to its destination, but some are quicker, safer, or cheaper. Utility allows the agent to distinguish between these routes.



Example: Smart Thermostat

Goal:

- Maintain comfortable home temperature.

Perception:

- Senses current temperature and occupant preferences.

Actions:

- **Heat**, **Cool**, or **Maintain** temperature.

Utility Calculation:

- Balances occupant comfort with energy efficiency.

Decision-making:

- Chooses action to maximize comfort while minimizing energy use.

Utility Achievement:

- Adjusts temperature to keep high utility, considering both comfort and efficiency.

Feature	Simple Reflex Agents	Model-Based Reflex Agents	Goal-Based Agents	Utility-Based Agents
Basis of Operation	Operate on the current percept only.	Use an internal model to keep track of the world state.	Act to achieve defined goals.	Act to maximize a utility function.
Environment Interaction	React to stimuli with conditioned actions.	Update internal state from current state and action history.	Consider future consequences of actions.	Evaluate success based on a utility function.
Flexibility	Very limited; cannot handle new situations well.	More flexible, can handle unseen scenarios to an extent.	Flexible; can adapt actions to achieve goals in changing environments.	Highly flexible; aims for optimal performance.
Learning Ability	None; they do not learn from past actions.	Limited; can improve the model with new experiences.	Can learn better strategies to achieve goals.	Can adjust utility function based on outcomes.
Example	Thermostat controlling a heating system.	A car with anti-lock braking system.	Chess-playing AI considering future moves.	Investment AI maximizing portfolio return.

Key characteristics of a learning agent

- Adaptability
- Memory
- Feedback Sensitivity
- Decision-Making Ability
- Exploration and Exploitation
- Generalization
- Performance Improvement Over Time
- Problem-Solving Skills

- **<u>Chapter-2</u> (**PROBLEM SOLVING METHODS**):** Problem solving Methods - Search Strategies- Uninformed - Informed - Heuristics - Local Search Algorithms and Optimization Problems - Searching with Partial Observations - Constraint Satisfaction Problems - Constraint Propagation - Backtracking Search - Game Playing - Optimal Decisions in Games - Alpha - Beta Pruning - Stochastic Games.

Problem Spaces States Space Representation

The concept of "Problem Spaces States Space Representation" is a method used in artificial intelligence to formulate and solve problems. We use this representation for several reasons:

- **Structure:** It provides a structured way of defining all possible configurations that can occur while solving a problem.
- **Search:** It allows algorithms to search through possible states.
- **Optimization and Decision Making:** Helps in finding the most efficient path or sequence of actions to reach a desired goal state from an initial state.
- **Clarity:** Visually illustrates the complexity of a problem, making it easier to understand and explain.
- **Scalability:** It can be applied to simple as well as very complex problems, scaling from basic puzzles to real-world issues in robotics or planning.

Problem Spaces States Space Representation

Defining a problem as a state space search

- A problem can be defined formally by five components:
  - {Initial State, Action(s), Result(s, a), Goal Test, Path Cost Function}

- **Initial State**:
  - The agent is at the starting point, for example, 'Home'.
- **Action(s)**:
  - The possible actions include choosing a road to travel from one intersection to another.
- **Result(s, a)**:
  - For a chosen action 'a' (road taken), the result would be a new state representing the agent's new location (another intersection).
- **Goal Test**:
  - A function to determine if the agent has reached the destination, 'Work'.
- **Path Cost Function**:
  - A function that adds up the distance (or time) to travel from the initial state to the current state via the chosen paths. The objective is to minimize this cost.

Search Strategies

- Search strategies refer to systematic methods and approaches used to explore and find relevant information or solutions within a given search space or dataset. Parameters for Evaluating Search Technique Performance:
  - **Completeness**: Determines if the search technique guarantees finding a solution if one exists within the search space.
  - **Time and Space Complexity**: Evaluates the efficiency of the search technique in terms of the time and space required to find a solution.
  - **Optimality**: Determines whether the search technique can find the best or optimal solution among all possible solutions.

Uninformed Search(Brute Force Search/Blind Search/Exhaustive Search)

- Uninformed search strategies explore the search space without any specific information or heuristics about the problem. Here we proceed in a systematic way by exploring nodes in some predetermined order or simply by selecting nodes at random.
  - **Advantage**:
    - **Simplicity**: Uninformed search strategies are generally easy to implement and understand.
  - **Disadvantage**:
    - **Inefficiency**: Without additional information, uninformed search strategies may require an extensive search, leading to inefficiency in terms of time and space.
  - **Examples**:
    - **Breadth First Search**
    - **Depth First Search**
    - **Uniform Cost Search**

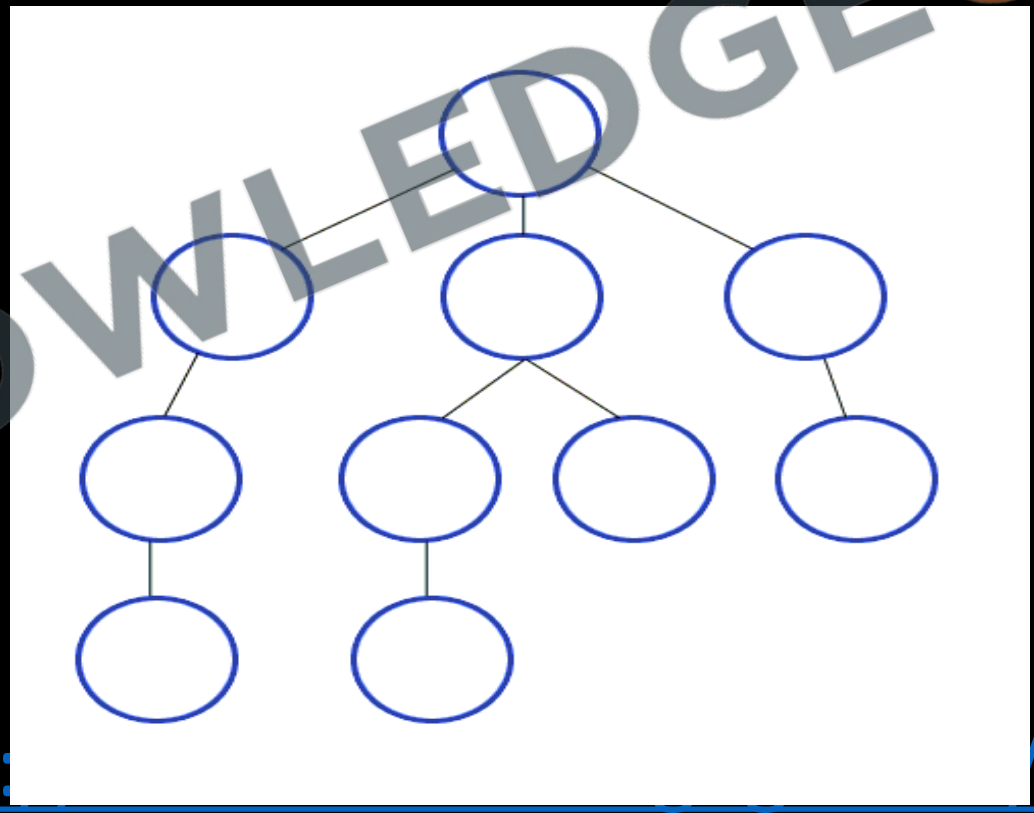
Informed Search(Heuristic Search)

- Informed search strategies utilize domain-specific information or heuristics to guide the search towards more promising paths. Here we have knowledge such as how far we are from the goal, path cost, how to reach to goal node etc. This knowledge helps agents to explore less to the search space and find more efficiently the goal node.
- **Advantage**:
  - **Efficiency**: By leveraging domain knowledge, informed search strategies can make informed decisions and focus the search on more relevant areas, leading to faster convergence to a solution.
- **Disadvantage**:
  - **Heuristic Accuracy**: The effectiveness of informed search strategies heavily relies on the quality and accuracy of the chosen heuristic function. An inaccurate or misleading heuristic can lead to suboptimal or incorrect solutions.
- **Example**:
  - **Hill Climbing**
  - **Best First Search**
  - **A* Algorithm**

Aspect	Uninformed Search	Informed Search
Knowledge	Searches without any prior knowledge.	Uses knowledge, such as heuristics, to guide the search.
Time Efficiency	Generally more time-consuming.	Finds solutions quicker by prioritizing certain paths.
Complexity	Higher complexity due to lack of information, affecting time and space complexity.	Reduced complexity and typically more efficient in both time and space due to informed decisions.
Example Techniques	Depth-First Search (DFS), Breadth-First Search (BFS).	A* Search, Heuristic Depth-First Search, Best-First Search.
Solution Approach	Explores paths blindly.	Explores paths strategically towards the goal.

BFS

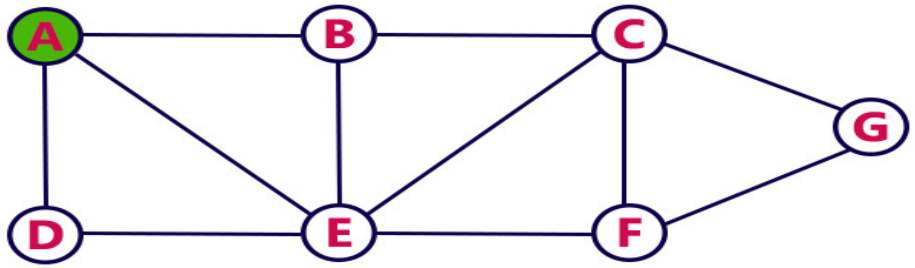
- Breadth-First Search is an uninformed search strategy that explores all neighboring nodes at the current level before moving to the next level starting from the root.
- This is achieved very simply by using a FIFO queue. New nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first.



http: /gate

Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.

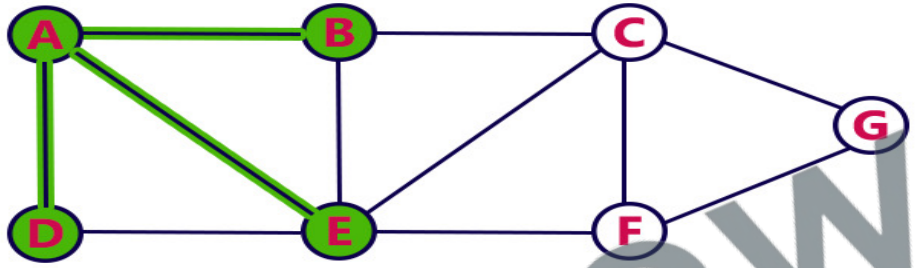


Queue

A						

Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..

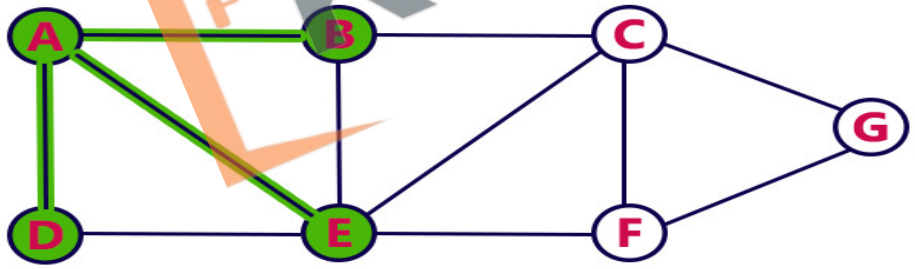


Queue

	D	E	B			

Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.

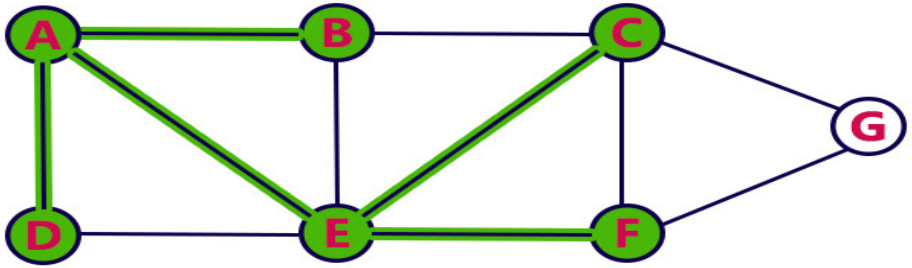


Queue

		E	B			

Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.

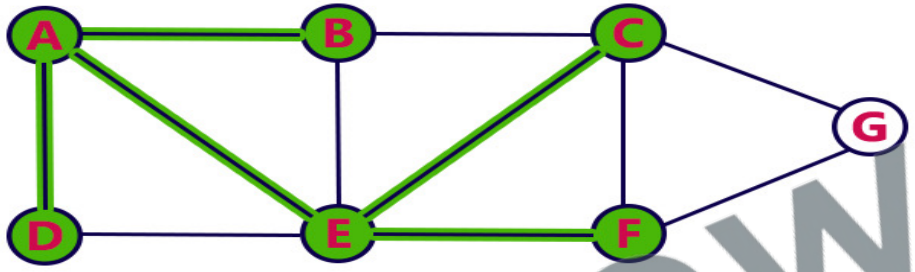


Queue

			B	C	F	

Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.

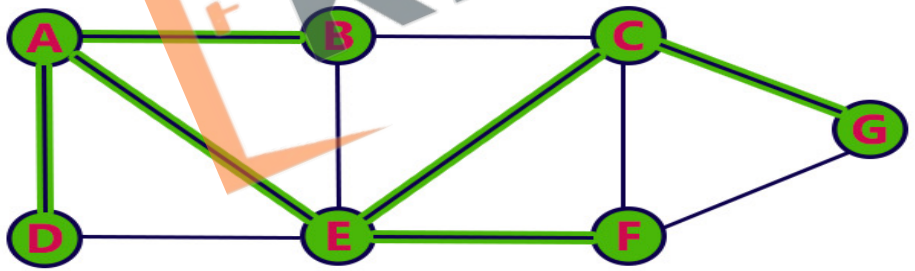


Queue

				C	F	

Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.

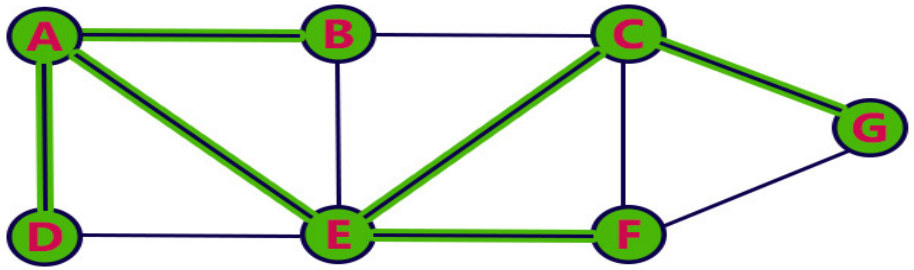


Queue

					F	G

Step 7:

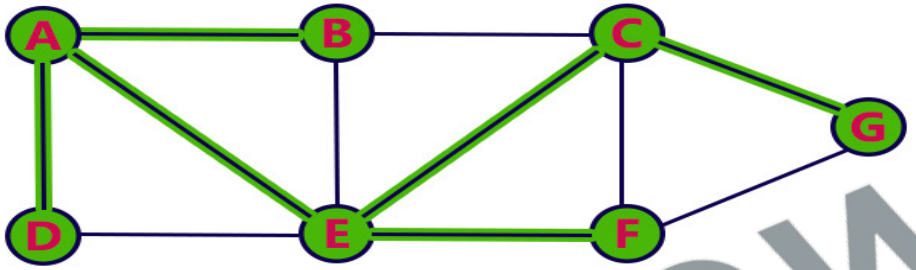
- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.



Queue

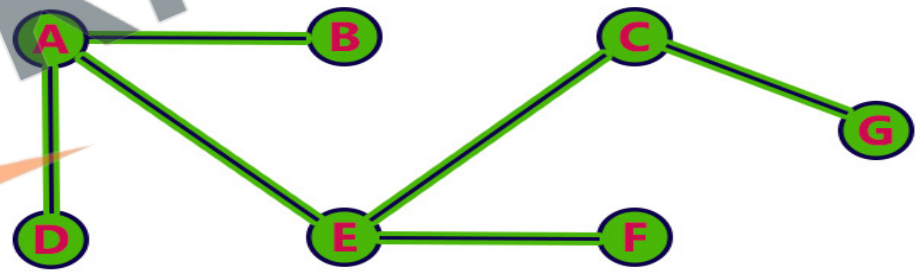
Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



Queue

- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



/gate

Algorithm:

1. Start with the initial state as the root node.
2. Enqueue the root node into a queue.
3. While the queue is not empty, do the following:
    1. Dequeue a node from the front of the queue.
    2. Expand the node and enqueue its unvisited neighboring nodes.
    3. Mark the dequeued node as visited.
4. Repeat steps 3 until the queue is empty or a goal state is found.

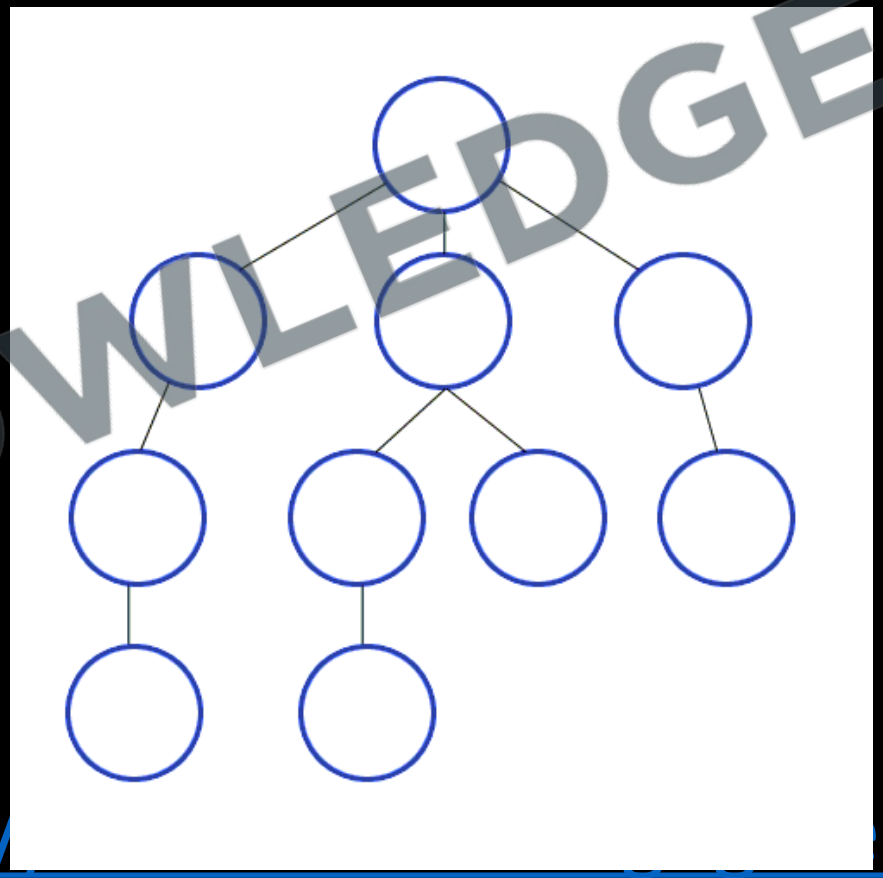
- **Advantages**:
  - Guarantees finding the shortest path if one exists in an unweighted graph.
  - Closer nodes are discovered before moving to farther ones.

- **Disadvantages**:
  - Inefficient in terms of time and space for large search spaces.
  - Not suitable for scenarios where the cost or weight of edges is not uniform.

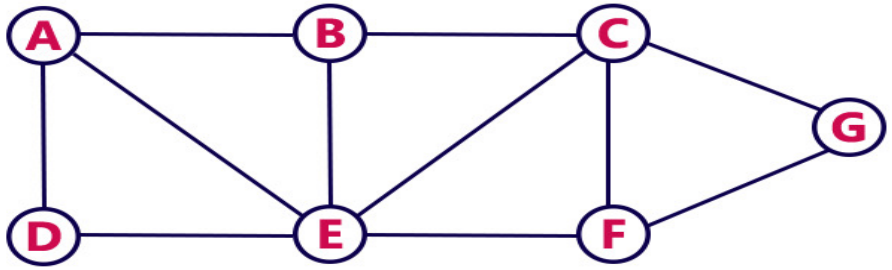
- **Completeness**:
  - Breadth-First Search is complete if the search space is finite or if there is a solution within a finite depth.
- **Optimality**:
  - Breadth-First Search is optimal in terms of finding the shortest path in an unweighted or uniformly weighted graph.
- **Time Complexity**:
  - The time complexity of Breadth-First Search is $O(b^d)$, where b is the branching factor and d is the depth of the shallowest goal node.
- **Space Complexity**:
  - The space complexity of Breadth-First Search is $O(b^d)$, as it stores all the nodes at each level in the search tree in memory.

Depth-first search

- Depth-First Search is an uninformed search strategy that explores as far as possible along each branch before backtracking. It traverses the depth of a search tree or graph before exploring the neighboring nodes.

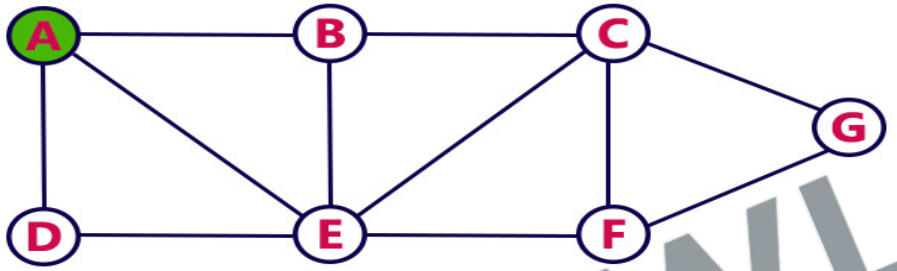


Consider the following example graph to perform DFS traversal



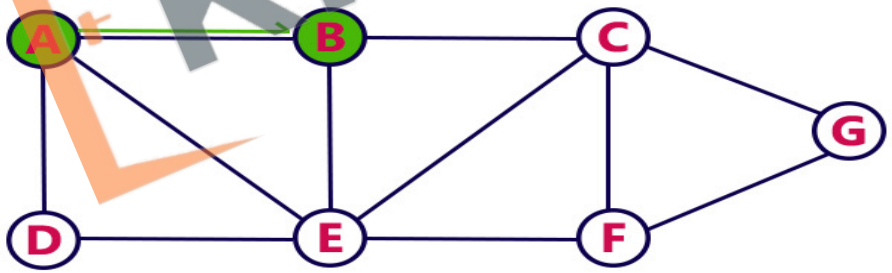
Step 1:

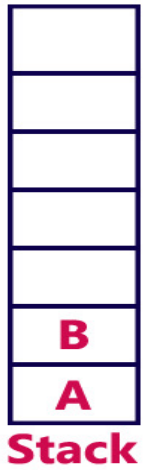
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.



Step 2:

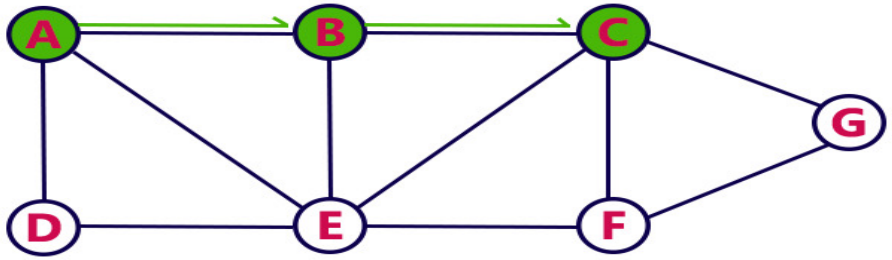
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

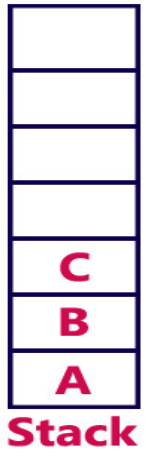




Step 3:

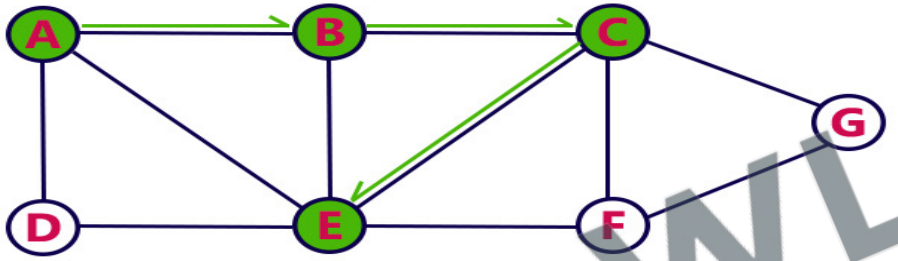
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.





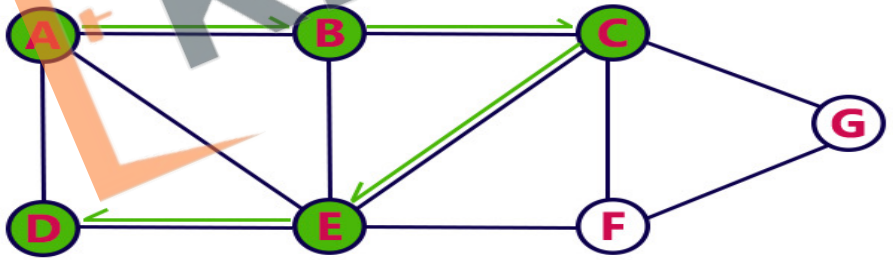
Step 4:

- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack



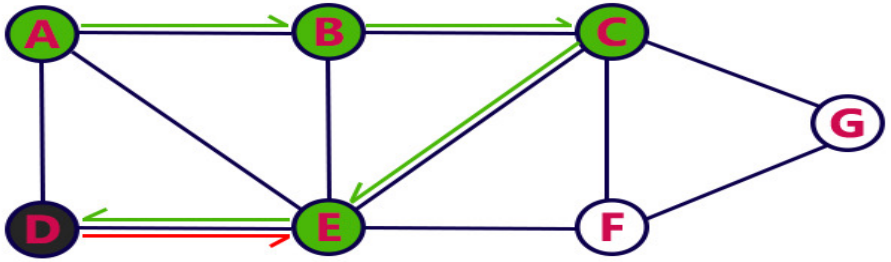
Step 5:

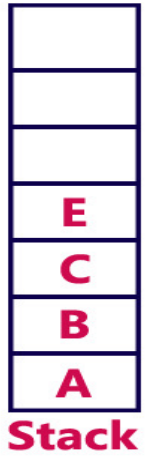
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack



Step 6:

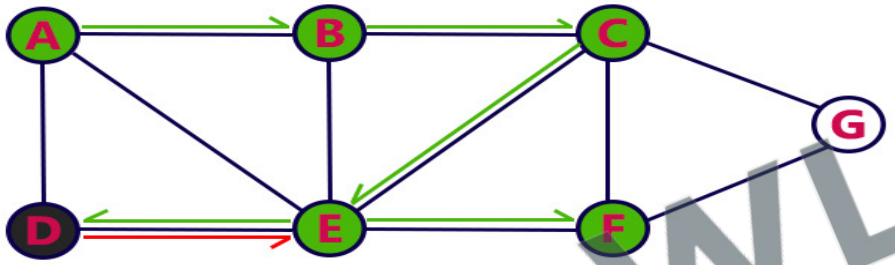
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.





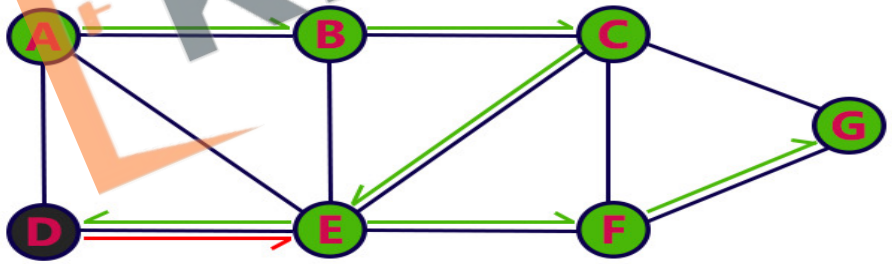
Step 7:

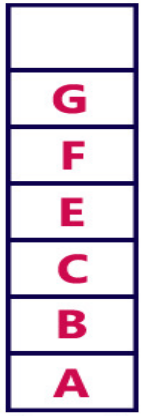
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.



Step 8:

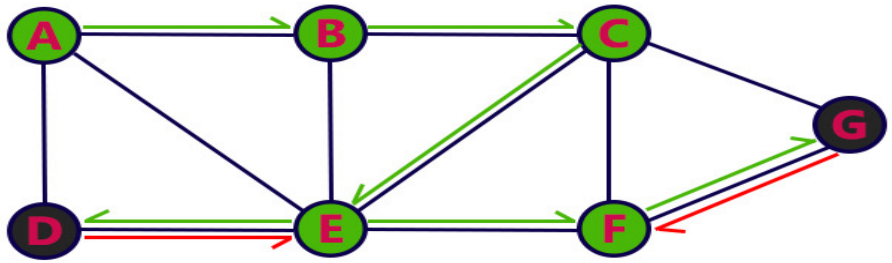
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.

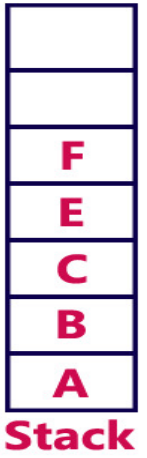




Step 9:

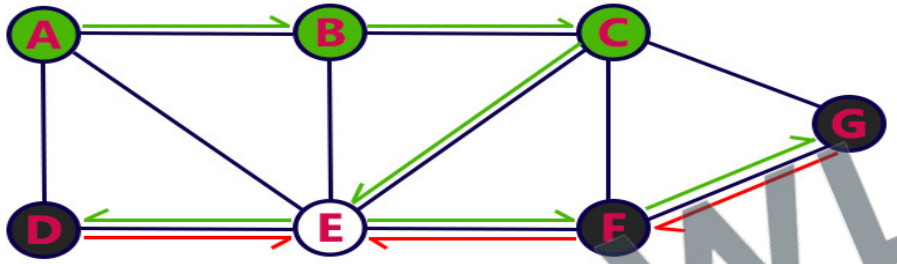
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.





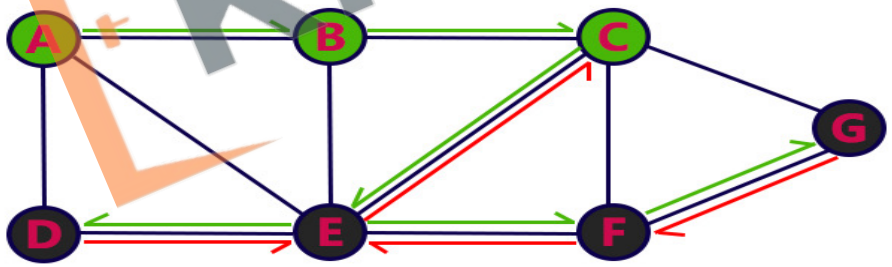
Step 10:

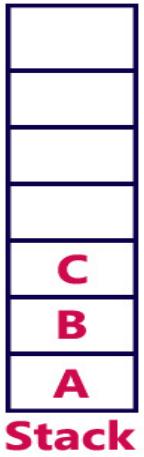
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.



Step 11:

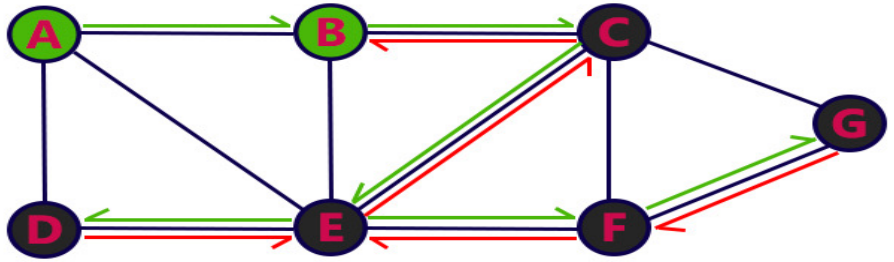
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

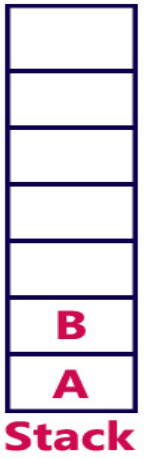




Step 12:

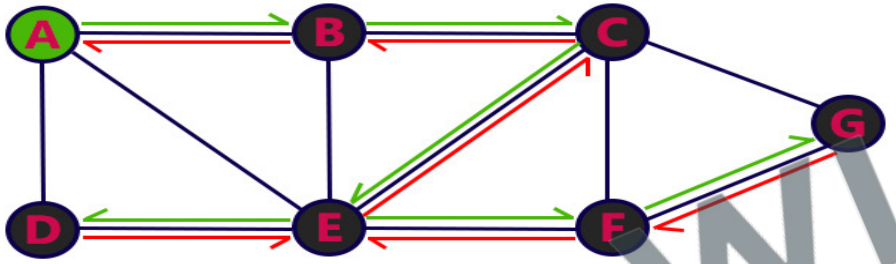
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.





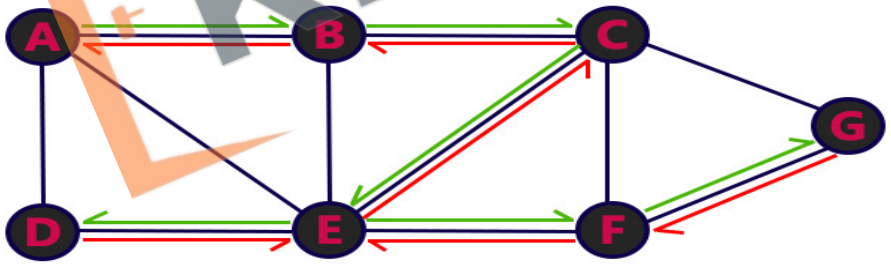
Step 13:

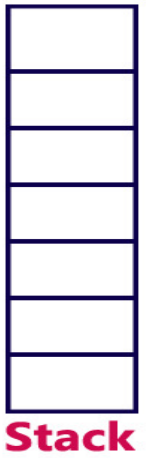
- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.



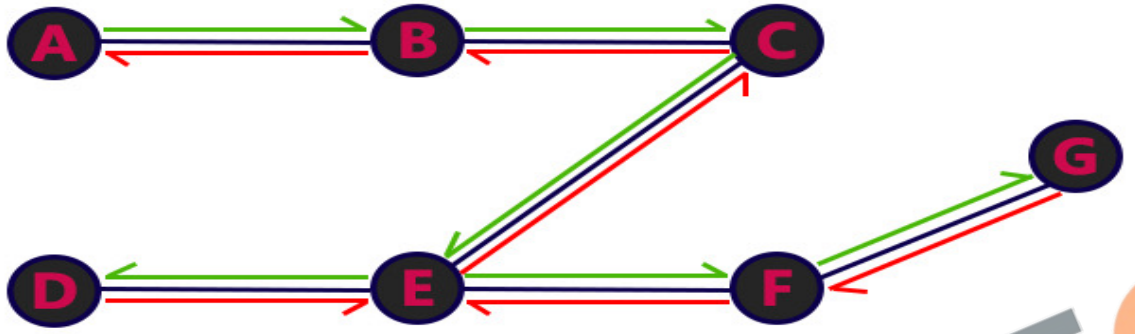
Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.





- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



Algorithm:

1. Start with the initial state as the root node.
2. Explore a node and recursively explore its unvisited neighboring nodes.
3. Backtrack if all neighboring nodes have been visited or there are no unvisited nodes left.
4. Repeat steps 2 and 3 until a goal state is found or all nodes have been visited.

Advantages:

- **Memory Efficiency**: Depth-First Search stores fewer nodes in memory compared to breadth-first search, as it explores deeper paths first.

Disadvantages:

- **Completeness Not Guaranteed**: DFS may get trapped in infinite loops or infinite branches without reaching the goal state if there are cycles in the search space.
- Non-Optimal Solution: DFS does not guarantee finding the optimal solution, as it may find a solution at a greater depth before finding a shorter path.

Completeness:

- DFS is not complete if the search space is infinite or contains cycles. If depth is finite than it is complete.

Optimality:

- DFS does not guarantee finding the optimal solution, as it may find a solution at a greater depth before finding a shorter path.

Time Complexity:

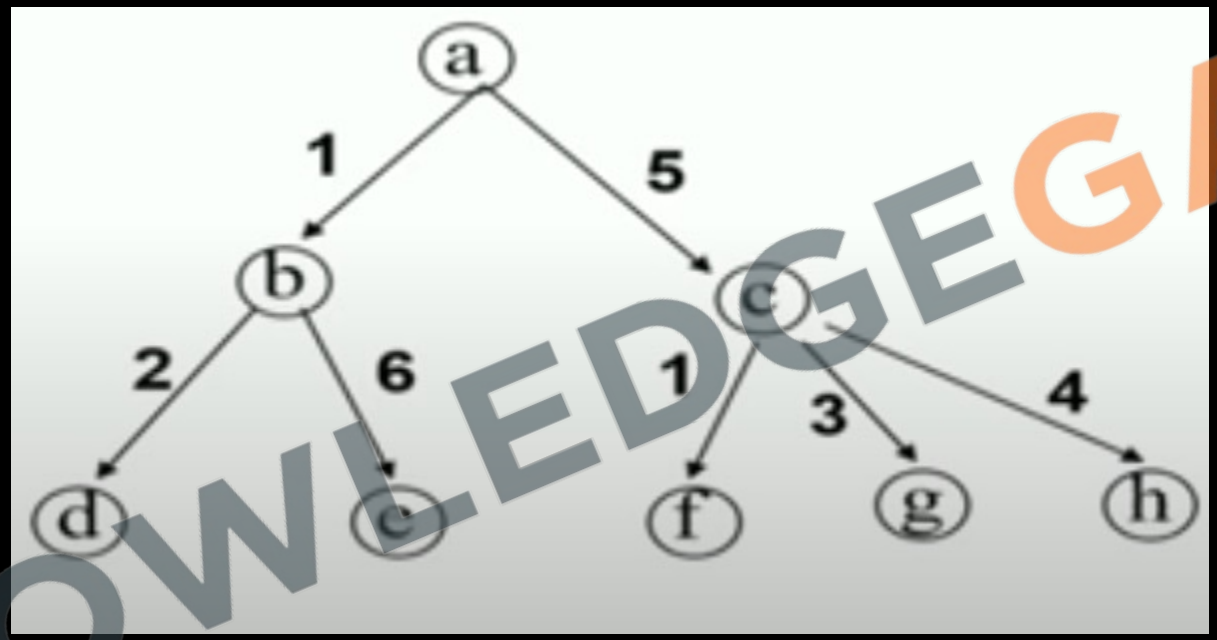
- The time complexity of DFS can vary depending on the search space structure. In the worst case, it can be $O(b^m)$, where b is the branching factor and m is the maximum depth of the search tree.

Space Complexity:

- The space complexity of DFS is O(bm), where b is the branching factor and m is the maximum depth of the search tree. It stores nodes along the current path in memory.

Depth-limited search

- It is an extension of Breadth-First Search (BFS) that takes into account the cost of reaching each node to find the lowest-cost path to the goal. **Example:** Consider the graph below.



- **Based on the cost we will expand the graph in order: a→ b → d → c → f → e → g → h**

Algorithm:

1. Start with the initial state as the root node.
2. Maintain a priority queue or a priority-based data structure to store nodes based on their path cost.
3. Enqueue the root node with a path cost of zero.
4. While the priority queue is not empty, do the following:
    1. Dequeue the node with the lowest path cost from the priority queue.
    2. If the dequeued node is the goal state, terminate the search and return the solution.
    3. Otherwise, expand the node and enqueue its unvisited neighboring nodes with their updated path costs.
5. Repeat steps 4 until the goal state is found or the priority queue is empty.

Advantages:

- **Optimal Solution**: Uniform Cost Search guarantees finding the lowest-cost path to the goal.

Disadvantages:

- **Time and Space Complexity**: The time and space complexity can be high, especially in large search spaces or when there are many nodes with different path costs.

Completeness:

- Uniform Cost Search is complete if the search space is finite, and all path costs are greater than some threshold.

Optimality:

- Uniform Cost Search is optimal as it guarantees finding the lowest-cost path to the goal.

Time Complexity:

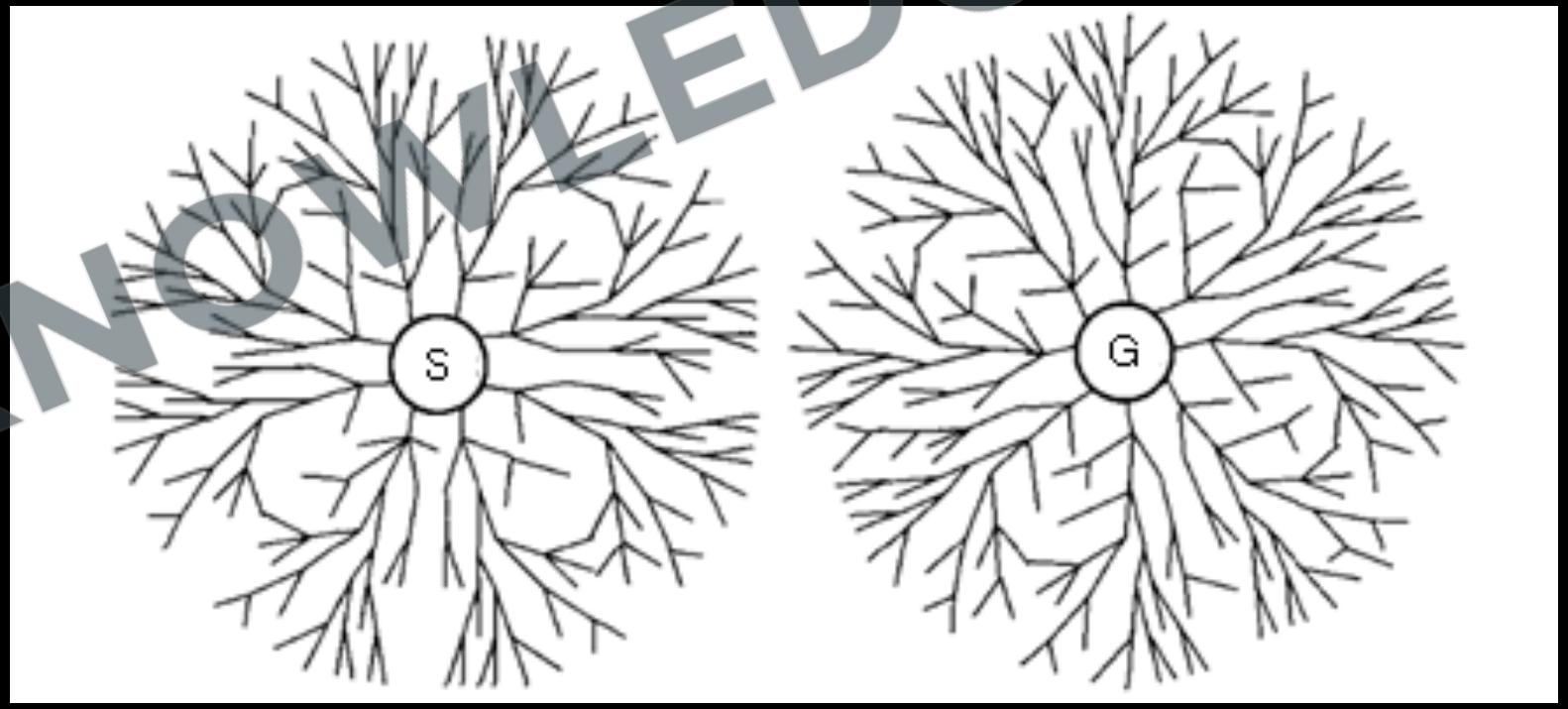
- The time complexity of Uniform Cost Search depends on the number of nodes and the cost of the lowest-cost path to the goal. In the worst case, it can be exponential, i.e., $O(b^d)$, where b is the branching factor and d is the depth of the shallowest goal node.

Space Complexity:

- The space complexity of Uniform Cost Search can also be exponential in the worst case, i.e., $O(b^d)$, as it may need to store all the nodes along the lowest-cost path in memory.

Bidirectional Search

- Bidirectional Search is a search algorithm that simultaneously performs two separate searches, one forward from the initial state and one backward from the goal state. It aims to meet in the middle by searching for a common node reached from both directions.
- The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d. Bidirectional search is implemented by replacing the goal test with a check to see whether the frontiers of the two searches intersect; if they do, a solution has been found.



Advantages:

- **Faster Exploration**: Bidirectional Search can be faster than traditional searches by exploring the search space simultaneously from both ends, potentially reducing the search effort.
- **Reduces Effective Branching Factor**: As the search progresses from both directions, the effective branching factor is reduced, leading to a more efficient search.

Disadvantages:

- **Increased Memory Requirement**: Bidirectional Search requires storing visited nodes from both directions, leading to increased memory consumption compared to unidirectional searches.
- **Additional Overhead**: The coordination and synchronization between the two searches introduce additional overhead in terms of implementation complexity.

Completeness:

- Bidirectional Search is complete if both the forward and backward searches are complete in a finite search space.

Optimality:

- Bidirectional Search is optimal if both the forward and backward searches are optimal.

Time Complexity:

- The time complexity of Bidirectional Search depends on the branching factor, the depth of the shallowest goal state, and the meeting point of the two searches. In the best case, it can be $O(b^{d/2})$, where b is the branching factor and d is the depth of the goal state.

Space Complexity:

- The space complexity of Bidirectional Search depends on the memory required to store visited nodes from both directions. In the best case, it can be $O(b^{d/2})$, where b is the branching factor and d is the depth of the goal state.

Comparison of Uniformed Searches

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Problems in Uninformed Search

1. **Blind Exploration**: Uninformed search strategies, such as Breadth-First Search or Depth-First Search, lack domain-specific knowledge or heuristics. They explore the search space without any information about the problem structure or the goal state

2. **Inefficient in Complex Spaces**: Uninformed search strategies can be inefficient in large or complex search spaces, as they may require extensive exploration of irrelevant or unpromising paths.

INFORMED (HEURISTIC) SEARCH STRATEGIES

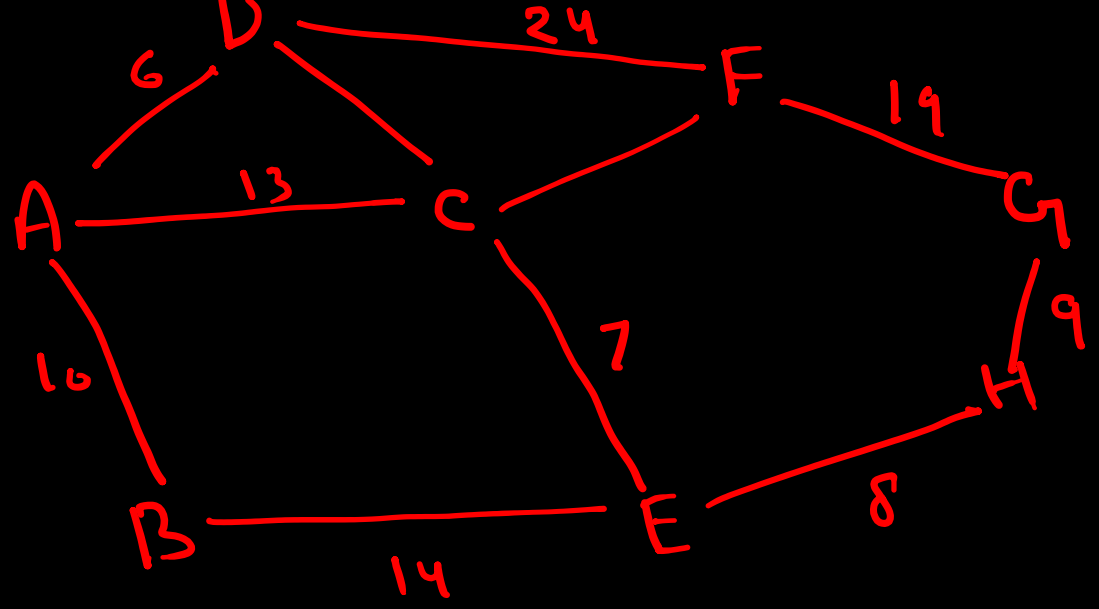
- Heuristic search strategies are like smart search methods that use special knowledge or hints to find solutions more efficiently. It's similar to when we use our intuition or common sense to solve a problem.

Best-first Search

- Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. It's called "best-first" because it greedily chooses the node that seems most promising according to the heuristic.

- Best-first search takes the advantage of both BFS and DFS. The evaluation function is construed as a cost estimate, so the node with the *lowest* evaluation is expanded first.

- ***h(n) = estimated cost of the cheapest path from the state at node n to a goal state.***



node	h(n)
A =	39
B =	31
C =	24
D =	34
E =	18
F =	16
H =	9
G =	0

Algorithm:

1. Define a ‘priority queue’ of nodes, where each node has a priority equal to its heuristic value ‘h(n)’.
2. Start with the ‘root node’
3. Dequeue the node with the highest priority.
4. If the node contains the goal state, return it and exit. Else, enqueue all the node's successors.
5. If the queue is empty, return failure. Else, go to step 3.

Advantages of Best-First Search

- It can find a solution without exploring much of the state space.
- It uses less memory than other informed search methods like A* as it does not store all the generated nodes.

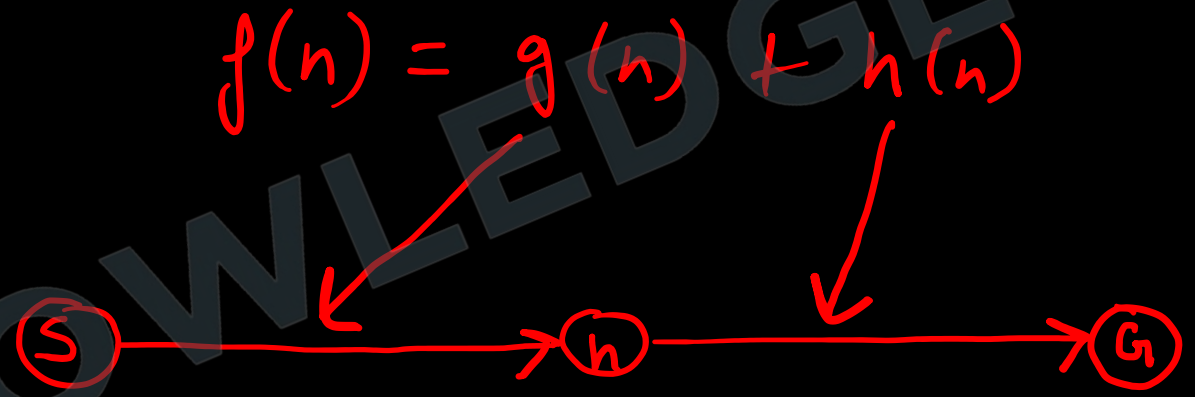
Disadvantages of Best-First Search

- It is not complete. In some cases, it may get stuck in an infinite loop.
- It is not optimal. It does not guarantee the shortest possible path will be found.
- It heavily depends on the accuracy of the heuristic.

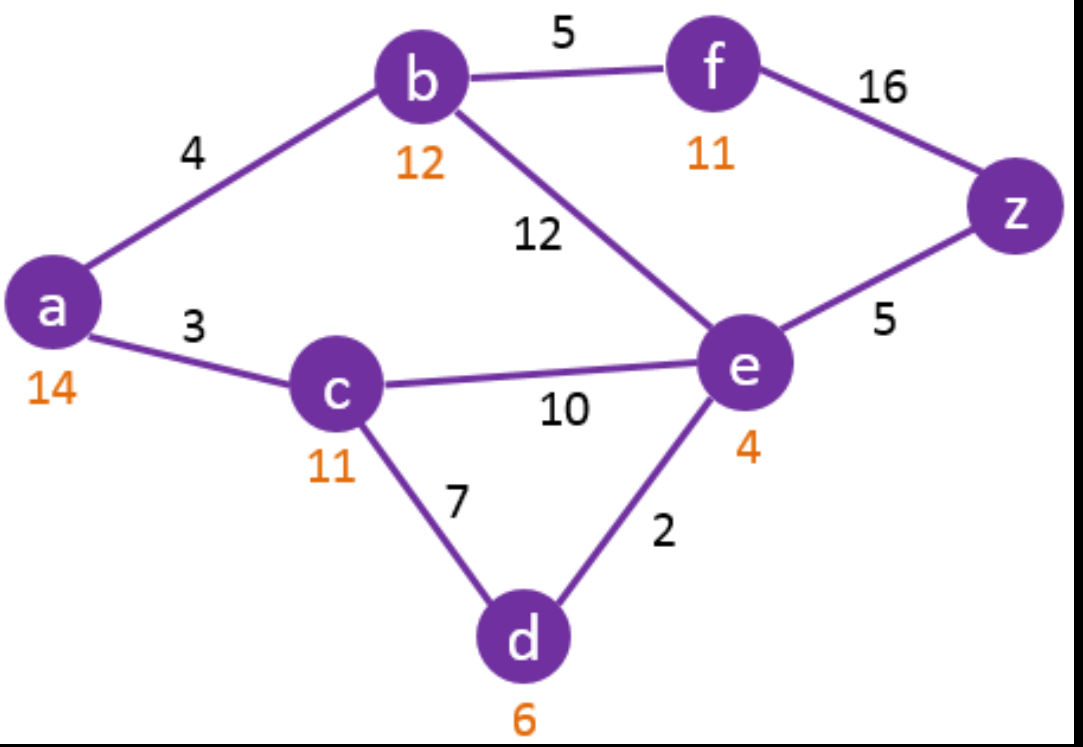
- **Time and Space Complexity**
  - The time and space complexity of Best-First Search are both $O(b^d)$, where b is the branching factor (number of successors per state), and d is the depth of the shallowest solution.
  - These complexities can be very high in problems with a large number of states and actions. In the worst case, the algorithm will need to visit all nodes, and since each node is stored, the space complexity is also proportional to the number of nodes.

A* search: Minimizing the total estimated solution cost

- A* Search is an informed search algorithm that combines the advantages of both Uniform Cost Search and Greedy Best-First Search. It evaluates a node based on a combination of the cost of the path from the start node to that node and an estimated heuristic function that estimates the cost to reach the goal form the current node.



- A∗ search evaluates nodes by combining g(n), the cost to reach the node, and h(n), the estimated cost to get from the node to the goal: **f(n) = g(n) + h(n)**



Algorithm:

1. Start with the initial state as the root node.
2. Create an evaluation function that combines the cost of the path and a heuristic estimate.
3. Initialize an empty priority queue or priority-based data structure.
4. Enqueue the initial state into the priority queue based on the evaluation function.
5. While the priority queue is not empty, do the following:
    1. Dequeue the node with the highest priority from the priority queue.
    2. If the dequeued node is the goal state, terminate the search and return the solution.
    3. Otherwise, expand the node and enqueue its unvisited neighboring nodes into the priority queue based on the evaluation function.
6. Repeat steps 5 until a solution is found or the priority queue is empty.

Completeness:

- A* Search is complete if the search space is finite and the heuristic function is admissible (never overestimates the actual cost).

Optimality:

- A* Search is optimal if the heuristic function is admissible and consistent (also known as monotonic).

Time Complexity:

- The time complexity of A* Search depends on the heuristic function, the branching factor, and the structure of the search space. In the worst case, it can be exponential.

Space Complexity:

- The space complexity of A* Search depends on the size of the priority queue and the number of nodes stored in memory. In the worst case, it can be exponential.

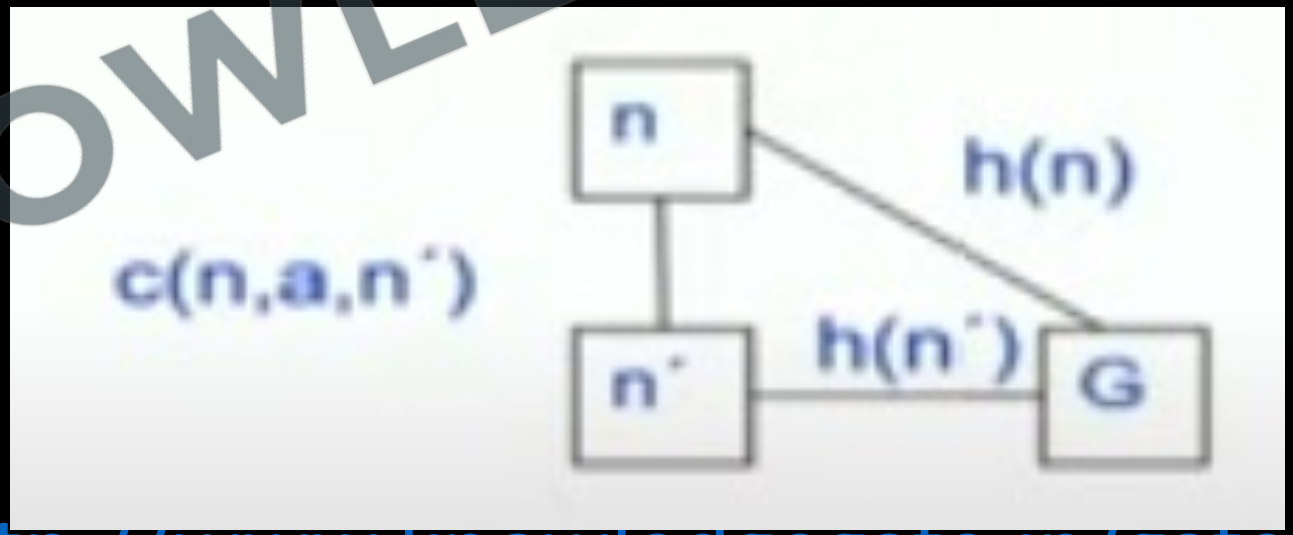
Conditions for optimality: Admissibility and consistency

- The first condition we require for optimality is that h(n) be an **admissible heuristic.**
- An admissible heuristic is one that ***never overestimates*** the cost to reach the goal. Admissible heuristics are by nature **optimistic** because they think the cost of solving the problem is less than it actually is.
- For example, Straight-line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot be an overestimate.

- A second, slightly stronger condition called **consistency** (or **monotonicity**) is required only for applications of A∗ to graph search.

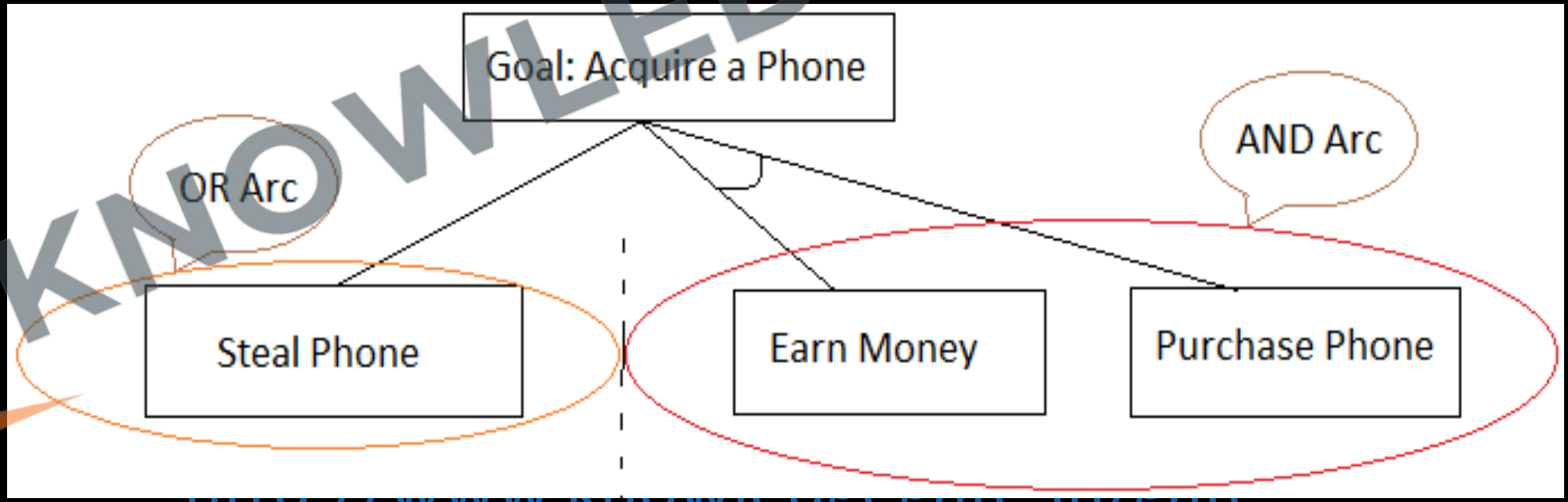
- A heuristic h(n) is **consistent** if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

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

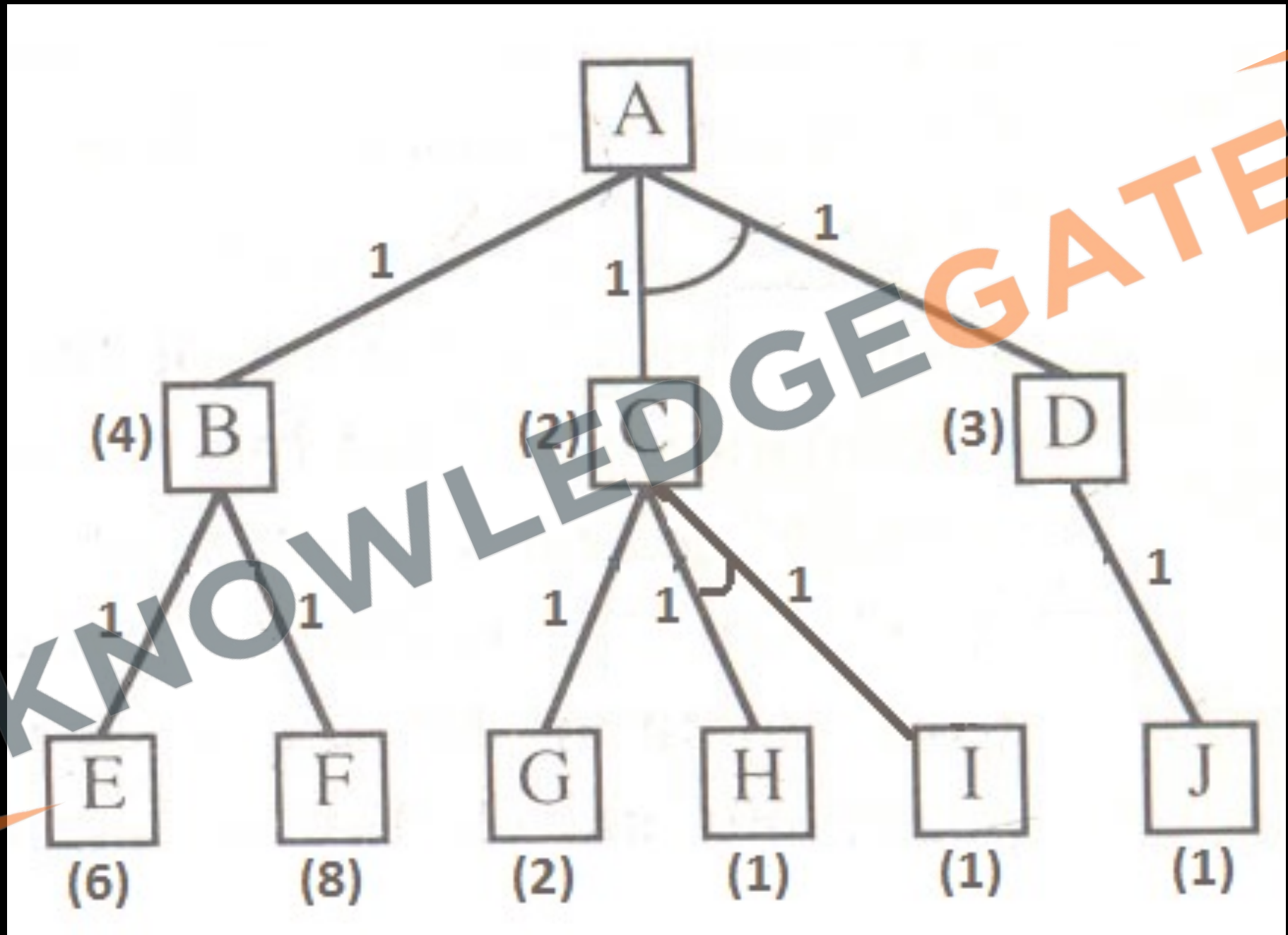


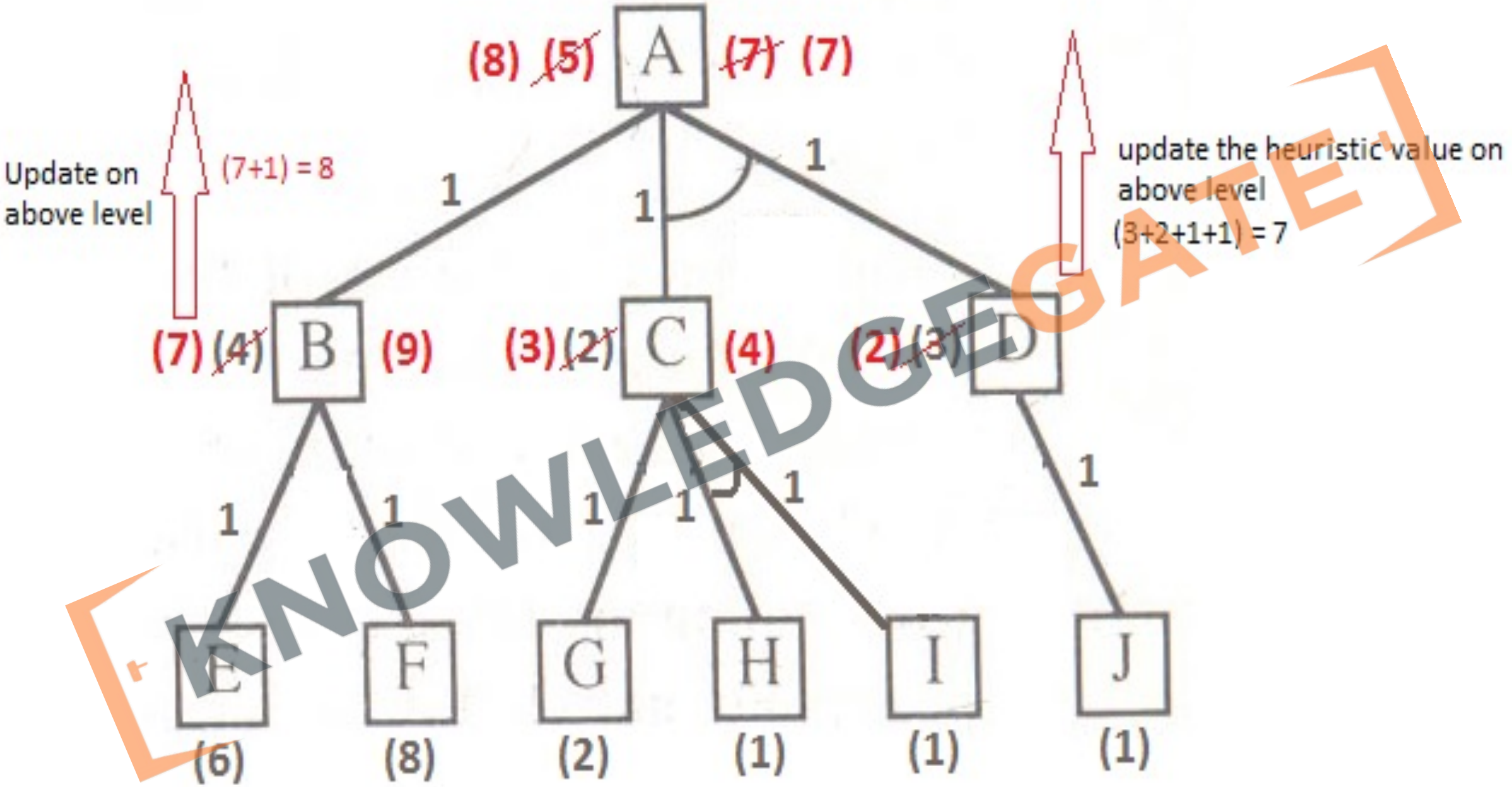
Problem with A* Algorithm

- So far we have considered search strategies for OR graphs through which we want to find a single path to a goal.
- The AND-OR GRAPH (or tree) is useful for representing the solution of problems that can solved by decomposing them into a set of smaller problems, all of which must then be solved. This decomposition, or reduction, generates arcs that we call AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.



AO Search Algorithm*





The pseudocode of AO* is as follows:

1. Initialize the graph with a single node (the start node).
2. While the solution graph contains non-terminal nodes (nodes that have successors not in the graph):
    1. Choose a non-terminal node for expansion based on a given strategy.
    2. Expand the node (add successors to the graph and update the costs of nodes).
3. The process continues until the start node is labeled as a terminal node.

Advantages of AO Algorithm*

- It can efficiently solve problems with multiple paths due to its use of heuristics.
- It is optimal when the heuristic function is admissible (never overestimates the true cost).

Disadvantages of AO Algorithm*

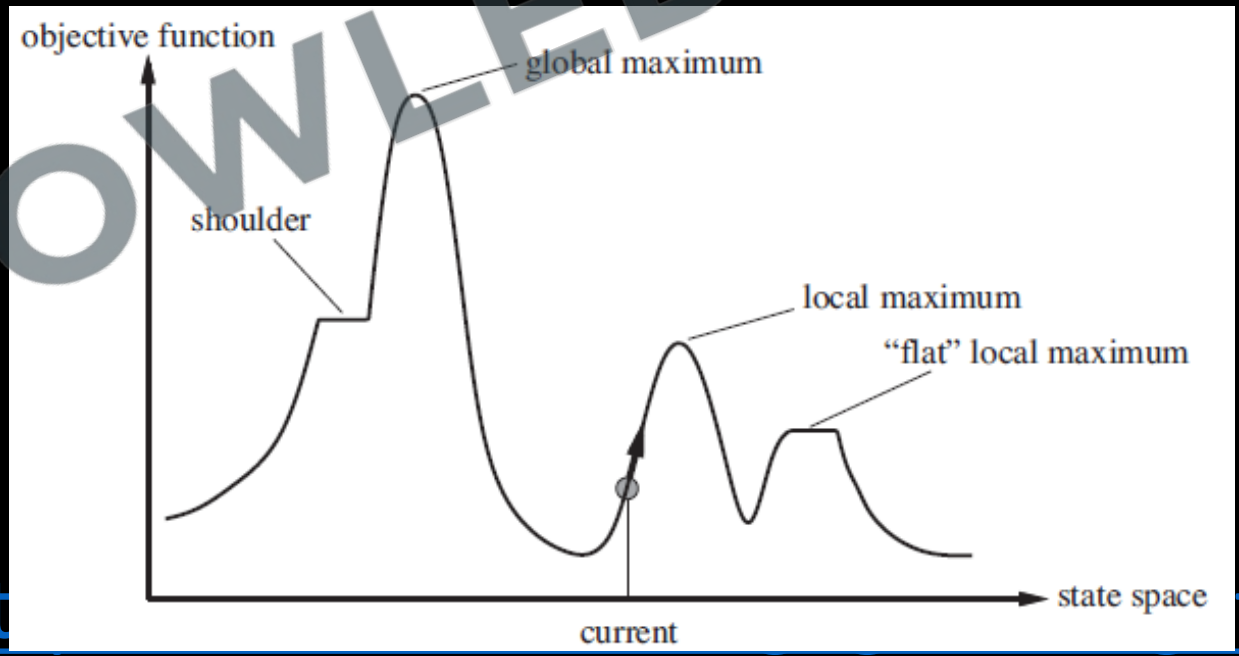
- It can consume a large amount of memory, similar to the A* algorithm.
- The performance of AO* is heavily dependent on the accuracy of the heuristic function. If the heuristic function is not well-chosen, AO* could perform poorly.

- **Completeness and Optimality**
  - AO* is complete, meaning it is guaranteed to find a solution if one exists. It is also optimal, meaning it will find the best solution, provided that the heuristic function is admissible (never overestimates the true cost) and consistent (satisfies the triangle inequality).

- **Time and Space Complexity**
  - The time and space complexity of the AO* algorithm is highly dependent on the problem at hand, particularly the size of the state space, the number of goals, and the quality of the heuristic function. In the worst case, the algorithm will need to visit all nodes, and since each node is stored, the space complexity is also proportional to the number of nodes.

Hill Climbing Algorithm

- It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making a change, If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.
- Hill climbing is sometimes called **greedy local search with no backtracking** because it grabs a good neighbor state without thinking ahead about where to go next.

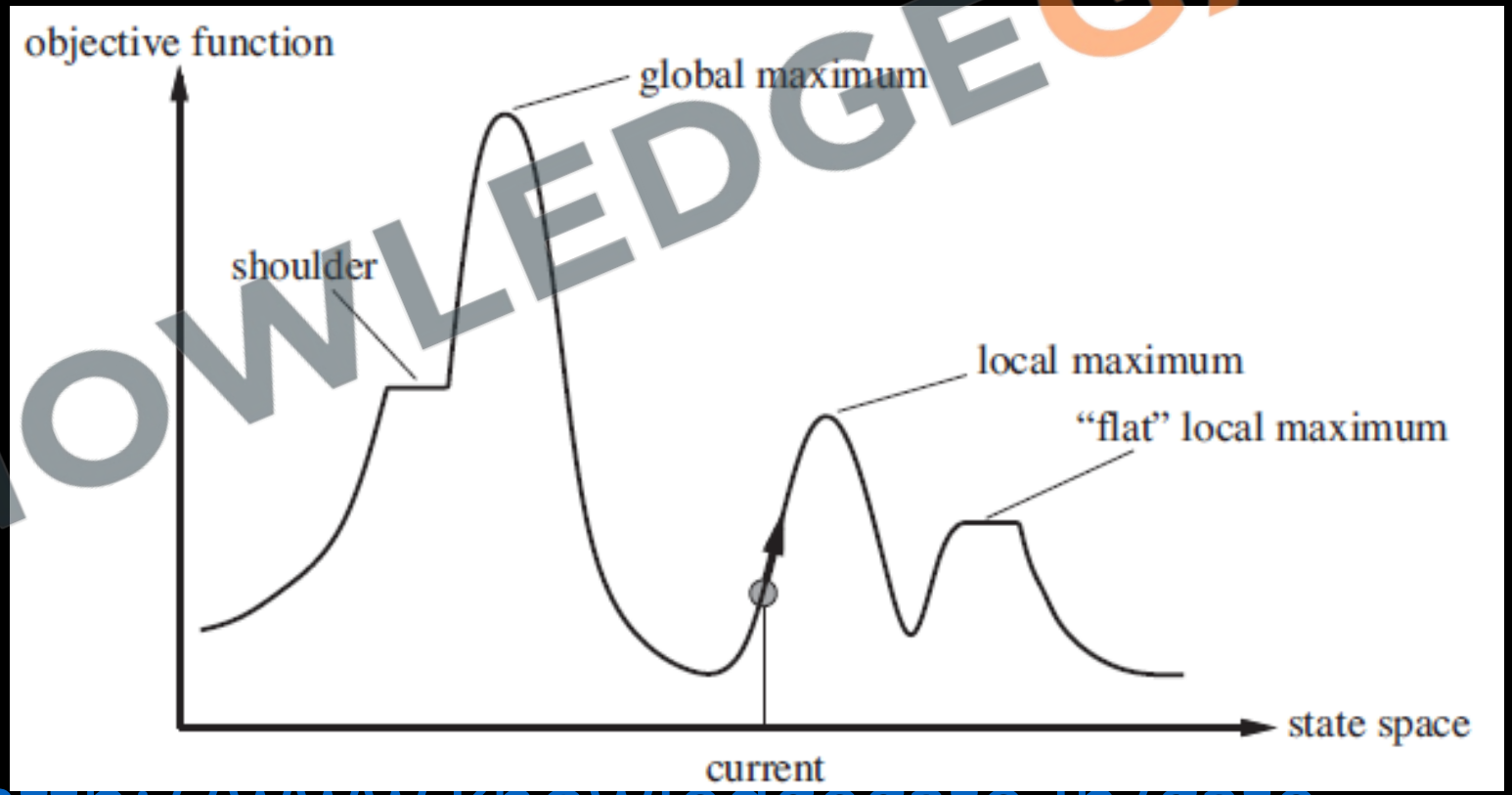


Algorithm for Simple Hill climbing

- **Start**: Begin at a random point on the problem landscape.
- **Evaluate**: Assess the current position.
- **Look Around**: Check neighboring points and evaluate them.
- **Can Improve**: Determine if any neighbors are better (higher value) than the current position.
- **Move**: If a better neighbor exists, move to that point.
- **Repeat**: Continue the process from the new point.
- **No Improvement**: If no better neighbors are found, you've hit a peak.
- **End**: The process stops when no further improvement can be made.

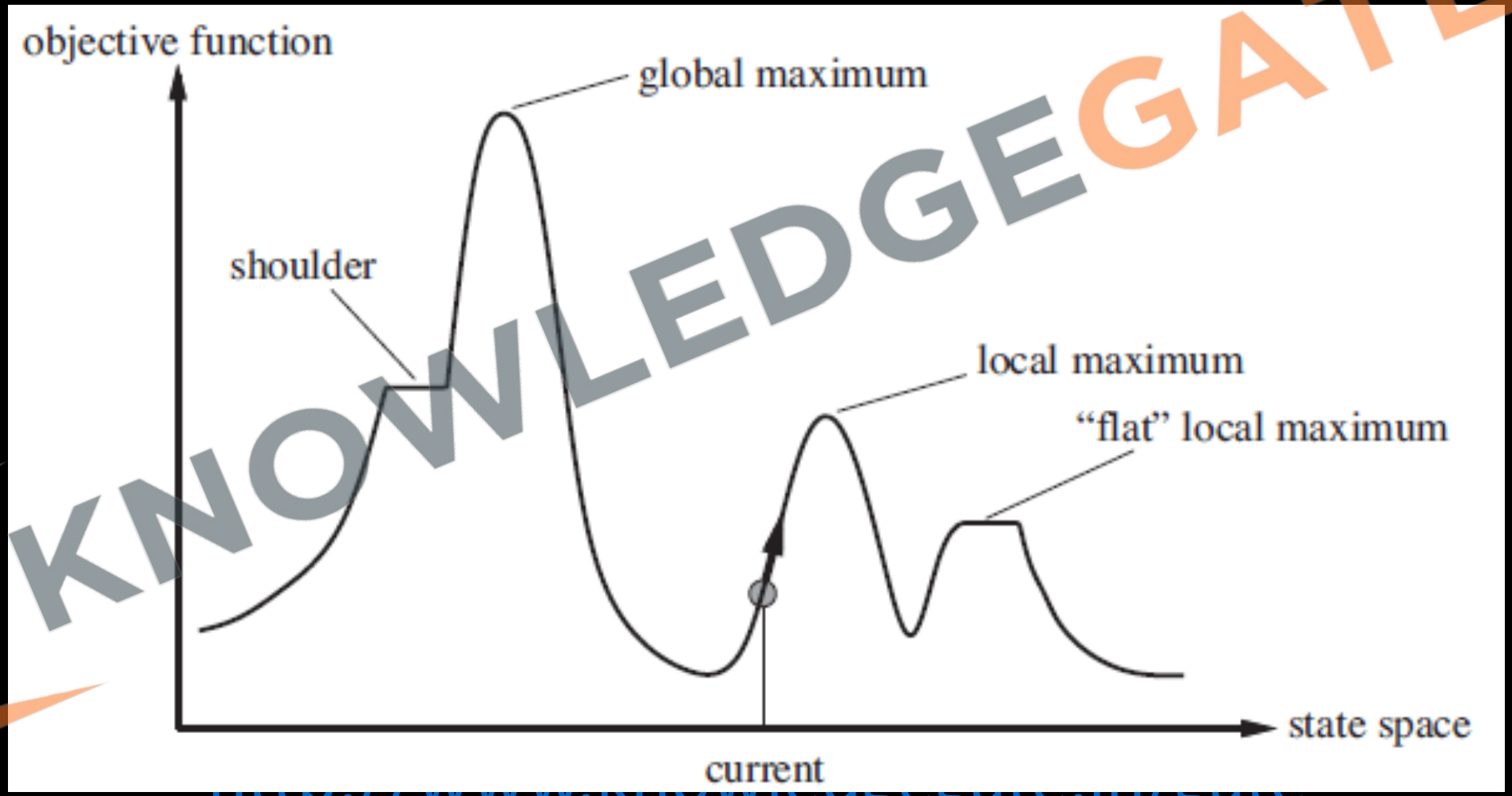
Problems with Hill Climbing

- **Local maxima**: a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.



- **Ridges:** Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

- **Plateau:** A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau. In each case, the algorithm reaches a point at which no progress is being made.



Advantages of Hill Climbing:

- It's simple to understand and easy to implement.
- It requires less computational power compared to other search algorithms.
- If the heuristic is well chosen, it can find a solution in a reasonable time.

Disadvantages of Hill Climbing:

- It's not guaranteed to find the optimal solution.
- It's highly sensitive to the initial state and can get stuck in local optima.
- It does not maintain a search history, which can cause the algorithm to cycle or loop.
- It can't deal effectively with flat regions of the search space (plateau) or regions that form a ridge.

Random-restart hill climbing

- To avoid local optima, you can perform multiple runs of the algorithm from different random initial states. This is called random-restart hill climbing and increases the chance of finding a global optimum.
- **Tabu Search:** To prevent the algorithm from getting stuck in a loop or revisiting states, a list of previously visited states can be maintained, which are then avoided in future steps.
- **Local Beam Search:** To tackle plateau and local optima, this variation of hill climbing keeps track of k states rather than just one. It begins with k randomly generated states. At each step, all the successors of all k states are generated, and if any one is a goal, it halts. Else, it selects the best k successors from the complete list and repeats.

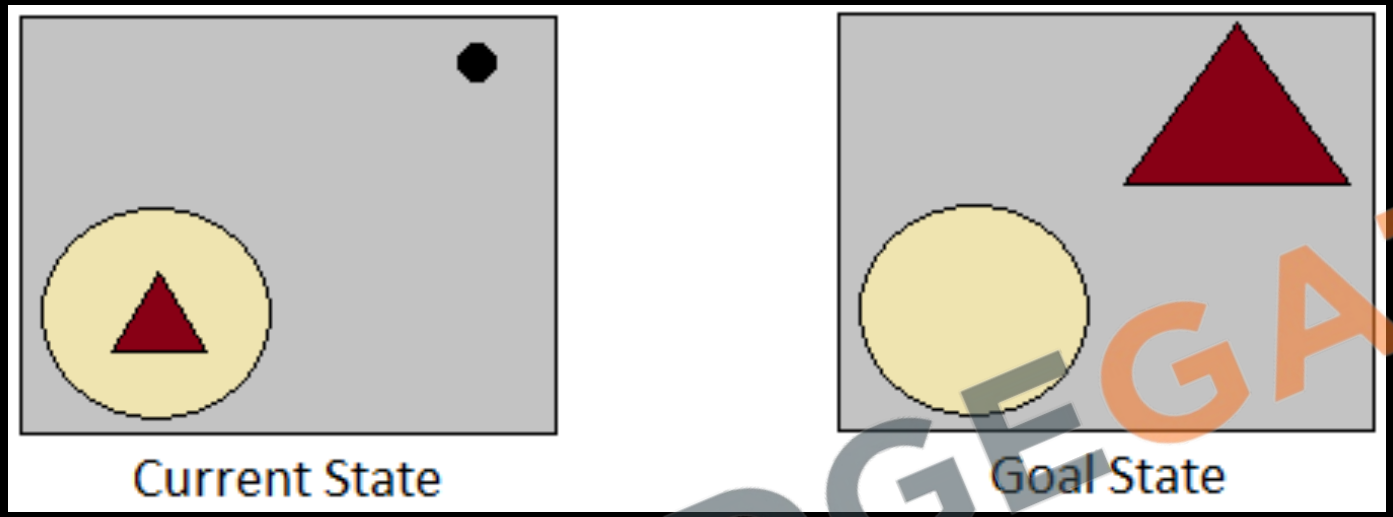
Means-Ends Analysis

- **What it is**: A strategic approach combining forward and backward reasoning to tackle complex and large problems in AI.
- **Why use it**: It helps in focusing the problem-solving effort on significant discrepancies between the current state and the goal, reducing unnecessary exploration.

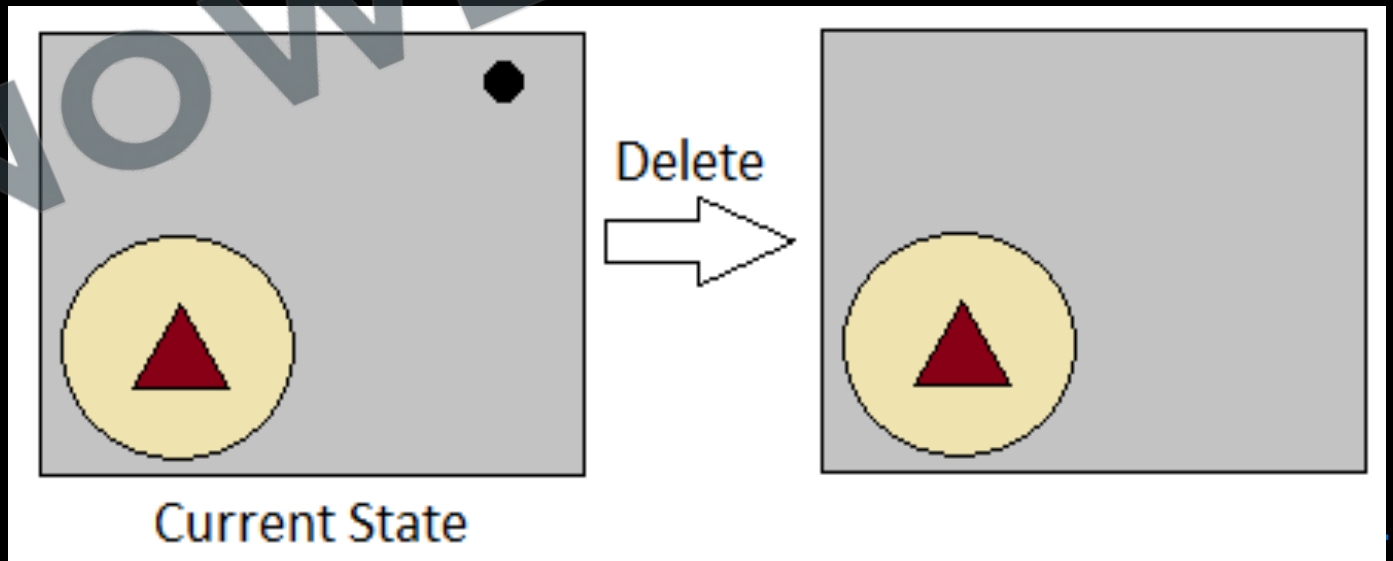
Process:

- **Break down the problem**: Split a large problem into major components and address these first.
- **Address sub-problems**: Once the major components are aligned towards the goal, solve any smaller issues that emerge.
- **Integrate strategies**: Use a blend of forward search (from the starting point) and backward search (from the goal) to guide the problem-solving process effectively.

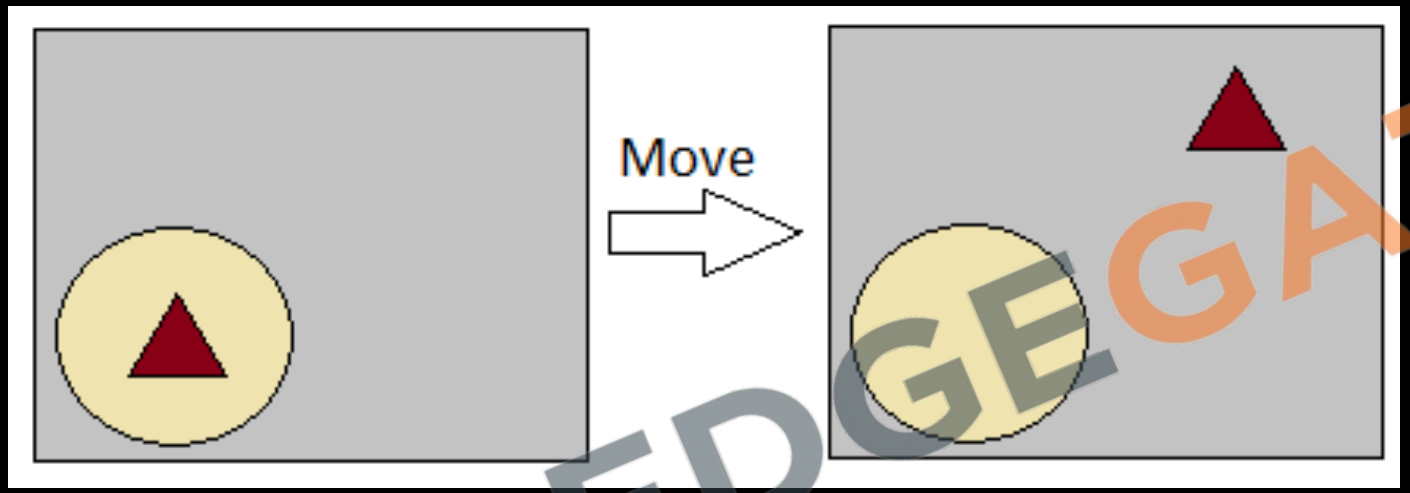
- We first evaluate the difference between the current and the goal state. For each difference, we will generate a new state and will apply the operators.



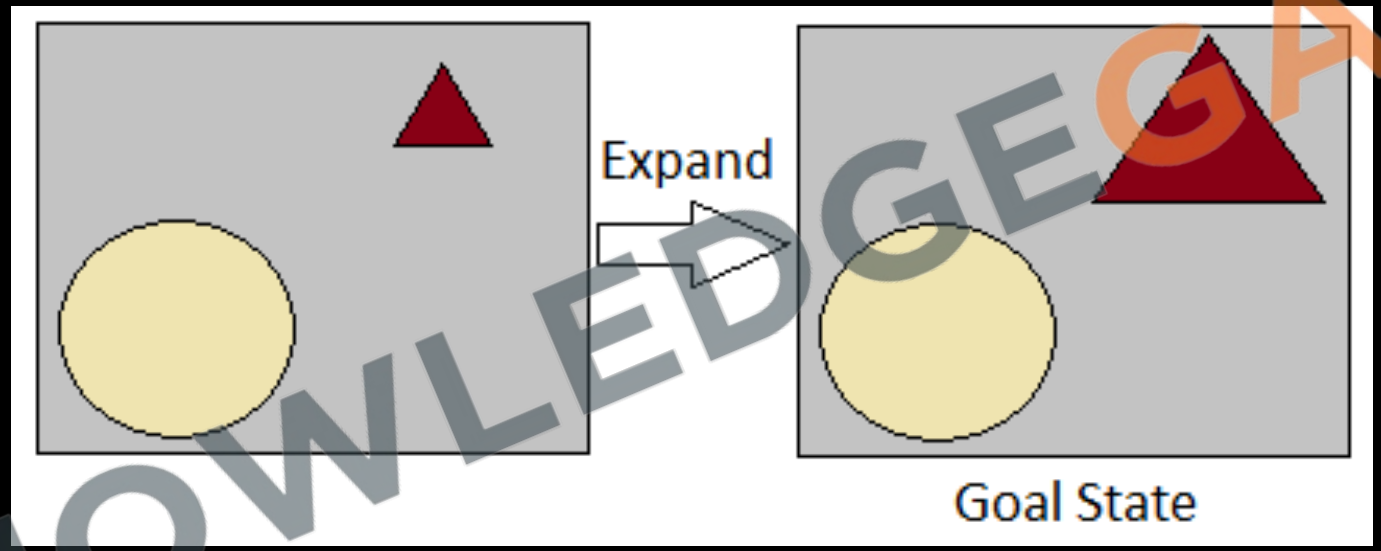
- **<u>Applying Delete operator</u>:** The first difference that we find is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



- **Applying Move Operator:** After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the triangle is outside the circle, so, we will apply the Move Operator.



- **Applying Expand Operator:** Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the triangle, so, we will apply Expand operator, and finally, it will generate the goal state.



Operator Subgoaling:

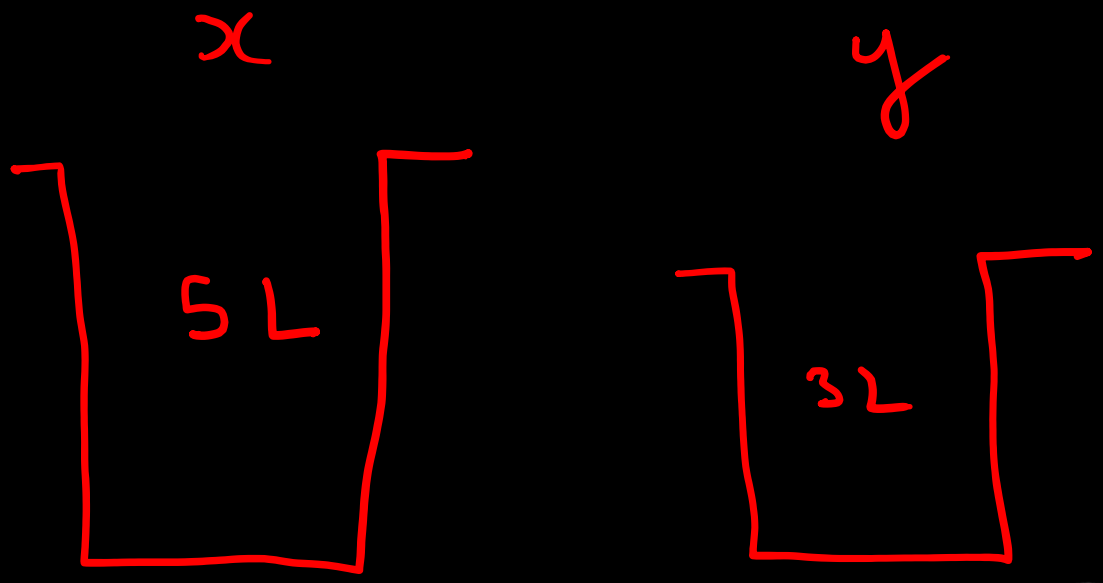
- **What it is**: A process within Means-Ends Analysis (MEA) for handling situations where a direct action (operator) cannot be applied to the current state.
- **Purpose**: To establish the necessary conditions (preconditions) for applying an operator by setting up sub-goals that modify the current state.

- **Means-Ends Analysis Steps**:
  - **Identify Differences**: Note the discrepancies between the current and goal states.
  - **Set Sub-goals**: Create intermediate goals aimed at reducing identified differences.
  - **Find Operators**: Look for actions that can achieve these sub-goals.
  - **Apply Operators**: Execute the most promising actions. If constraints prevent direct action, set new sub-goals to remove these constraints.
  - **Iterate**: Repeat these steps until the goal is achieved or no further progress can be made.

Water Jug Problem

- The Water Jug Problem is a classic example often used in computer science and artificial intelligence to demonstrate problem-solving methods such as Means-Ends Analysis. Here's a brief overview:

- **Problem Statement**: You have two jugs with different capacities (for example, one jug holds 3 liters and the other holds 5 liters) and a water source. The goal is to measure out a specific amount of water (say 4 liters), you can fill, empty, or transfer water between the jugs and no measurements marking on either jug.



→ Jug x can contain 5L

→ Jug y can contain 3L

Goal is to have 4L in Jug x

5 3

(x, y)

<0,0>

<5,0>

<2,3>

<2,0>

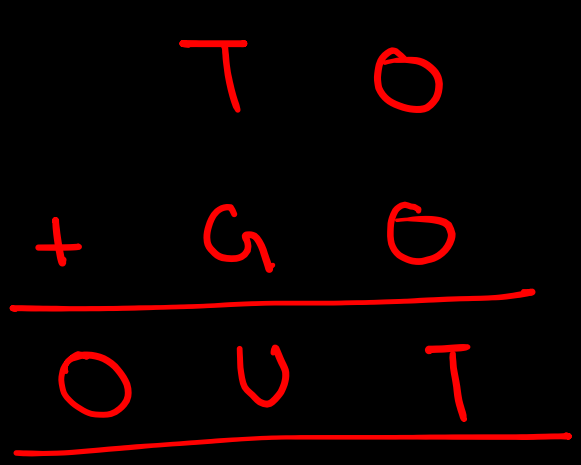
<0,2>

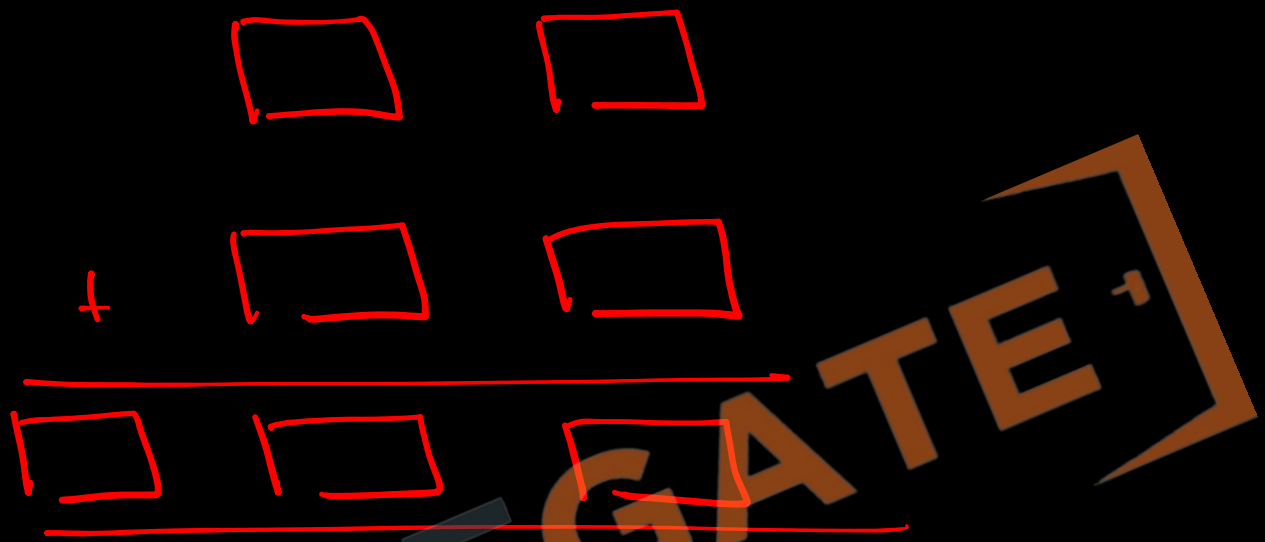
<5,2>

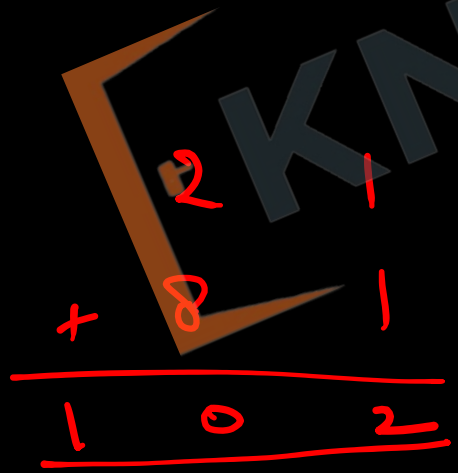
<4,3>

Constraints Satisfaction Problem

- CSPs are a key topic in computer science, especially in artificial intelligence (AI).
- **Definition**: A CSP consists of a set of variables, a domain for each variable, and a set of constraints that specify allowable combinations of values.
- **Variables**: Elements that have to be assigned values from their domains. For example, in a scheduling problem, variables could be different time slots.
- **Domains**: The range of possible values that each variable can take. For example, in a coloring problem, the domain might be a set of colors.
- **Constraints**: Rules that define which combinations of values are valid or invalid. They restrict the ways in which variables can be assigned values.
- **Solutions**: Assignments of values to all variables that do not violate any constraints. A problem can have zero, one, or multiple solutions.
- **Solving Methods**: There are various strategies for solving CSPs, including backtracking, forward checking, and constraint propagation.
- **Applications**: CSPs are widely used in many fields such as scheduling, planning, resource allocation, timetabling, and in games and puzzles like Sudoku.







C_4 C_3 C_2 C_1

```
  S E N D
+ M O R E
---------
M O N E Y
```

$M = 1$

$S + M \geq 10$

$\therefore S = 9$

$E + O = N$ if $C_2 = 0$ $[E = N]$

$\therefore C_2 = 1$

$O = 0$

let $E = 5$

then $N = 6$

$D + E = Y$

$D + 5 = Y$

if $C_1 = 1$

then $D \geq 5$ ×

6 ×

7 ✓

8 ×

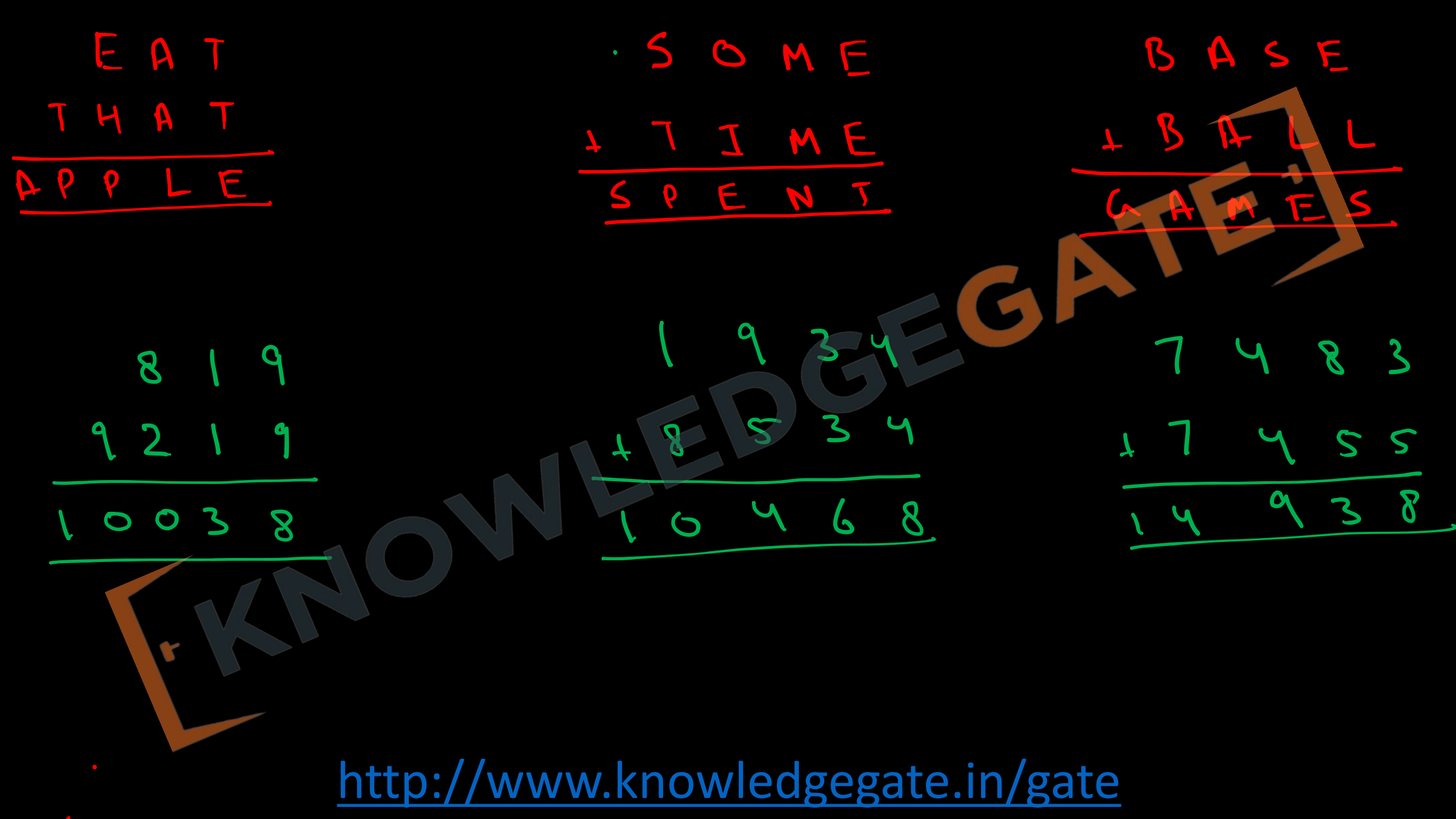
9 ×

```
  9 5 6 7
+ 1 0 8 5
---------
1 0 6 5 2
```

if $C_1 = 1$

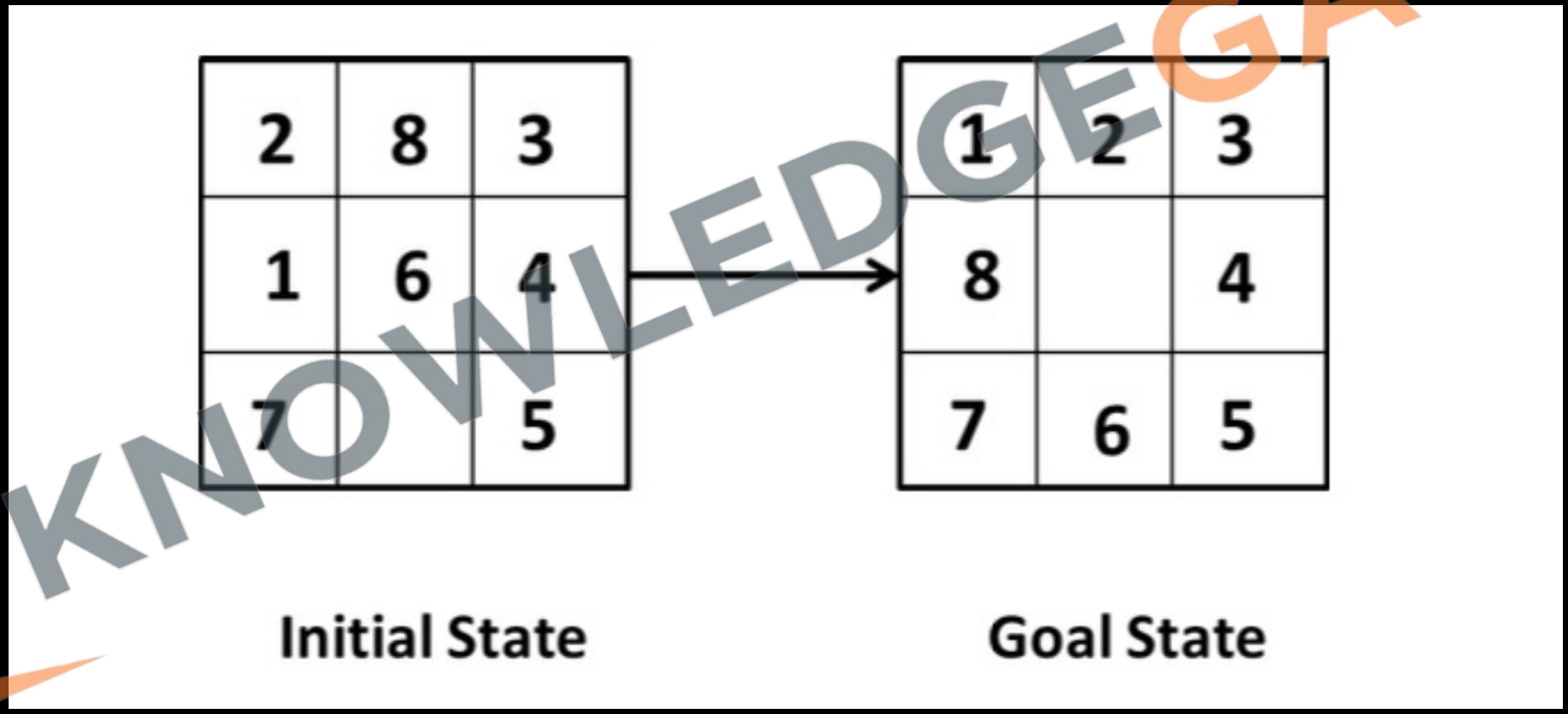
$R = 8$

$D = 7$, $Y = 2$

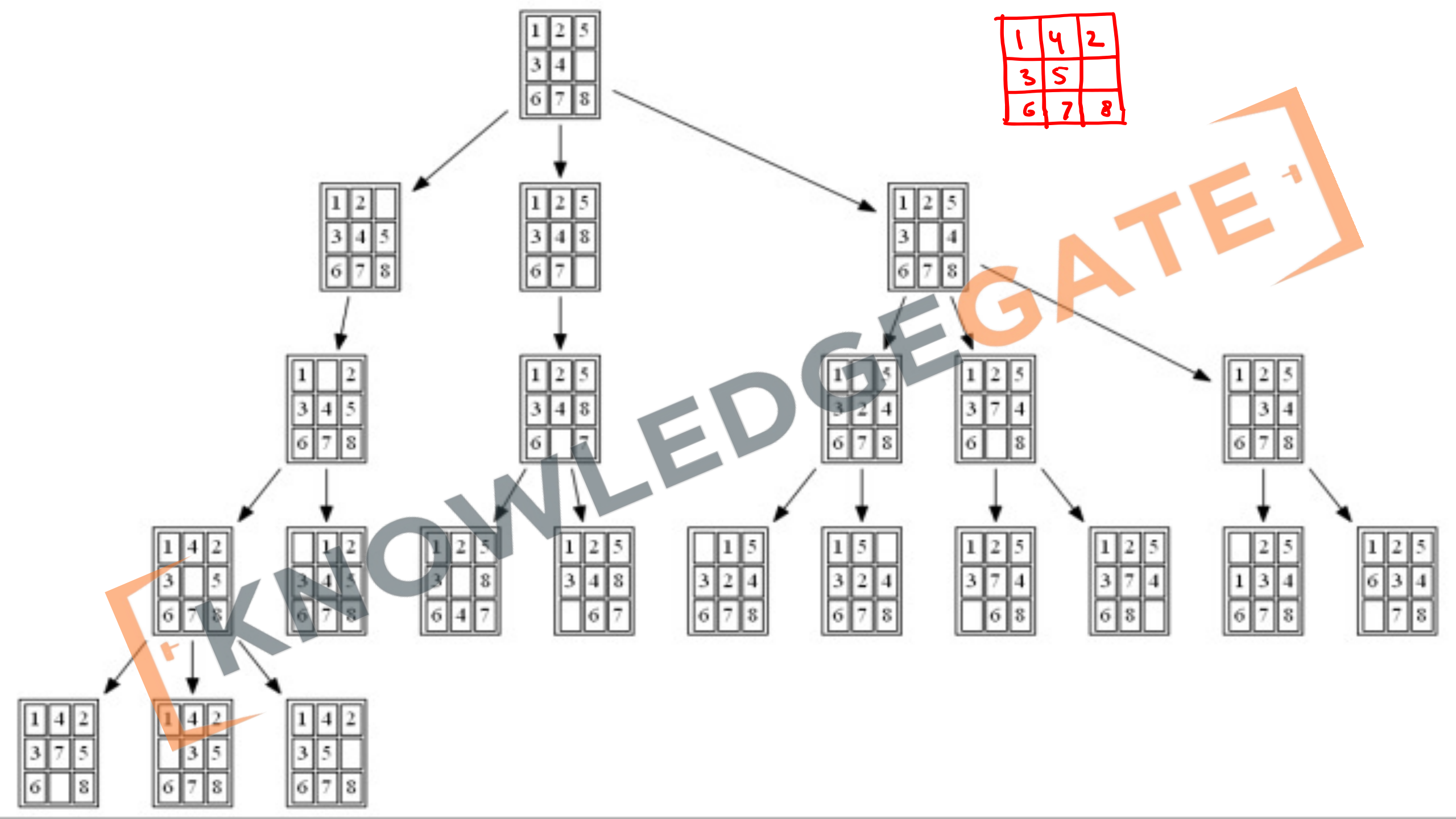

EAT
THAT
APPLE
8 1 9
9 2 1 9
1 0 0 3 8
SOME
+ TIME
SPENT
1 9 3 4
+ 8 5 3 4
1 0 4 6 8
BASE
+ BALL
GAMES
7 4 8 3
+ 7 4 5 5
1 4 9 3 8
KNOWLEDGEGATE
http://www.knowledgegate.in/gate

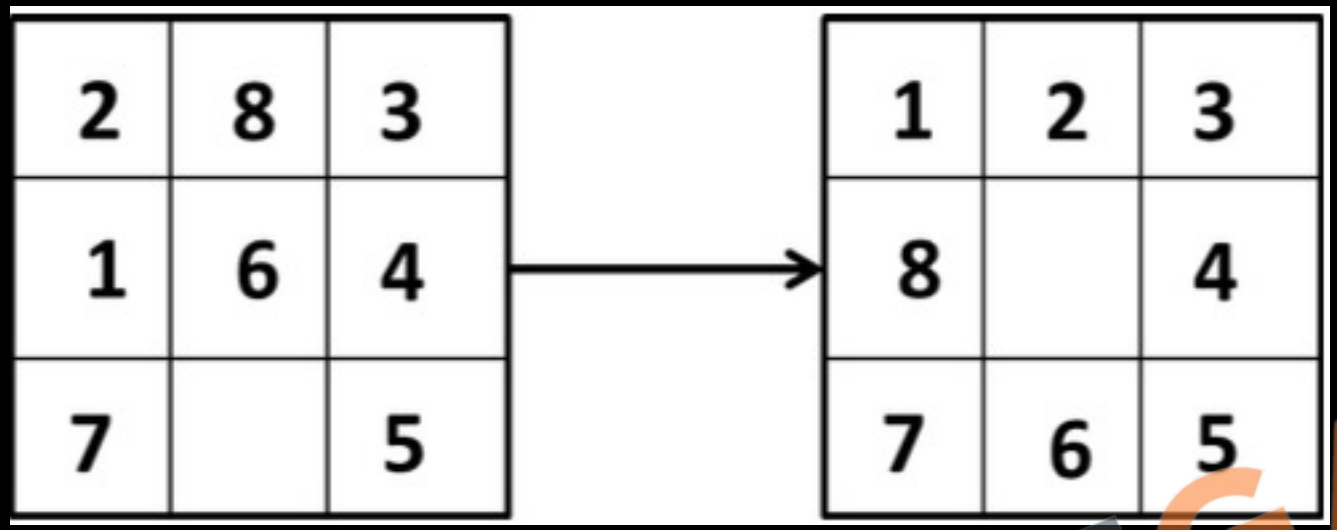
The 8-puzzle Problem

- The 8-puzzle problem is a classic search problem in the field of Artificial Intelligence (AI) and computer science. It involves a 3x3 grid with 8 numbered tiles and one empty space. The goal is to move the tiles around until they are in a specific goal state.



- **Grid Structure**: The puzzle consists of a 3x3 grid, making 9 spaces. There are 8 numbered tiles and one empty space (often represented by 0 or a blank).
- **Initial State**: The puzzle starts in a random configuration with the tiles not in order.
- **Goal State**: The objective is to arrange the tiles in numerical order.
- **Moves**: You can move a tile into the empty space if it is adjacent to it (up, down, left, or right). Diagonal moves are not allowed.
- **Difficulty Levels**: Not all initial states are solvable.
- **Search Algorithms**: To find a solution, various search algorithms can be used, such as Breadth-First Search (BFS), Depth-First Search (DFS), A* Search, and others. These algorithms explore possible moves from the initial state to reach the goal state.
- **Heuristics**: For more efficient search, heuristics such as the Manhattan distance or the number of misplaced tiles can be used, especially in algorithms like A* to estimate the cost to reach the goal from a given state.
- **Applications**: Solving the 8-puzzle problem helps in understanding important concepts in AI and algorithms, such as state space, search strategies, and the use of heuristics.

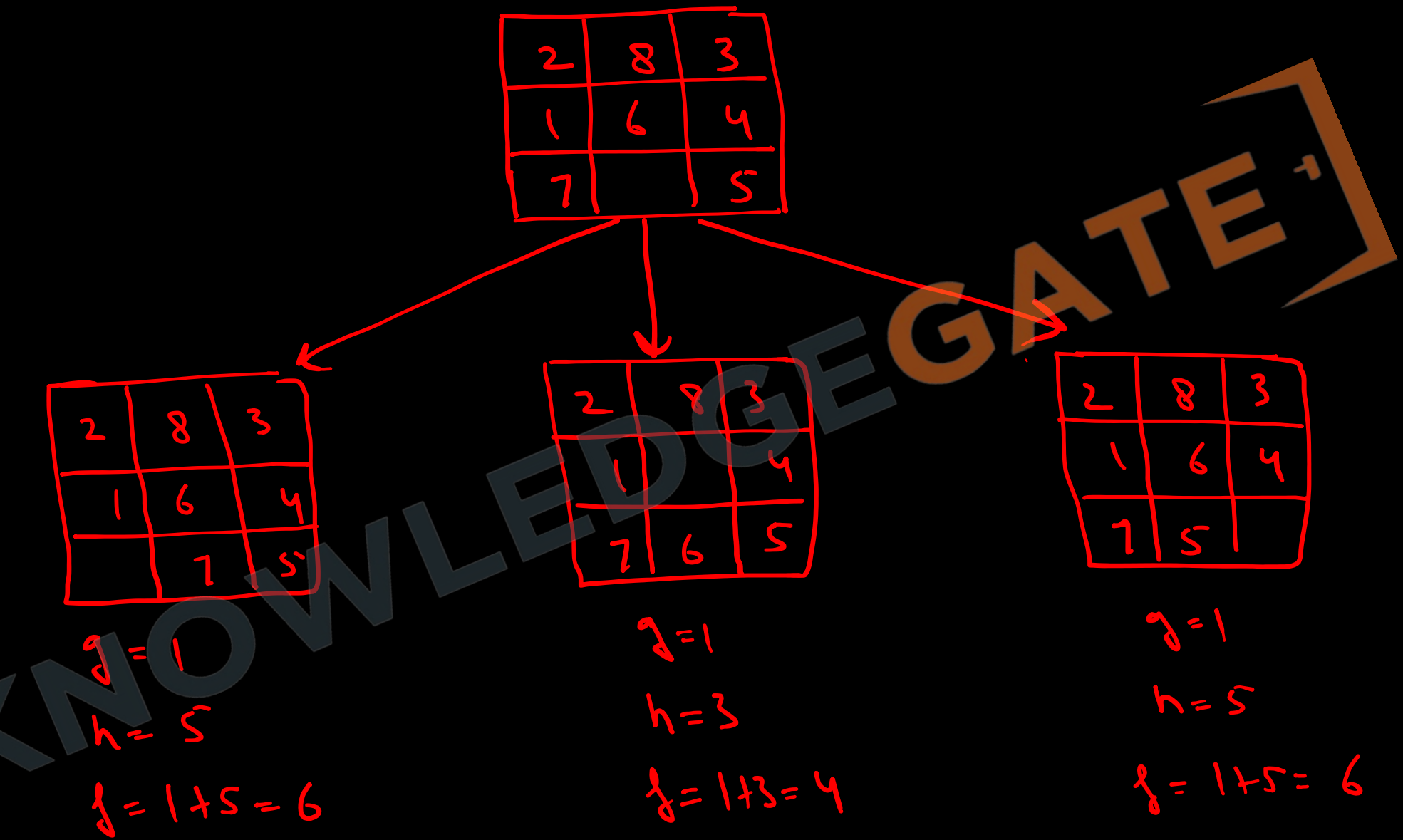

1 4 2
3 5
6 7 8
KNOWLEDGEGATE



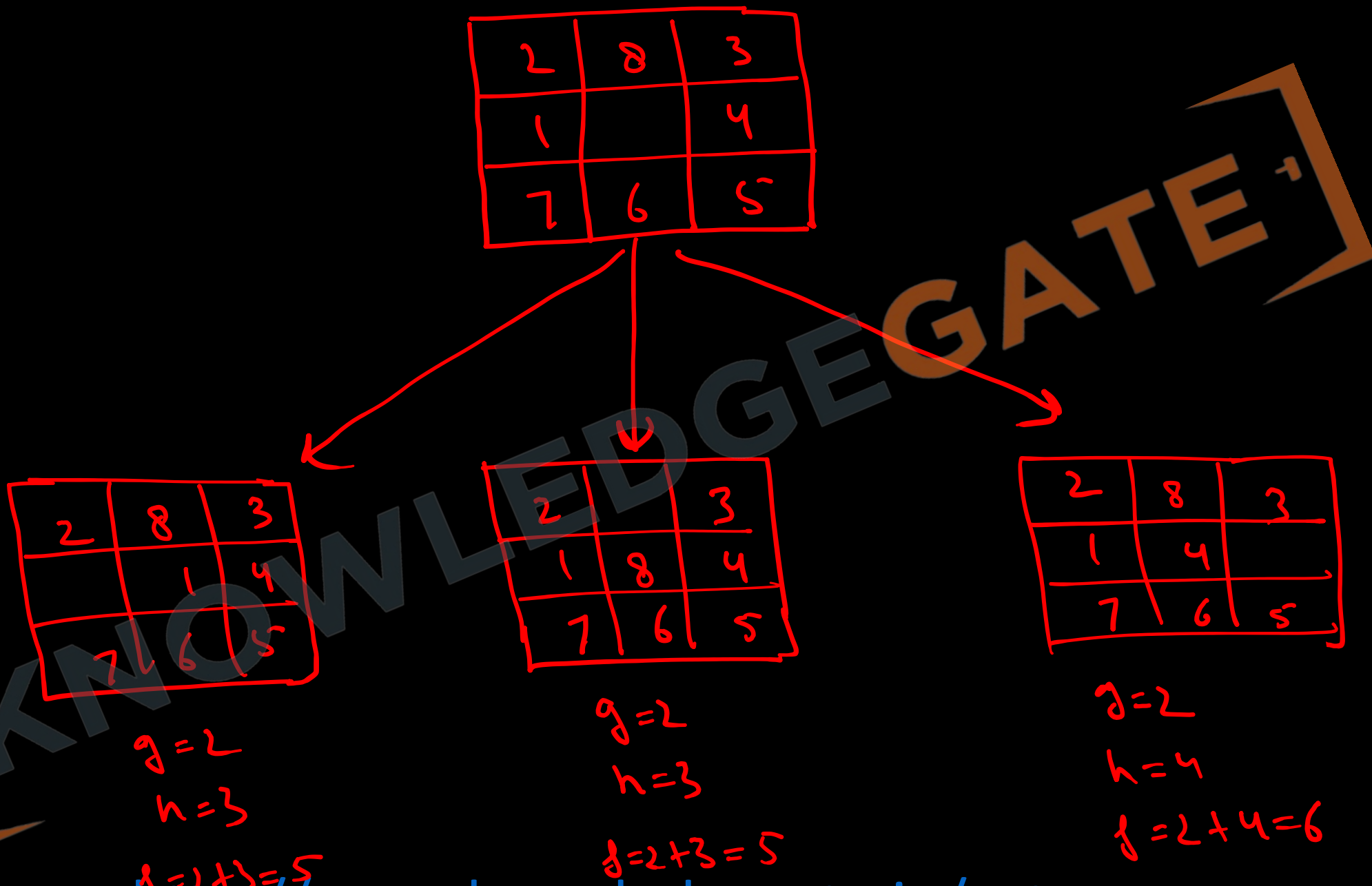
$g = 0$, $h = 4$

- Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
- f(n) = g(n) + h(n)
- Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

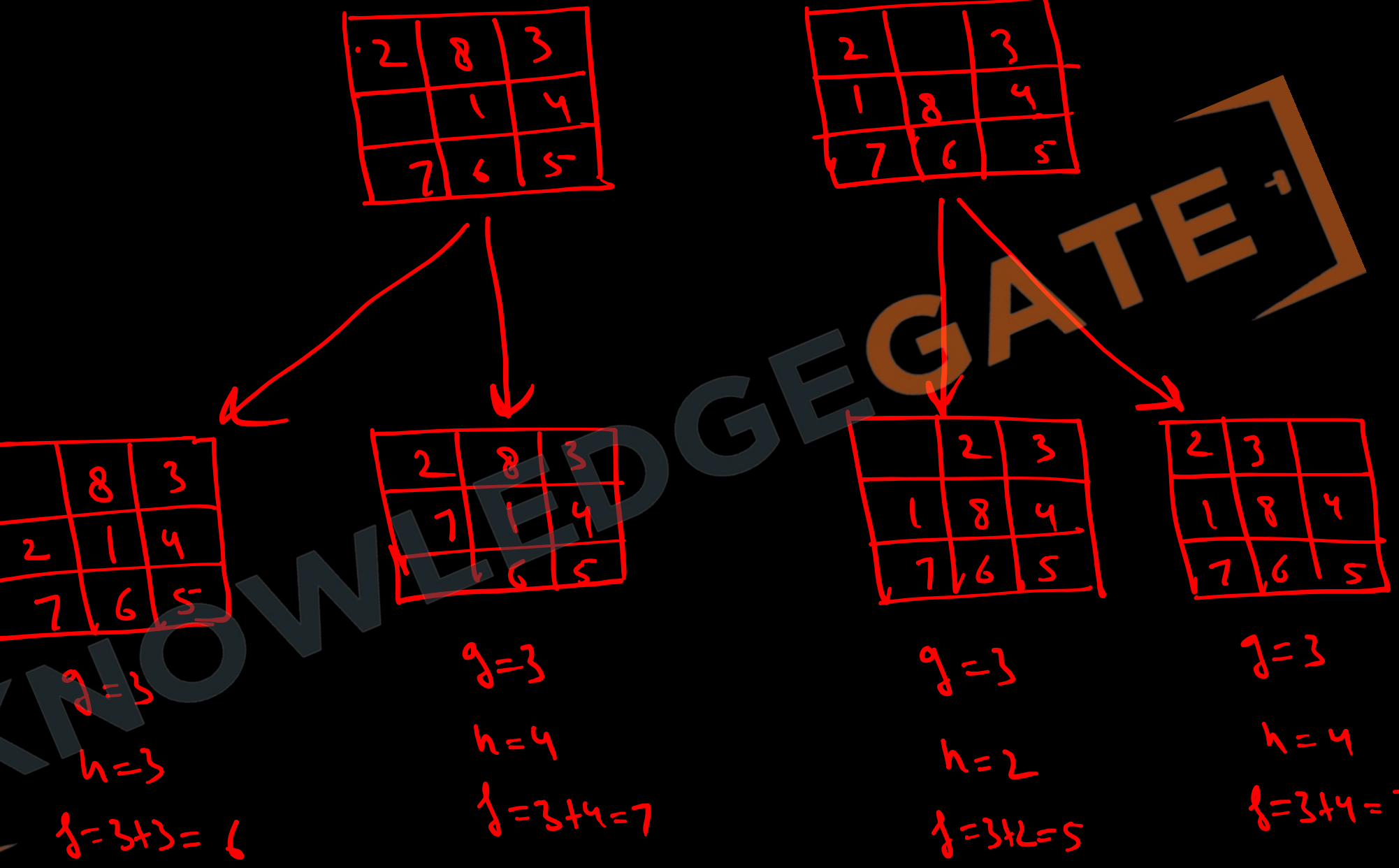
1	2	3
8		4
7	6	5



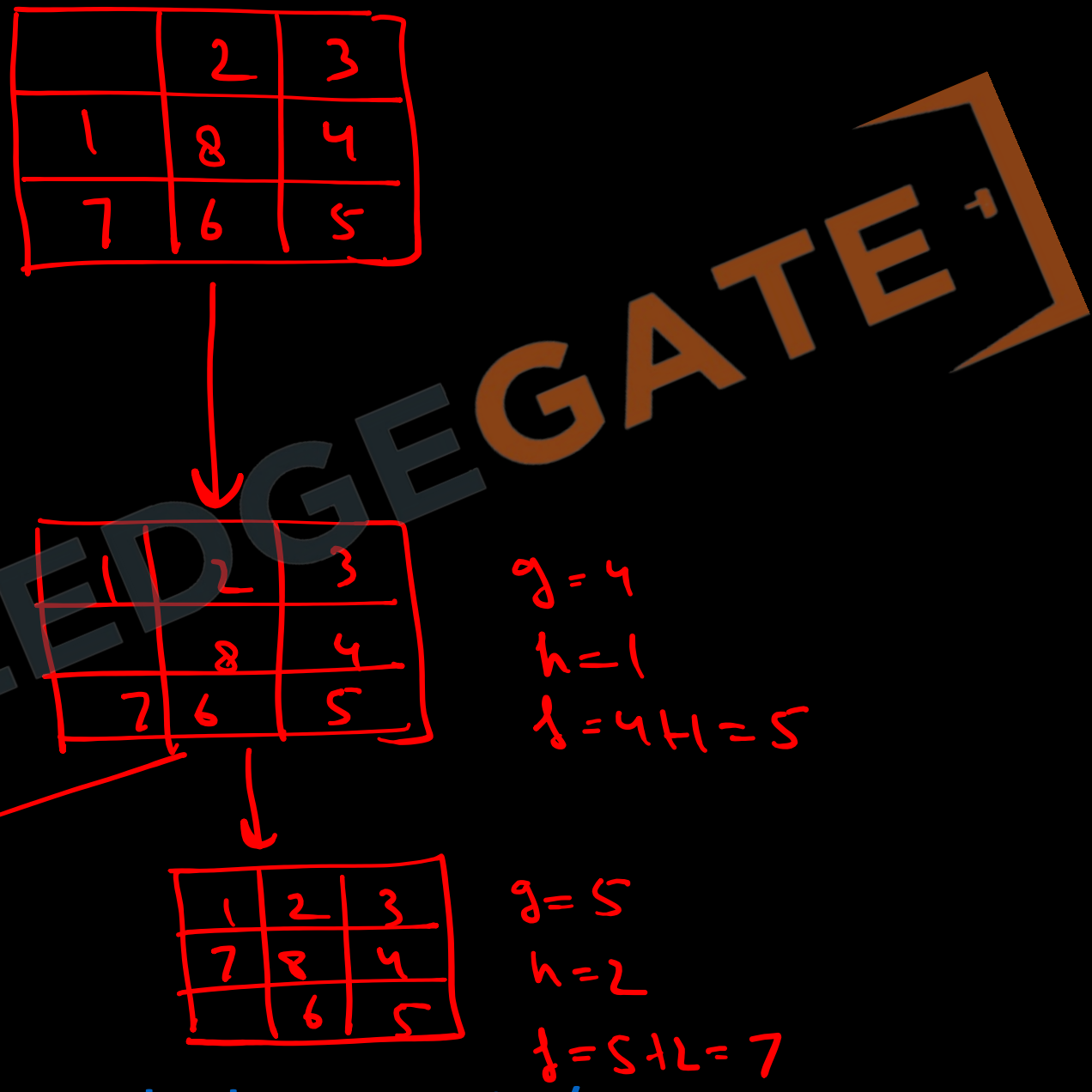
1	2	3
8		4
7	6	5



1	2	3
8		4
7	6	5

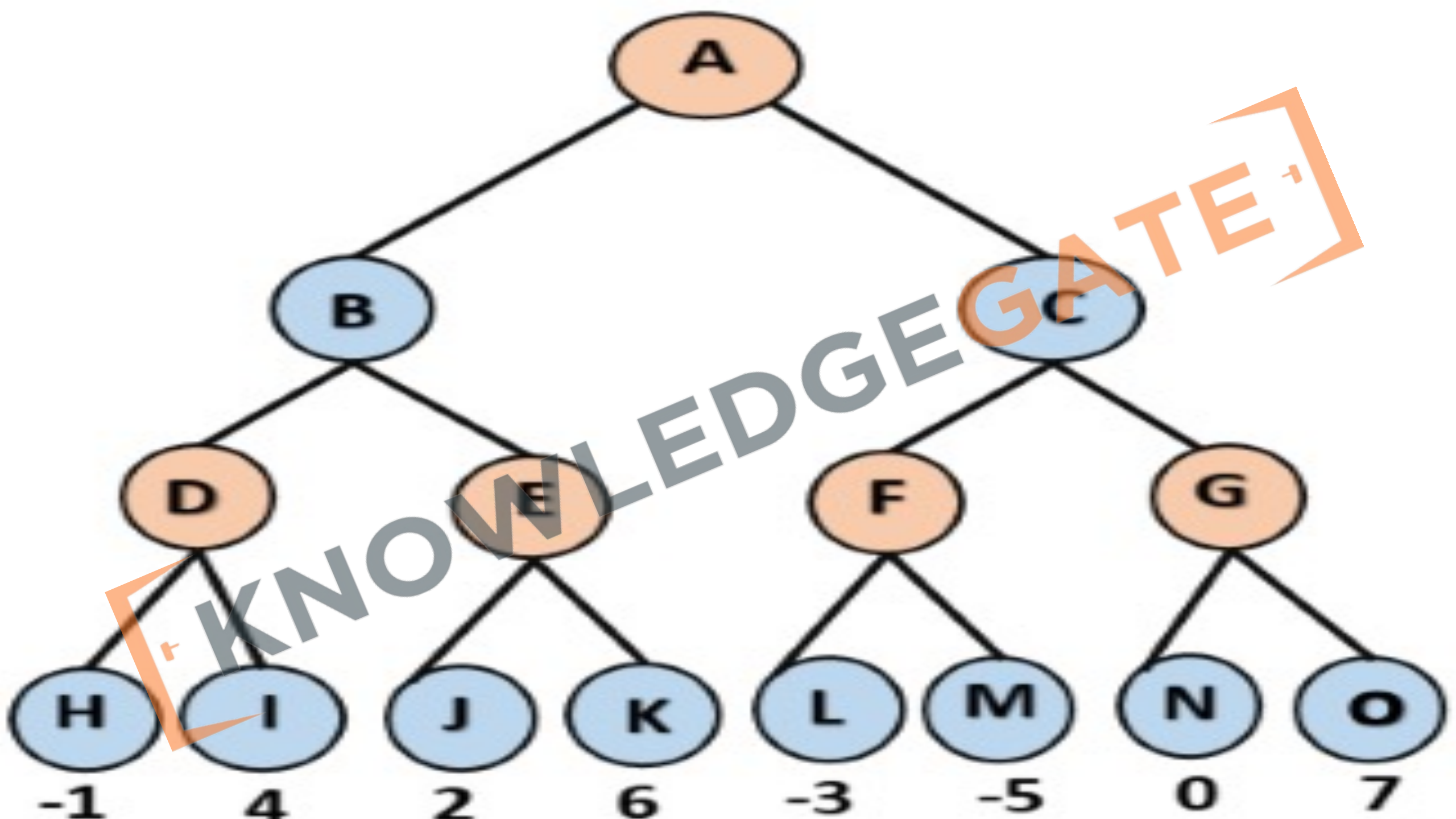


1	2	3
8		4
7	6	5



The minimax algorithm

- **Definition**: Minimax is a specialized search algorithm used in game playing to determine the optimal sequence of moves for a player in a zero-sum game involving two players.
- **Mechanism**: It operates by recursively using backtracking to simulate all possible moves in the game, effectively searching through a game tree.
- **Decision Making**: The algorithm computes the minimax decision for the current state and uses a depth-first search algorithm for the exploration of the complete game tree.
- **Players' Roles**: In the game tree, there are two types of nodes: MAX nodes, where the algorithm selects the move with the maximum value, and MIN nodes, where the algorithm selects the move with the minimum value.
- **Games**: It's typically applied to perfect information games like chess, checkers, and tic-tac-toe.
- **Initial Values**: In the algorithm, the MAX player is often represented with negative infinity (-∞) as the initial worst case, while MIN is represented with positive infinity (∞).

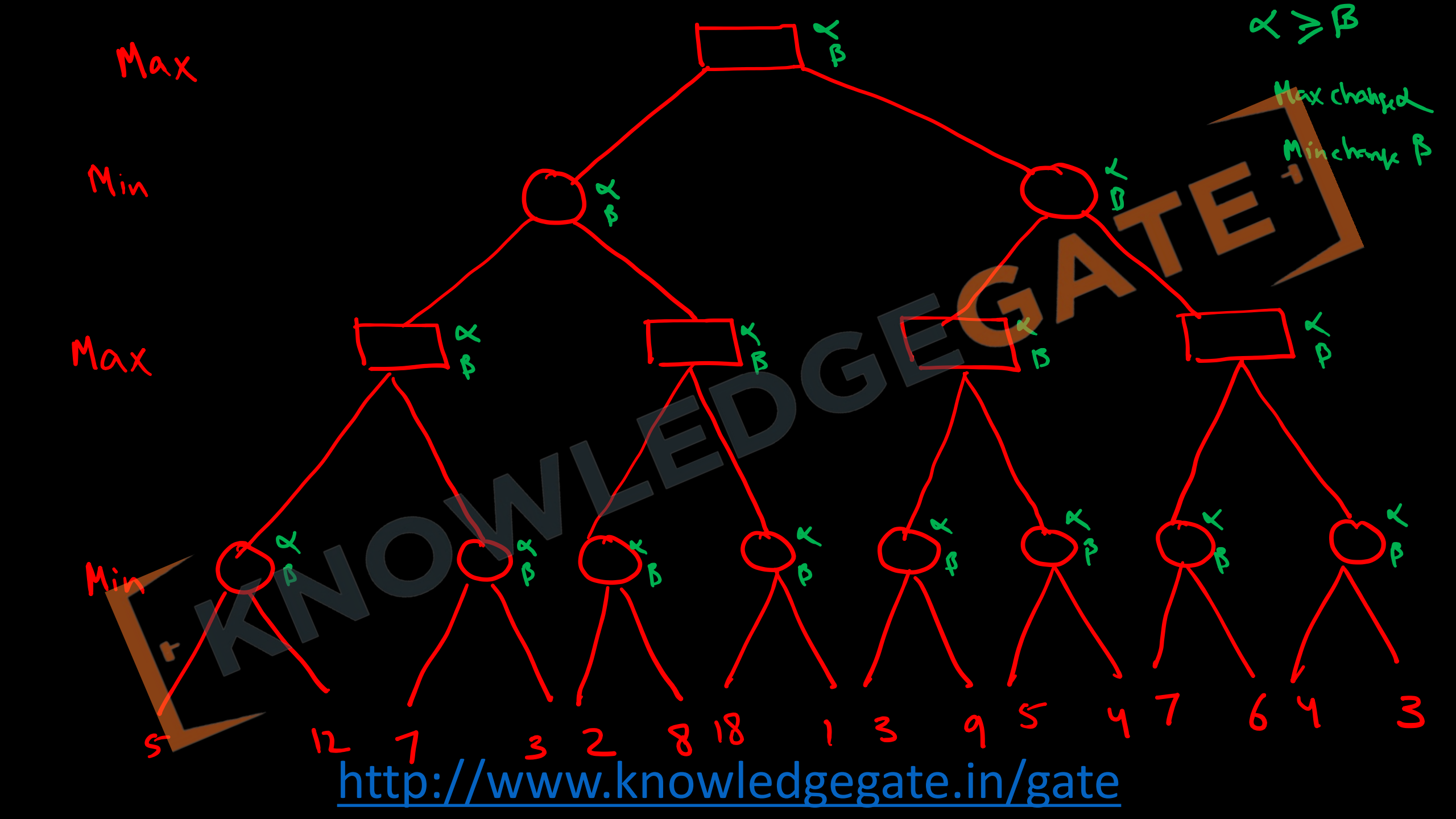


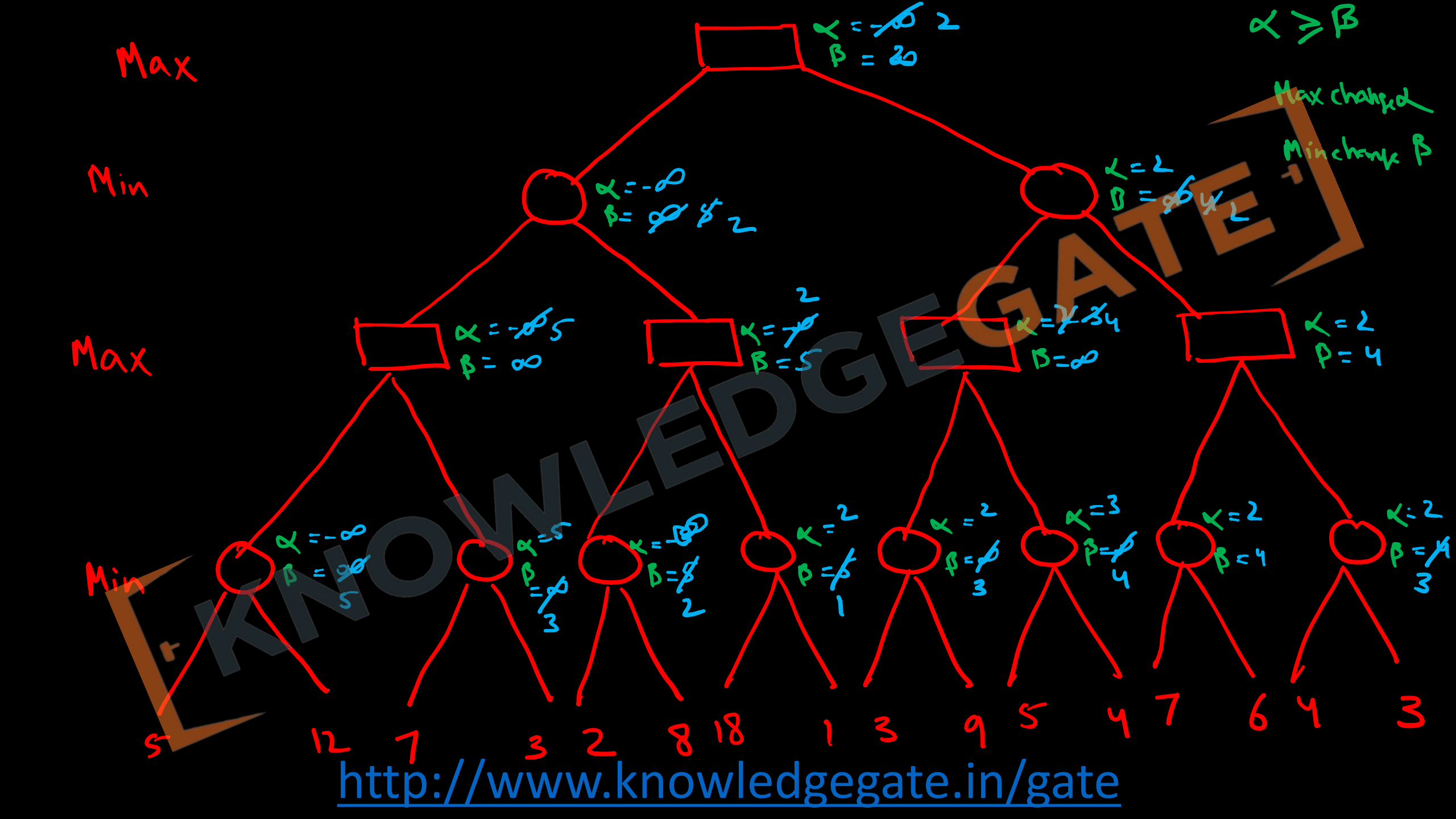
- **Properties**:
  - **Completeness**: The algorithm is complete and will definitely find a solution if one exists.
  - **Optimality**: It guarantees to find the optimal strategy for both players.
  - **Time Complexity**: The time complexity is O(b^m), where b is the branching factor (the average number of child nodes for each node in the tree) and m is the maximum depth of the tree.
  - **Space Complexity**: The space complexity is also O(b^m), because of the need to store all the nodes in memory.

- **Limitations**:
  - The algorithm can be slow for complex games with many possible moves (like chess).
  - A game like chess can have around 35 possible moves at any point, leading to a vast number of possible game states to evaluate.

ALPHA–BETA PRUNING

- Alpha-beta pruning is an optimization technique for the minimax algorithm. It significantly reduces the number of nodes that need to be evaluated in the game tree without affecting the final result. Alpha-beta pruning skips the evaluation of certain branches in the game tree by identifying moves that are evidently worse than previously examined moves.
- **Alpha Value**: The best value that the maximizer currently can guarantee at that level or above.
- **Beta Value**: The best value that the minimizer currently can guarantee at that level or above.
- **Initial Values**: Alpha starts at negative infinity (-∞), and Beta starts at positive infinity (∞).
- **When to Prune**: A branch is pruned when the minimizer's best option (Beta) is less than the maximizer's best option (Alpha) since the maximizer will never allow the game to go down a path that could lead to a worse outcome than it has already found.



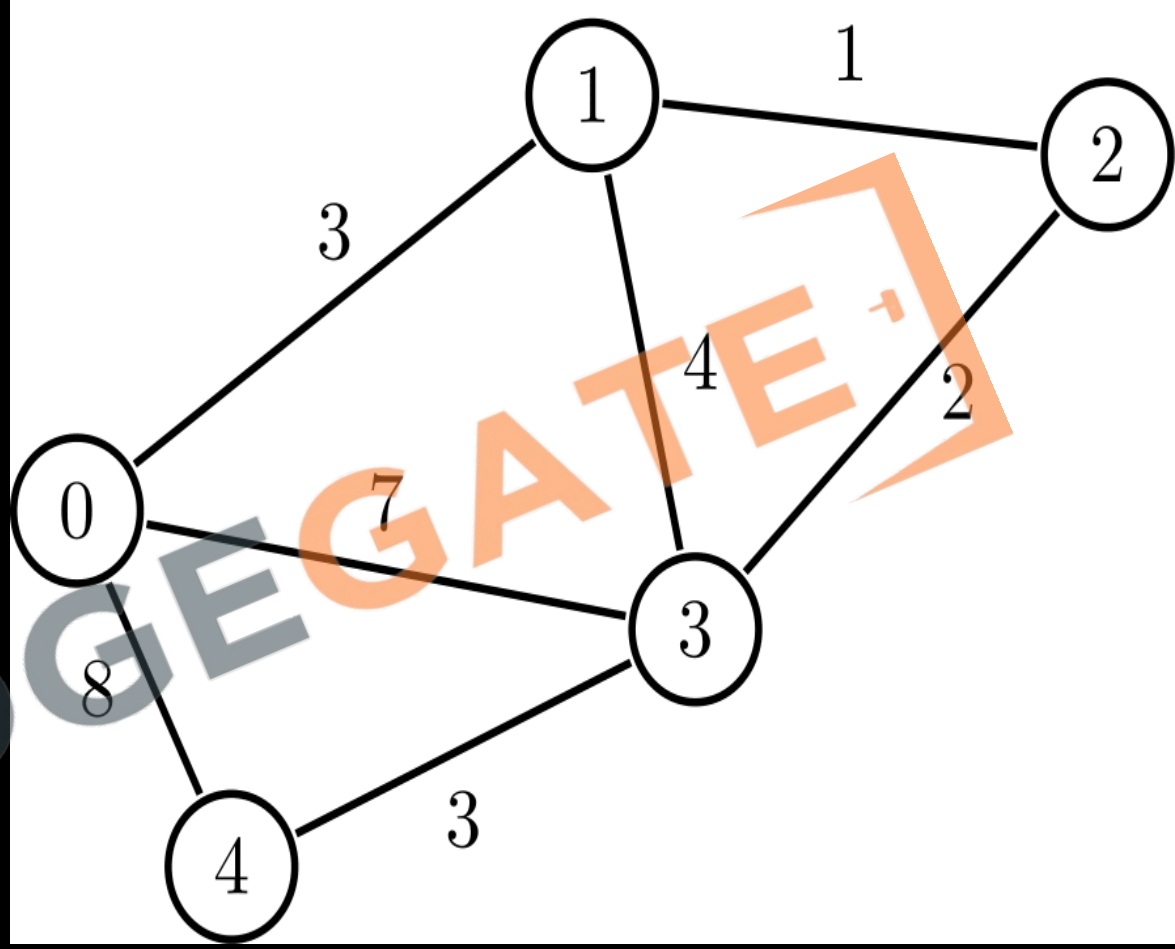


Benefits:

- **Efficiency**: Alpha-beta pruning can greatly reduce the number of nodes that are explored in the game tree, leading to faster decision-making.
- **Optimality**: The final decision made by the alpha-beta pruning is the same as the decision that would be made by the full minimax algorithm, ensuring optimality is not compromised.
- **Widely Applicable**: It can be applied to any game tree, not just two-player games, though the algorithm assumes perfect play on both sides.

Branch and Bound

- **Objective**: To find the optimal solution in problems like the traveling salesman, knapsack, and job scheduling.
- **Mechanism**: Searches a state space tree of potential solutions by branching and bounding.
- **Branching**: Expands nodes, creating children that represent partial solutions.
- **Bounding**: Uses bounds to estimate the minimum or maximum value of regions in the space; prunes if the bound is worse than the best solution so far.
- **Strategies**: Can use best-first, breadth-first, or depth-first search to explore nodes.
- **Pruning**: Skips branches that can't improve the best current solution.
- **Efficiency**: Reduces the search space, making some intractable problems solvable.
- **Result**: Finds the global optimum if the bounding function is accurate.



0/1 Knapsack using Branch and Bound

i	1	2	3	4
P	31	29	21	24
W	6	8	5	3

Chapter-3

(KNOWLEDGE REPRESENTATION)

LOGICAL AGENTS

- Intelligence comes from Knowledge and Knowledge comes from the ability to reason. **Knowledge-based agents**: The intelligence of humans is achieved—not by purely reflex mechanisms but by processes of **reasoning** that operate on internal **representations** of knowledge.

- In AI, this approach to intelligence is embodied in **knowledge-based agents**. **Logical Agents:** Agents that can form representations of a complex world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do.

KNOWLEDGE-BASED AGENTS

- The central component of a knowledge-based agent is its **knowledge base**, or KB which may initially contain some **background knowledge**. A knowledge base is a set of **sentences**. Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world.

- Sometimes we dignify a sentence with the name **axiom.** We add new sentences to the knowledge base and query what is known through **Inferences** (deriving new sentences from old).

- A **proposition** is a declarative sentence (that is, a sentence that declares a fact) that is either true or false, but not both.
  1. Delhi is the capital of USA
  2. How are you doing
  3. 5<= 11
  4. Temperature is less than 10 C
  5. It is cold today
  6. Read this carefully
  7. X + y = z

- Premises is a statement that provides reason or support for the conclusion(proposition). Premises(proposition) is always considered to be true.
- If a set of Premises(P) yield another proposition Q(Conclusion), then it is called an Argument.
- An argument is said to be valid if the conclusion Q can be derived from the premises by applying the rules of inference.

$\{P_1, P_2, P_3..., P_N\} \vdash Q$	P_1 P_2 P_3 . . . P_N ..... Q	$\{P_1 \wedge P_2 \wedge P_3 \wedge .... \wedge P_N\} \vdash Q$

Types of proposition

- We use letters to denote **propositional variables** to represent propositions, just as letters are used to denote numerical variables. The conventional letters used for propositional variables are *p, q, r, s.*
- Many mathematical statements are constructed by combining one or more propositions. New propositions, called **compound propositions**, are formed from existing propositions using logical operators.

Operators / Connectives

1. **Negation: -** let p be a proposition, then negation of p new proposition, denoted by ¬*p*, is the statement "it is not the case that p".

Negation	
P	¬P
F	
T	

Conjunction

- Let *p* and *q* be propositions. The *conjunction* of *p* and *q*, denoted by *p* ∧ *q*, is the proposition "*p* and *q*." The conjunction *p* ∧ *q* is true when both *p* and *q* are true and is false otherwise.

Conjunction		
p	q	p ∧ q
F	F	
F	T	
T	F	
T	T	

Disjunction

- Let *p* and *q* be propositions. The *disjunction* of *p* and *q*, denoted by *p* ∨ *q*, is the proposition. "*p* or *q*." The disjunction *p* ∨ *q* is false when both *p* and *q* are false and is true otherwise.

Disjunction		
p	q	p ∨ q
F	F	
F	T	
T	F	
T	T	

Implication

1. Let *p* and *q* be propositions. The *conditional statement* $p \rightarrow q$ is the proposition "if *p*, then q". The conditional statement $p \rightarrow q$ is false when *p* is true and *q* is false, and true otherwise.
2. In conditional statement $p \rightarrow q$, *p* is called the hypothesis (or antecedent or premise) and q is called the conclusion.

Implication		
p	q	$p \rightarrow q$
F	F	
F	T	
T	F	
T	T	

p	q	P → q	¬p	¬q	¬P → ¬q	q → p	¬q → ¬p	¬p V q
F	F							
F	T							
T	F							
T	T							

- $p \rightarrow q$ *implication*
- $q \rightarrow p$ *converse*
- $\neg p \rightarrow \neg q$ *inverse*
- $\neg q \rightarrow \neg p$ *contra positive*
- $p \rightarrow q = \neg q \rightarrow \neg p$
- $p \rightarrow q$ *will be true if either p is false or q is true,* $p \rightarrow q = \neg p \vee q$

Bi-conditional

- Let *p* and *q* be propositions. The bi*conditional statement* $p \leftrightarrow q$ is the proposition.
  - " *p if and only* q".
  - "*p* is necessary and sufficient for *q*"
  - "if *p* then *q*, and conversely"
  - "*p* iff *q*."
  - $p \leftrightarrow q = (p \rightarrow q) \wedge (q \rightarrow p)$
- The biconditional statement $p \leftrightarrow q$ *is true when p and q have the same values, and false otherwise.*

Bi-conditional		
p	q	P ↔ q
F	F	
F	T	
T	F	
T	T	

Type of cases

- **Tautology/valid: -** A propositional function which is always having truth in the last column, is called tautology. E.g. $p \vee \neg p$

p	¬ p	p V ¬ p
F	T	
T	F	

- **<u>Contradiction/ Unsatisfiable</u>: -** A propositional function which is always having false in the last column, is called Contradiction. E.g. $p \wedge \neg p$

p	¬ p	p ∧ ¬ p
F	T	
T	F	

- **Contingency:** - A propositional function which is neither a tautology nor a contradiction, is called Contingency. E.g. ***p*** ∨ ***q***

p	q	p ∨ q
F	F	
F	T	
T	F	
T	T	

- **Satisfiable: -** A propositional function which is not contradiction is satisfiable. i.e. it must have at least one truth value in the final column e.g. $p \vee q$

De Morgan's Laws

- $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$

P	Q	$\neg(P \wedge Q)$	$\neg P \vee \neg Q$
T	T	F	F
T	F	T	T
F	T	T	T
F	F	T	T

- $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$

P	Q	$\neg(P \vee Q)$	$\neg P \wedge \neg Q$
T	T	F	F
T	F	F	F
F	T	F	F
F	F	T	T

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Well Formed Formulas

- A sentence S is also called a well formed formula is defined as follows:
- A symbol S is a sentence.
- If S is a sentence then ¬S is also a sentence
- If S is a sentence then (S) is also a sentence.
- If S and T are sentences then:
  - S ∧ T
  - S ∨ T
  - S ⇒ T
  - S ⇔ T

  Are also a sentence.
- A sentence resulted from finite number of applications of the above rules.

TABLE 1 Rules of Inference.

Rule of Inference	*Name*
p $p \rightarrow q$ $\therefore q$	Modus ponens
$\neg q$ $p \rightarrow q$ $\therefore \neg p$	Modus tollens
$p \rightarrow q$ $q \rightarrow r$ $\therefore p \rightarrow r$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	Disjunctive syllogism
p $\therefore p \vee q$	Addition
$p \wedge q$ $\therefore p$	Simplification
p q $\therefore p \wedge q$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	Resolution

Disadvantages of Propositional Logic

- Propositional logic is too puny a language to represent knowledge of complex environments in a concise way. Thus we introduce first order logic which is sufficiently expressive to represent a good deal of our commonsense knowledge.

First order Predicate Logic

- Sometime propositional logic cannot derive any meaningful information even though, we as human can understand that argument is meaningful or not.

- P_1: Every Indian like cricket
- P_2: Sunny is an Indian
- Q: Sunny Likes cricket

- The reason propositional logic fails here because using only inference system we can not conclude Q from P_1 and P_2.

- In first order logic we understand, a new approach of subject and predicate to extract more information from a statement
  - 1 is a natural number (1 is subject, natural number is predicate)
  - we can write FOPL (short hand notation) for this as NatNo(1) = 1 is natural number
  - Similarly, we can understand the meaning of NatNo(2) as 2 is a natural number
  - NatNo(x): x is a natural number

- If i say four Indian are there I_1, I_2, I_3, I_4
- I_1 likes cricket $\wedge$ I_2 likes cricket $\wedge$ I_3 likes cricket $\wedge$ I_4 likes cricket
- Cricket(I_1) $\wedge$ Cricket(I_2) $\wedge$ Cricket(I_3) $\wedge$ Cricket(I_4)
- But problem with this notation is as there is 130+ corers Indian this formula will become very long and in some case we actually do not know how many subjects are there in the universe of discourse. so, we again need a short hand formula.
- $\forall_x$ Cricket(x), if we confine x to be Indian then it means every x like cricket.

- **Universal quantifiers**: - The universal quantification of a propositional function is the proposition that asserts

- P(x) is true for all values of x in the universe of discourse.

- The universe of discourse specifies the possible value of x.

- $\forall_x$ P(x), i.e. for all value of a P(x) is true

- Let try some other statement ‘Some Indian like samosa’
  - if i say four Indian are there I_1, I_2, I_3, I_4
  - I_1 like samosa $\vee$ I_2 like samosa $\vee$ I_3 like samosa $\vee$ I_4 like samosa
  - Samosa(I_1) $\vee$ Samosa(I_2) $\vee$ Samosa(I_3) $\vee$ Samosa(I_4)
  - $\exists_x$ Samosa(x), if we confine x to be Indian then it means some x likes samosa.

- **Existential quantifiers**: - with existential quantifier of a propositional that is true if and only if P(x) is true for at least one value of x in the universe of discourse.

- There exists an element x is the universe of discourse such that P(x) is true.

- $\exists_x P(x)$, i.e. for at least one value of a P(x) is true

- let’s change the universe of discourse from Indian to human
  - if human is Indian then it likes cricket
  - Indian(x): x is an Indian
  - Cricket(x): x likes Cricket

  - if I_1 is Indian then likes cricket $\wedge$ if I_2 is Indian then likes cricket $\wedge$ if I_3 is Indian then likes cricket $\wedge$ if I_4 is Indian then likes cricket

  - $[Indian(I_1) \rightarrow cricket(I_1)] \wedge [Indian(I_2) \rightarrow cricket(I_2)] \wedge [Indian(I_3) \rightarrow cricket(I_3)] \wedge [Indian(I_4) \rightarrow cricket(I_4)]$

  - $\forall_x [Indian(x) \rightarrow cricket(x)]$

- let’s change the universe of discourse from Indian to human
  - if human is Indian then it likes samosa
  - Indian(x): x is an Indian
  - Samosa(x): x likes Samosa

  - if I_1 is Indian then likes samosa $\vee$ if I_2 is Indian then likes samosa $\vee$ if I_3 is Indian then likes samosa $\vee$ if I_4 is Indian then likes samosa

  - $[Indian(I_1) \wedge samosa(I_1)] \vee [Indian(I_2) \wedge samosa(I_2)] \vee [Indian(I_3) \wedge samosa(I_3)] \vee [Indian(I_4) \wedge samosa(I_4)]$

  - $\exists_x [Indian(x) \wedge samosa(x)]$

Negation

- $\neg [\forall_x P(x)] = \exists_x \neg P(x)$
- $\neg [\exists_x P(x)] = \forall_x \neg P(x)$

Q consider the statement $\exists_x[P(x) \wedge \neg Q(x)]$, Which of the following is equivalent?

a) $\forall_x [P(x) \rightarrow Q(x)]$

b) $\forall_x [\neg P(x) \rightarrow Q(x)]$

c) $\neg \{\forall_x [P(x) \rightarrow Q(x)]\}$

d) $\neg \{\forall_x [\neg P(x) \rightarrow Q(x)]\}$

Q let in a set of all integers

G (x, y): x is greater than y

“for any given positive integer, there is a greater positive integer”

a) $\forall_x \exists_y G(x, y)$

b) $\exists_y \forall_x G(x, y)$

c) $\forall_y \exists_x G(x, y)$

d) $\exists_x \forall_y G(x, y)$

Q let in a set of all humans

L (x, y): x likes y

"there is someone, whom no one like"

a) $\forall_x \exists_y \{\neg L(x, y)\}$

b) $\{\neg \forall_x \exists_y L(x, y)\}$

c) $\neg \{\forall_y \exists_x L(x, y)\}$

d) $\neg \{\exists_y \forall_x L(x, y)\}$

Q The CORRECT formula for the sentence, **"not all rainy days are cold"** is

a) $\forall_d$ (Rainy(d) $\wedge$ ~Cold(d))

b) $\forall_d$ (~Rainy(d) $\rightarrow$ Cold(d))

c) $\exists_d$ (~Rainy(d) $\rightarrow$ Cold(d))

d) $\exists_d$ (Rainy(d) $\wedge$ ~Cold(d))

Q Consider the statement

"Not all that glitters is gold"

Predicate glitters(x) is true if x glitters and predicate gold(x) is true if x is gold. Which one of the following logical formulae represents the above statement?

a) $\forall_x$: glitters(x) $\Rightarrow$ $\neg$gold(x)

b) $\forall_x$: gold(x) $\Rightarrow$ glitters(x)

c) $\exists_x$: gold(x) $\wedge$ $\neg$glitters(x)

d) $\exists_x$: glitters(x) $\wedge$ $\neg$gold(x)

Q What is the logical translation of the following lowing statement?

"None of my friends are perfect."

A) $\exists_x(F(x) \wedge \neg P(x))$

B) $\exists_x(\neg F(x) \wedge P(x))$

C) $\exists_x(\neg F(x) \wedge \neg P(x))$

D) $\neg\exists_x(F(x) \wedge P(x))$

Q Consider the following well-formed formulae:

1) $\neg\forall x(P(x))$ **2)** $\neg\exists x(P(x))$ **3)** $\neg\exists x(\neg P(x))$ **4)** $\exists x(\neg P(x))$

Which of the above are equivalent?

a) I and III

b) I and IV

c) II and III

d) II and IV

Q Which one of the following is the most appropriate logical formula to represent the statement? **"Gold and silver ornaments are precious".** The following notations are used:

G(x): x is a gold ornament

S(x): x is a silver ornament

P(x): x is precious

(A) $\forall_x (P(x) \rightarrow (G(x) \wedge S(x)))$

(B) $\forall_x ((G(x) \wedge S(x)) \rightarrow P(x))$

(C) $\exists_x ((G(x) \wedge S(x)) \rightarrow P(x)$

(D) $\forall_x ((G(x) \vee S(x)) \rightarrow P(x))$

Q Which one of the first order predicate calculus statements given below correctly express the following English statement? **"Tigers and lions attack if they are hungry or threatened"**

a) $\forall_x[(tiger(x) \wedge lion(x)) \rightarrow (hungry(x) \vee threatened(x)) \rightarrow attacks(x)]$

b) $\forall_x[(tiger(x) \vee lion(x)) \rightarrow (hungry(x) \vee threatened(x)) \wedge attacks(x)]$

c) $\forall_x[(tiger(x) \vee lion(x)) \rightarrow attacks(x) \rightarrow (hungry(x) \vee threatened(x))]$

d) $\forall_x[(tiger(x) \vee lion(x)) \rightarrow (hungry(x) \vee threatened(x)) \rightarrow attacks(x)]$

Conjunctive Normal Form

- A formula is in conjunctive normal form (CNF) or clausal normal form if it is a conjunction of one or more clauses, where a clause is a disjunction of literals; otherwise put, it is an AND of ORs.

- **Procedure for converting Propositional logic to CNF**:

- **Example:** $A \Leftrightarrow (B \vee C)$ into CNF.

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(A \Rightarrow (B \vee C)) \wedge ((B \vee C) \Rightarrow A)$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:
   $(\neg A \vee B \vee C) \wedge (\neg(B \vee C) \vee A)$.

3. CNF requires ¬ to appear only in literals, so we “move ¬ inwards” by repeated application of the following equivalences

In the example, we require just one application of the last rule:

- (¬A ∨ B ∨ C) ∧ ((¬B ∧ ¬C) ∨ A).

4. Now we have a sentence containing nested ∧ and ∨ operators applied to literals. We apply the distributivity law, distributing ∨ over ∧ wherever possible.

- (¬A ∨ B ∨ C) ∧ (¬B ∨ A) ∧ (¬C ∨ A).

- The original sentence is now in CNF, as a conjunction of three clauses.

Rules of Inference

- Rules of inference that have been introduced in propositional logic as I told you can also be used in predicate logic. The only thing that makes the difference is the quantifiers.

- **Four different methods to deal with quantified sentences in first order logic:**

  - **Universal specialization / Universal instantiation.**
  - **Existential instantiation**
  - **Existential generalization**
  - **Universal generalization / universal introduction.**

Universal specialization / Universal instantiation

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences. As per UI, we can infer any sentence obtained by substituting a ground term for the variable.

- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from ∀ x P(x) for any object in the universe of discourse.

- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$

Example: "Every person like ice-cream”:

∀x P(x)

So we can infer that "John likes ice-cream“: P(c)

Existential Instantiation

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic. It can be applied only once to replace the existential sentence.

- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.

- It can be represented as:

$$\frac{\exists x\, P(x)}{P(c)}$$

Existential introduction

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

Universal Generalization

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$
- This rule can be used if we want to show that every element has a similar property. In this rule, x must not appear as a free variable.
- **Example:** Let's represent, P(c): "A byte contains 8 bits", so for ∀ x P(x) "All bytes contain 8 bits.", it will also be true.

Forward Chaining

- **Begins with what is known**: starts with basic facts or data.
- **Moves forward**: applies rules to these facts to derive new information.
- **Data-driven**: new conclusions are made as data is processed.
- **Adds knowledge**: builds upon existing information incrementally.
- Stops when no further deductions can be made or a goal is reached.

Example: A Rice Crop Management System

- **Initial Observation**: A farmer in Odisha notices that some rice plants are shorter than usual and have yellow leaves.
- **Rule 1**: If rice plants are short and have yellow leaves, they might lack nitrogen.
- **Rule 2**: If rice plants lack nitrogen, then using nitrogen-rich fertilizer may help.
- **Rule 3**: If the season is monsoon and the problem is nitrogen deficiency, then use urea fertilizer, which is suitable for wet conditions.

Forward Chaining Process:

1. **Observing Symptoms**: The farmer inputs into the system that the rice plants are short with yellow leaves.
2. **Applying Knowledge**: The system uses Rule 1 to deduce that the plants might be suffering from nitrogen deficiency.
3. **Taking Action**: Based on Rule 2, the system suggests that applying a nitrogen-rich fertilizer could be beneficial.
4. **Considering Conditions**: Seeing that it is the monsoon season, Rule 3 advises the farmer to use urea fertilizer specifically.
5. **Outcome**: The farmer follows the advice, applies urea, and over time the rice plants become healthier and grow taller.

A

B

$A \wedge B \longrightarrow L$

$A \wedge P \longrightarrow L$

$B \wedge L \longrightarrow M$

$L \wedge M \longrightarrow P$

$P \longrightarrow Q$

Backward Chaining

- **Starts with a goal**: looks at the desired outcome first.
- **Works backward**: checks if current knowledge supports the goal.
- **Goal-driven**: each step is aimed at proving the end objective.
- **Deduces necessities**: determines what conditions must be met for the goal.
- Stops when it reaches the beginning facts or rules, or the goal is proven or disproven.

Example: Deciding on a Postgraduate Course

- **Goal**: A student wants to determine the best postgraduate course to enroll in.
- **Question**: What course should the student choose for a successful career?
- **Check Desired Outcome**: The desired outcome is a fulfilling and prosperous career in a field the student is passionate about.
- **Compare Interests**: The student is interested in technology and innovation.
- **Search for Career Paths**: The system considers various career paths that align with technology and innovation.

- **Hypothesis 1**: Is a Master's in Computer Science suitable?
- **Check for Evidence**: The system checks the student's academic background, skills, and job market trends for computer science graduates.

- **Hypothesis 2**: Could an MBA in Technology Management be an alternative?

- **Check for Evidence**: The system evaluates the student's leadership potential, market demand for technology managers, and the student's long-term career goals.

- **Solution if Matched**: If the evidence strongly supports one path over the other based on the student's profile and job market trends, the system recommends that course.

- **Outcome**: The student chooses the recommended course which aligns with their interests, background, and market demand, leading to a more focused educational path towards a desired career.

A

B

$A \wedge B \longrightarrow L$

$A \wedge P \longrightarrow L$

$B \wedge L \longrightarrow M$

$L \wedge M \longrightarrow P$

$P \longrightarrow Q$

Forward Chaining	Backward Chaining
1. Data-driven approach: begins with known facts.	1. Goal-driven approach: starts with the desired conclusion.
2. Bottom-up reasoning: builds up from facts to a conclusion.	2. Top-down reasoning: works backwards from goal to facts.
3. Uses Modus Ponens: applies rules to derive new information.	3. Uses Modus Tollens: checks if conclusions can lead to the goal.
4. Continues until no new information can be derived.	4. Continues until it reaches the starting facts or rules.
5. Common in real-time systems and production rule systems.	5. Used in diagnostic systems where the end goal is known.

Resolution

- Resolution in AI is a logical inference rule used to deduce new information or solve problems by finding contradictions.
- It involves converting logical statements into a standardized form called clausal form, which is a set of normalized logical expressions.
- Contradictory pairs of clauses are identified, and through a process of unification and combination, new clauses are generated.
- This process is repeated iteratively until either a contradiction is found (represented by an empty clause) or no new information can be deduced.
- Resolution is particularly powerful in automated theorem proving and logic programming, such as in Prolog, where it's used to infer conclusions from given facts and rules.

- All birds fly: $\forall x$ (Bird(x) $\rightarrow$ Fly(x))
- Tweety is a bird: Bird(Tweety)

We want to prove that Tweety can fly. Using resolution:

1. We negate the query ¬Fly(Tweety) and add it to our knowledge base.
2. Our clauses in clausal form are:
   - ¬Bird(x) $\vee$ Fly(x) (from $\forall x$ (Bird(x) $\rightarrow$ Fly(x)))
   - Bird(Tweety)
   - ¬Fly(Tweety) (negated query)
3. Now we apply resolution:
   - We resolve ¬Fly(Tweety) with ¬Bird(x) $\vee$ Fly(x) using unification by substituting x with Tweety.
   - This gives us ¬Bird(Tweety) $\vee$ Fly(Tweety).
4. Finally, we resolve Bird(Tweety) with ¬Bird(Tweety) $\vee$ Fly(Tweety), which gives us Fly(Tweety), the desired fact.

Since we started with ¬Fly(Tweety) and ended up with Fly(Tweety), we have derived a contradiction, meaning our original negation of the query must be false, thus proving that, according to our knowledge base, Tweety can fly.

Knowledge Representation

- **Definition**: It's the study of how knowledge can be represented in a computer system to mimic human thought and reasoning.
- **Components**:
  - Facts: True information relevant to a specific domain.
  - Representation: How facts are formally expressed, often using logical structures.
- **Example**: For the fact "Charlie is a dog," a logical representation could be Dog(Charlie).
- **Characteristics**:
  - **Syntax and semantics**: Clearly defined rules for structure and meaning.
  - **Expressive capacity**: Ability to convey all necessary domain knowledge.
  - **Efficiency**: Supports effective computing and reasoning.

Example Scenario: Knowledge Representation of a Library System

Fact:

"Book X is available in the library."

Representation in Logical Structure:

Using a logical statement, this fact can be represented as:

`Available(Book X)`

Characteristics Applied:

- **Syntax and Semantics**: The function `Available()` clearly defines that it checks for the availability of a specific book.
- **Expressive Capacity**: This structure allows the system to represent the availability of any book in the library by substituting "Book X" with any book's name.
- **Efficiency**: Such representation supports quick queries about book availability, facilitating efficient information retrieval and decision-making processes in the library's computer system.

Knowledge Acquisition

- **Definition**:
  - Involves gathering expertise for use in expert systems, organizing it into structured formats like IF-THEN rules.
  - Also refers to the process of absorbing information into memory, focusing on effective retrieval.

- **Procedure**:
  - **Identification**: Break down complex issues into parts.
  - **Conceptualization**: Define key concepts.
  - **Formalization**: Organize knowledge formally for programmatic use.
  - **Implementation**: Code the structured knowledge.
  - **Testing**: Check and validate the system's functionality.

To create an expert system that helps diagnose skin diseases based on symptoms and patient history.

Knowledge Acquisition Process:

1. **Identification**:
   - Break down the domain of skin diseases into manageable categories, such as infectious, non-infectious, inflammatory, and allergic skin conditions.
2. **Conceptualization**:
   - Define key concepts and terms, such as "eczema," "psoriasis," and "dermatitis." Understand the common symptoms associated with each category, like redness, itching, or peeling.
3. **Formalization**:
   - Organize the knowledge into a structured format:
     - IF the patient exhibits symptoms X, Y, and Z, AND has a history of A,
     - THEN the likely diagnosis is B.
   - Example rule: IF the patient has red, itchy patches, AND has a family history of allergies, THEN consider "eczema" as a diagnosis.
4. **Implementation**:
   - Code the knowledge into the expert system using a suitable programming language, integrating decision-making logic based on the IF-THEN rules.
5. **Testing**:
   - Validate the system's functionality by inputting test cases (e.g., symptoms of known diseases) to see if the expert system correctly diagnoses them based on the programmed knowledge.

- In artificial intelligence (AI), knowledge representation is a critical area that enables systems to simulate human-like reasoning and decision-making. Various techniques are employed to represent knowledge in AI systems, each with specific applications and advantages. Here are some key techniques:

- **Logic-Based Representation**:
  - **Propositional Logic**: Uses simple statements that are either true or false.
  - **Predicate Logic**: Extends propositional logic with predicates that can express relations among objects and quantifiers to handle multiple entities.
  - **Example**: Representing the relationship, "All humans are mortal," can be written in predicate logic as ∀x (Human(x) → Mortal(x)).

- **Semantic Networks**:
  - Graph structures used to represent semantic relations between concepts. Nodes represent objects or concepts, and edges represent the relations between them.
  - Example: A node for "Socrates" linked by an "is a" edge to a "Human" node, and "Human" linked to "Mortal".

- **Frame-Based Representation**:
  - Uses data structures called frames, which are similar to objects in object-oriented languages. Frames allow the grouping of related properties and actions.
  - Example: A frame for "Bird" might include properties like "has feathers," "can fly," and actions like "lay eggs."

- **Production Rules**:
  - Consist of sets of rules in the form of IF-THEN constructs that are used to derive conclusions from given data.
  - Example: IF the patient has a fever and rash, THEN consider a diagnosis of measles.

- **Ontologies**:
  - Formal representations of a set of concepts within a domain and the relationships between those concepts. They are used to reason about the entities within that domain and are often employed in semantic web applications.
  - Example: In a medical ontology, concepts like "symptom," "disease," and "treatment" might be related in ways that define what symptoms are commonly associated with a disease.

- **Bayesian Networks**:
  - Probabilistic models that represent a set of variables and their conditional dependencies via a directed acyclic graph (DAG). These are useful for handling uncertainty in AI applications.
  - Example: A network might represent the probabilistic relationships between diseases and symptoms, allowing the system to infer disease probabilities given observed symptoms.

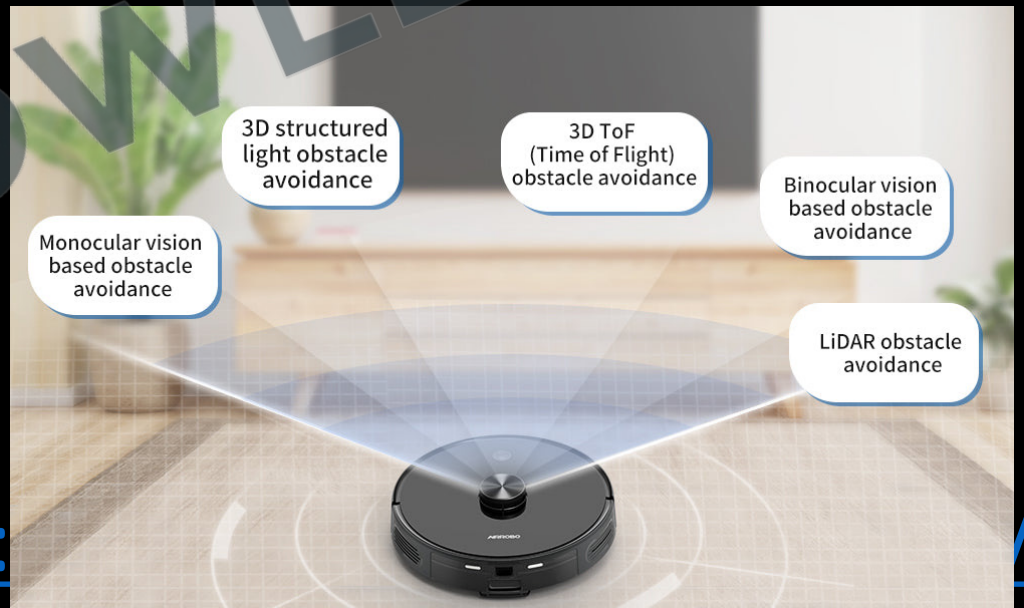
- **Neural Networks**:
  - Although typically classified under machine learning, neural networks can represent complex relationships between inputs and outputs through learned weights and can effectively capture and model knowledge after training.
  - Example: A neural network trained on historical weather data might predict future weather conditions.

- **<u>Chapter-4</u> (**SOFTWARE AGENTS**):** Architecture for Intelligent Agents - Agent communication - Negotiation and Bargaining - Argumentation among Agents - Trust and Reputation in Multi-agent systems.

Intelligent agents

Intelligent agents in the context of artificial intelligence (AI) refer to autonomous entities that observe through sensors and act upon an environment using actuators (i.e., they perform actions) to achieve specified goals autonomously. Here's a more detailed breakdown:

- **Autonomy**: Intelligent agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
  - **Example**: A Roomba vacuum cleaner navigates and cleans a room by itself, adjusting its path when encountering obstacles.

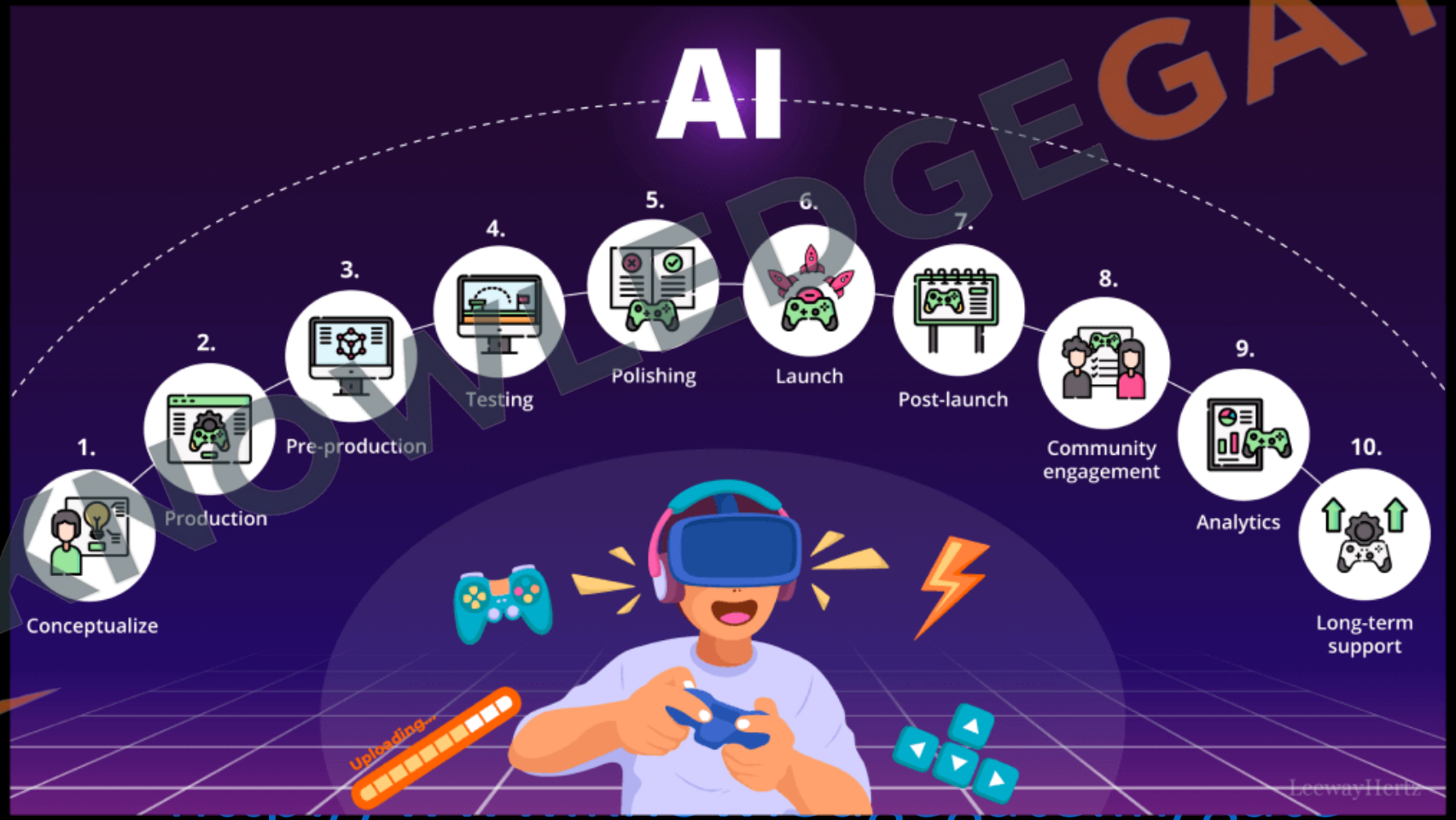


http: /gate

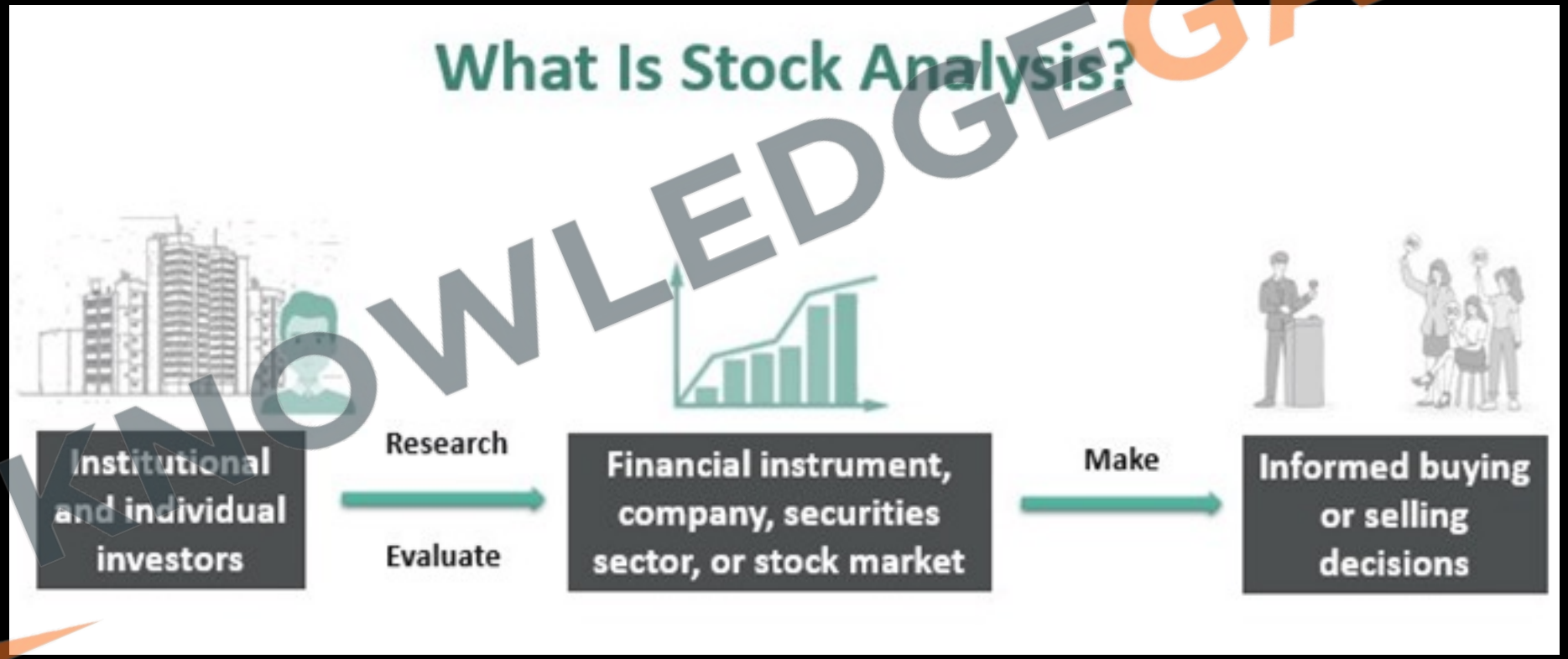
- **Sensory Capability**: They perceive their environment through sensory organs or data input mechanisms.
  - **Example**: A smart thermostat like Nest uses sensors to detect the temperature in a room and the presence of people to adjust heating and cooling settings.

- **Actuation**: Intelligent agents can act upon the environment using effectors (physical or virtual actuators) to change its state or the agent's own state to achieve goals.
  - **Example**: A chatbot on a customer service website interacts with users, providing answers to questions and guiding them through troubleshooting steps.

- **Goal-Oriented Behavior**: They are designed with specific objectives in mind and can make decisions to fulfill those objectives.
  - **Example**: An AI in a strategy video game seeks to win against human players by managing resources and deploying units strategically.



- **Rationality**: An intelligent agent acts to achieve the best outcome, or when there is uncertainty, the best expected outcome.
  - **Example**: An investment bot analyses stock market trends and makes buying or selling decisions to maximize investor returns.



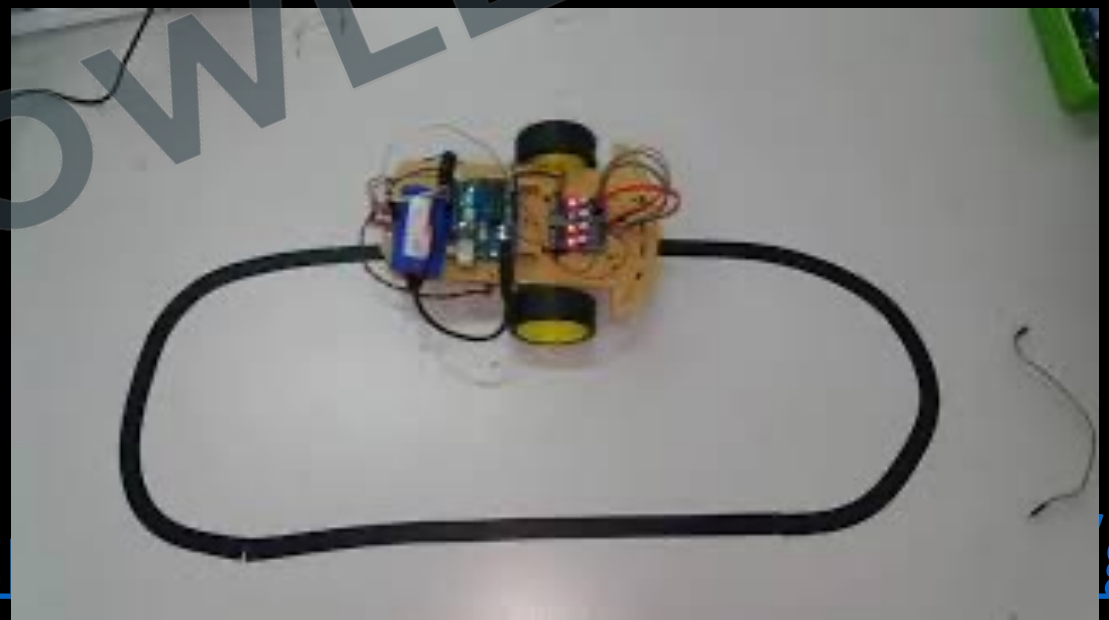
- **Learning and Adaptation**: Many intelligent agents have the ability to learn from their environment and experiences to improve their performance over time.
  - **Example**: Spotify's recommendation system learns from the user's listening habits to suggest new songs and playlists that match their tastes.

- **Interaction**: They may also be capable of interacting with other agents, including both intelligent agents and humans, in order to complete tasks or to simulate human interaction.
  - **Example**: Siri, Apple's virtual assistant, interacts with users to understand and execute a variety of tasks such as setting reminders or sending messages.

- Intelligent agents can range from simple, rule-based systems that perform automated tasks, to complex, learning and adapting systems with advanced features like natural language processing, problem-solving, and even forms of consciousness or self-awareness. They are applied in various domains such as virtual personal assistants, robots in manufacturing, autonomous vehicles, and software agents in complex simulation systems.

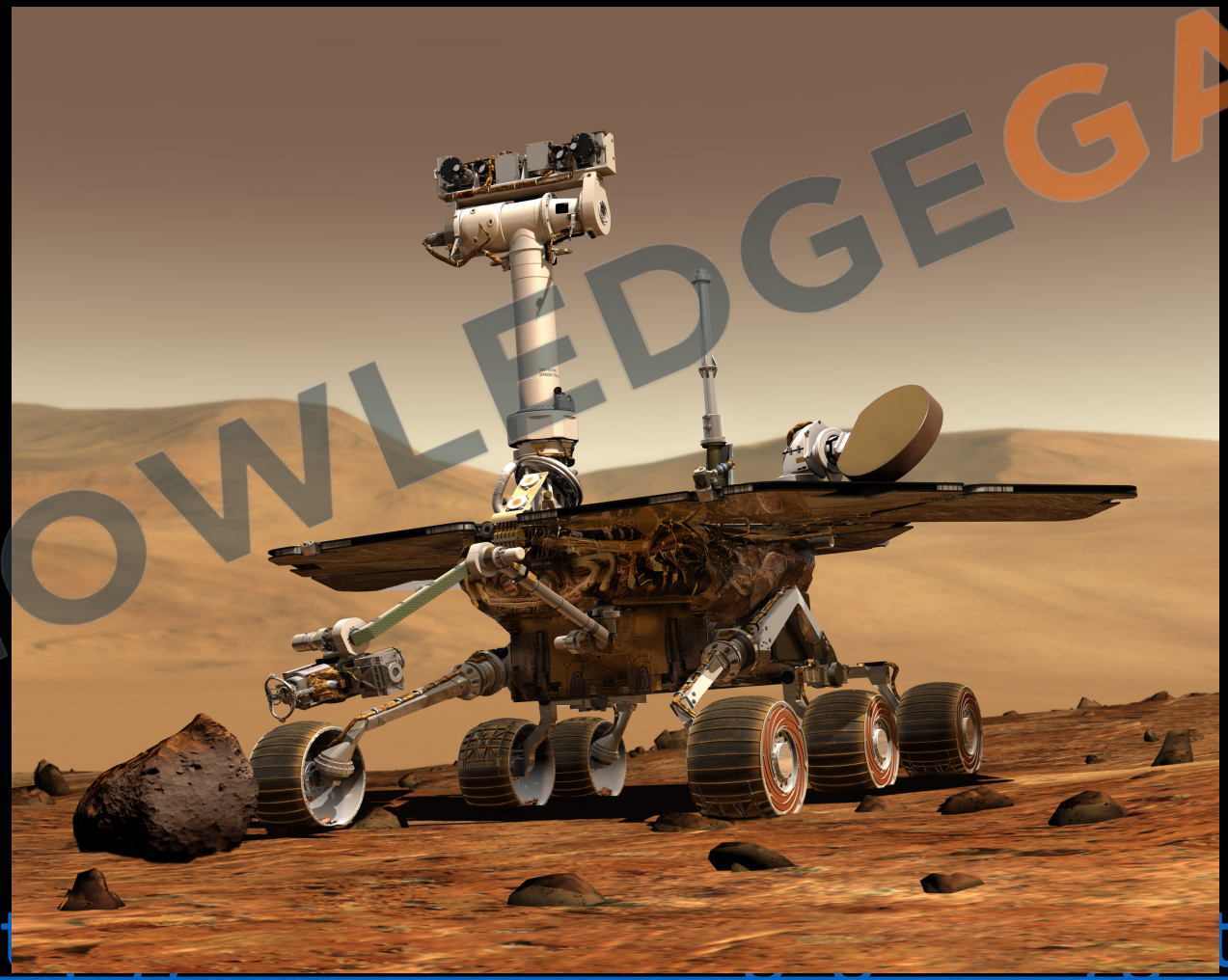
Architecture for intelligent agents

- The architecture for intelligent agents broadly refers to the underlying structure or model that defines how the agents perceive their environment, make decisions, and take actions. There are several types of architectures, each with its own approach to processing and responding to stimuli:
- **Reactive Architecture**:
  - Example: A line-following robot that adjusts its path based on real-time sensor input without an internal representation of the environment.

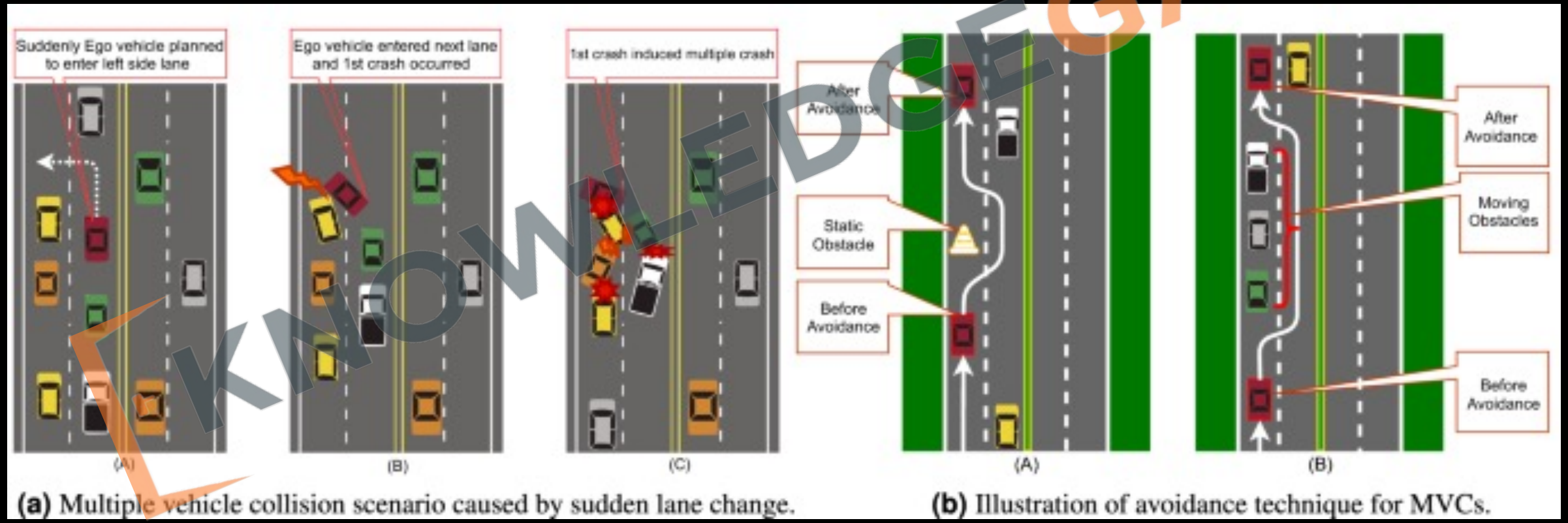


htt gate

- Deliberative/Reasoning Architecture:
  - **Example**: NASA's Mars rovers, which contain a world model and use planning and reasoning to decide on actions like navigation and experimentation.

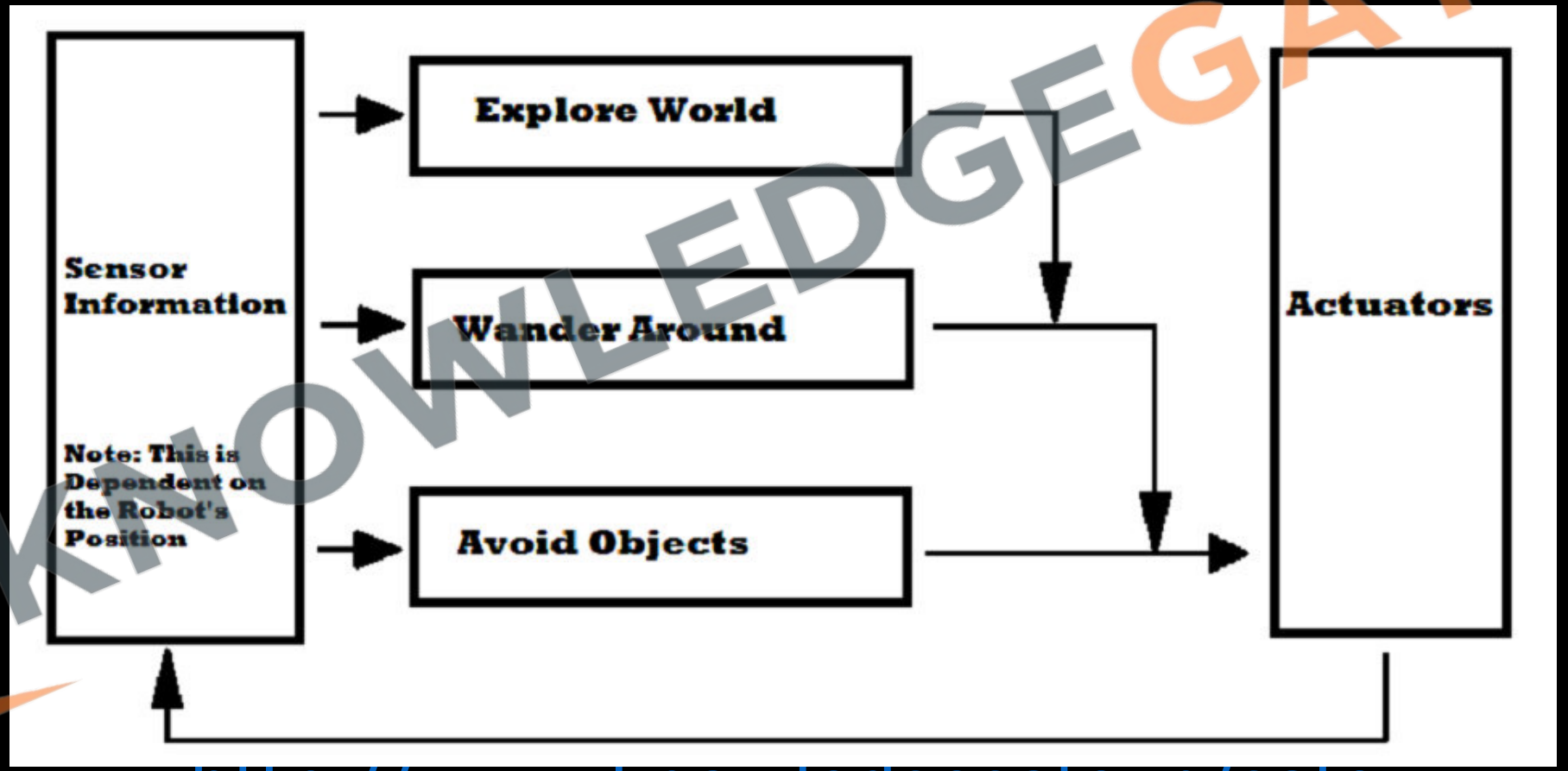


- Hybrid Architecture:
  - **Example**: Autonomous vehicles that combine reactive systems for collision avoidance with deliberative systems for route planning and navigation.



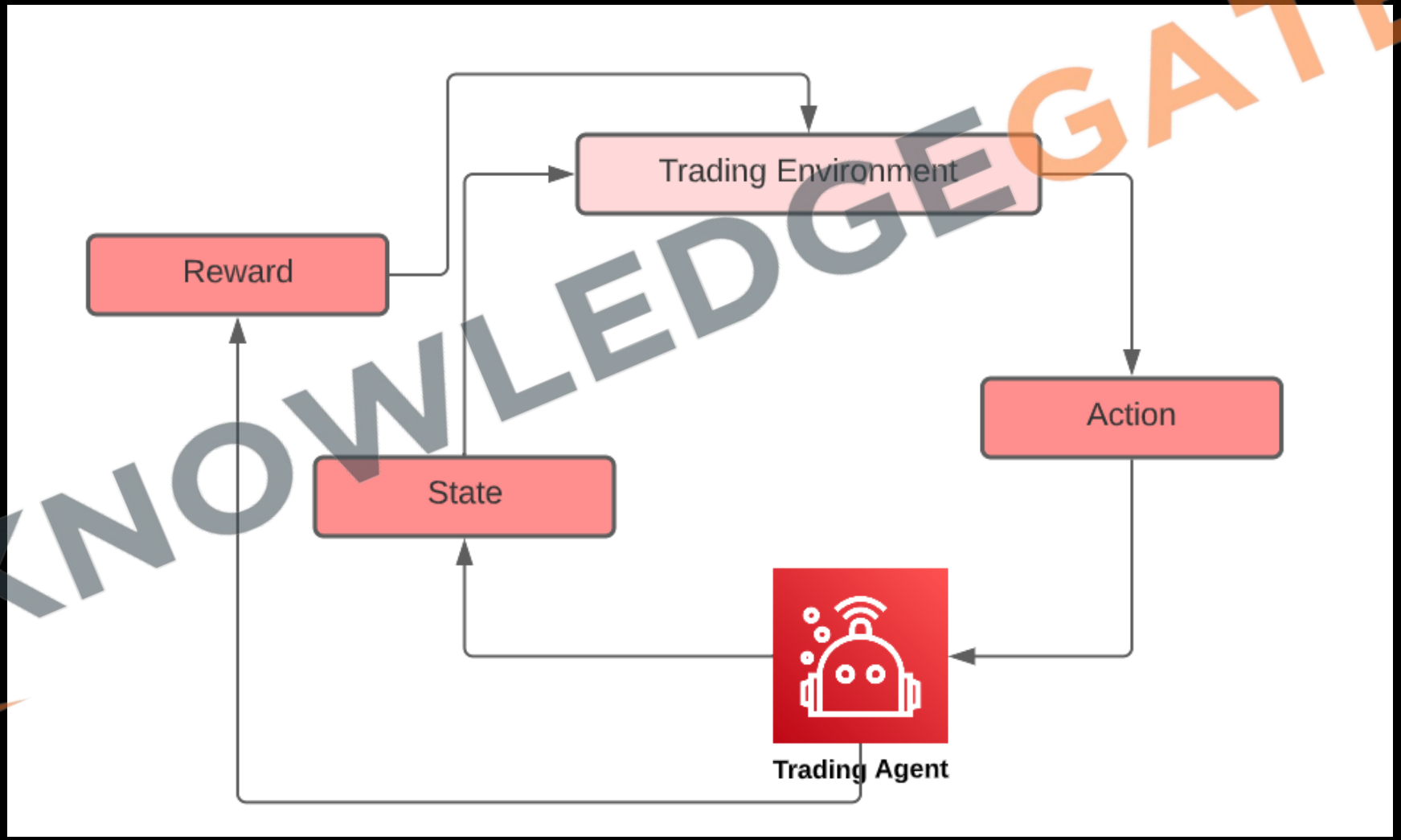
(a) Multiple vehicle collision scenario caused by sudden lane change. **(b)** Illustration of avoidance technique for MVCs.

- Layered Architecture:
  - **Example**: The subsumption architecture developed by Rodney Brooks, where a robot's behavior is controlled by a set of hierarchical layers that build upon reflexive actions up to complex behaviors.

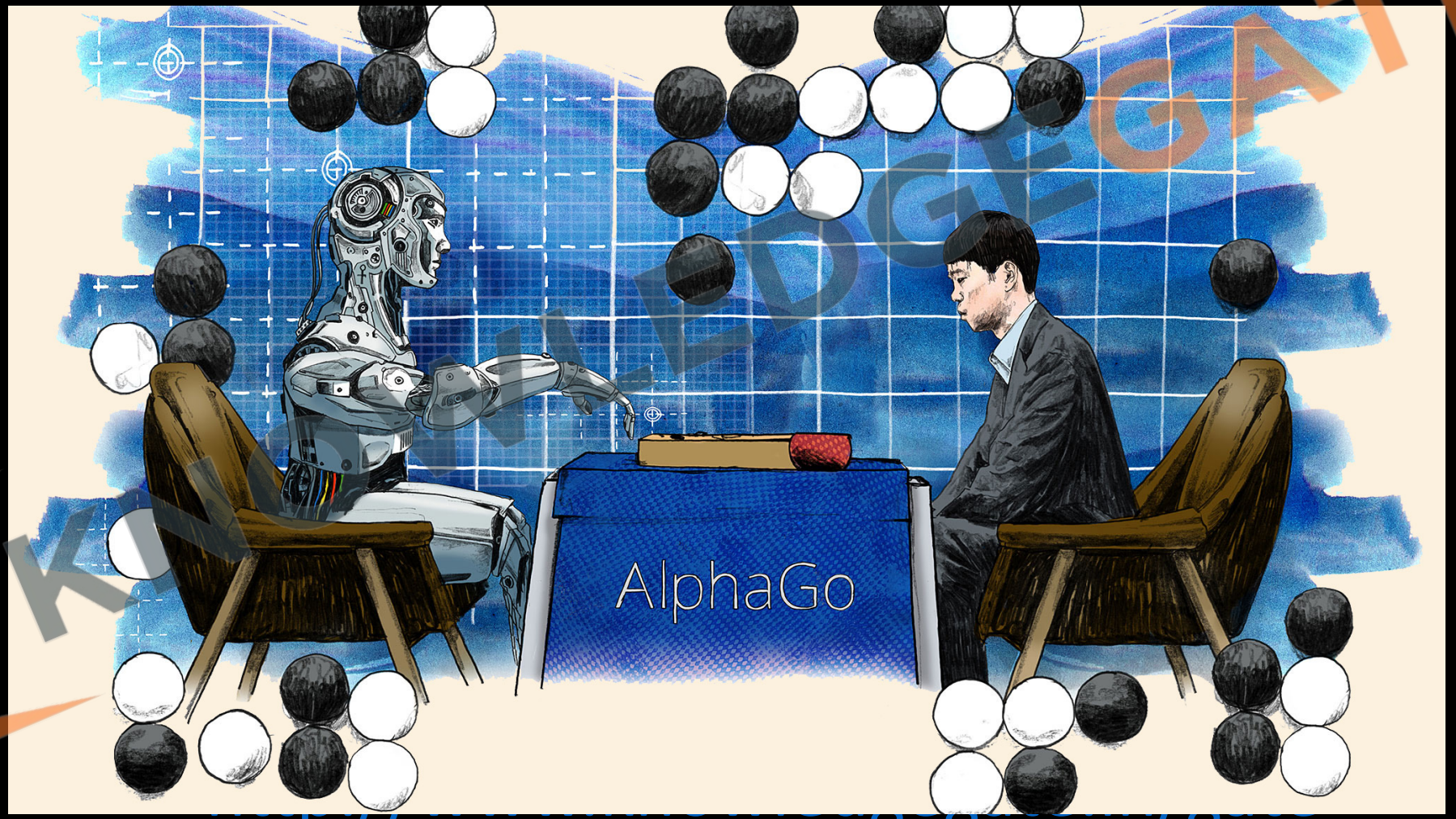


- Behavior-Based Architecture:
  - **Example**: Sony's AIBO, a robotic pet dog that exhibits behaviors based on a combination of internal states and sensory input.

- Agent-Oriented Programming:
  - **Example**: Multi-agent systems used for automated trading in stock markets where each agent represents a buyer or seller with specific strategies and goals.



- Learning Architecture:
  - **Example**: DeepMind's AlphaGo, which uses deep neural networks and reinforcement learning to improve its play in the game of Go.



Agent communication - Negotiation and Bargaining

- **Multi-Agent Interaction**:
  - **Theory**: In multi-agent systems, negotiation is a type of interaction among agents who aim to reconcile their different goals.
  - **Example**: Consider an e-commerce website where buyer and seller agents negotiate the price of goods. Each agent has the goal to buy low and sell high, respectively.

- **Resource Allocation Challenge**:
  - **Theory**: Negotiation in multi-agent systems often involves the allocation of scarce resources which can't be simultaneously satisfied for all competing agents.
  - **Example**: In network bandwidth allocation, multiple streaming services negotiate for bandwidth to provide a smooth service to users during peak hours.

- **Joint Decision Process**:
  - **Theory**: Negotiation and bargaining allow agents to reach a joint decision that meets each agent's objectives to some extent.
  - **Example**: In automated traffic management, different car agents negotiate the right of way at an intersection to minimize overall waiting time and avoid collisions.

- Main Features of Negotiation:
  - i. **Communication Language**: Agents use a predefined language for negotiations.
    - Example: Automated stock trading bots use Financial Information eXchange (FIX) language to place orders.
  - ii. **Negotiation Protocol**: Agents follow a set of rules during negotiation.
    - Example: Real estate agents use a standardized offer/counter-offer process when negotiating property sales.
  - iii. **Decision-Making**: Agents employ strategies to decide their stance in negotiations.
    - Example: A hotel booking agent uses past pricing data and current demand to decide on discounts for room rates.

- Attributes of Effective Negotiation:
  - i. **Efficiency**: Agents should not expend unnecessary resources in negotiations.
    - Example: Energy management systems in smart grids where agents representing households and utility companies negotiate power distribution efficiently.
  - ii. **Stability**: Agents have no reason to deviate from an agreed-upon deal.
    - Example: In international trade agreements, countries adhere to tariffs and quotas they've negotiated as deviation can lead to sanctions or trade wars.
  - iii. **Simplicity**: The negotiation mechanism should be straightforward and require minimal resources.
    - Example: Contactless payment systems negotiate transaction details with a simple tap, minimizing time and processing requirements.
  - iv. **Distribution**: The system functions without central control.
    - Example: Cryptocurrency trades where the negotiation mechanism is embedded in the decentralized blockchain protocol.
  - v. **Symmetry**: The mechanism does not favor any particular agent.
    - Example: Peer-to-peer file-sharing networks where each user's data is treated equally without any user having privileged access.

Arguments in Multi-Agent Systems

- **Argumentation as Communication**:
  - Just like people debate, AI agents discuss to convince others of their ideas.
  - **Example**: Consider two AI systems managing Delhi traffic. One argues for increasing green light time on a busy road to reduce jams, while the other argues against it to prevent congestion on cross streets.

- **Factors in Argumentation**:
  - Arguments have a structure like in debates, and they use logic. Agents also try to persuade others, like a friend convincing you to try a new restaurant.
  - Example: An AI system in a shopping app may argue for cheaper product options based on your purchase history, trying to persuade you that you'll like them.

- **Reasoning and Justification**:
  - Agents give reasons to back up what they say, like when you explain why you think it's going to rain by pointing to the dark clouds.
  - Example: A weather forecasting system in Mumbai uses data (reasons) to justify its prediction (conclusion) of a heavy monsoon.

- Types of Arguments:
  - a. **Informational Arguments**: Using facts to come to a new belief.
  - Example: An app says, "If the air quality index is high (fact), then it's not a good day for outdoor sports (new belief)."

  - b. **Motivational Arguments**: Using beliefs and wants to form a new want.
  - Example: If you believe it's hot outside and you want to stay cool, an AI assistant might suggest, "You might want to drink cold nimbu pani."

  - c. **Practical Arguments**: Using facts to decide what to do.
  - Example: "If the local cricket match is today (fact) and you want to watch it (subgoal), then you should finish work early (goal)."

  - d. **Social Arguments**: Considering social rules to decide what to do.
  - Example: "I can't play loud music late at night because my neighbors will be disturbed (social commitment)."

- **Interactions Between Arguments**:
  - Sometimes arguments go against each other, like when siblings argue over who gets to use the computer.
  - Example: In a village council meeting, one member's argument to use funds for a new well (support1 → conclusion1) might be opposed by another member who argues for new farm equipment (support2 → not conclusion1).

- By using these types of arguments, AI agents can discuss and make decisions just like we do in our daily lives, from choosing the quickest route to work to deciding the best way to spend a budget.

Trust and reputation

- Trust and reputation, along with belief maintenance, play critical roles in the dynamics of multi-agent systems, ensuring reliable interactions and stability within agent societies. Here's how they are intertwined:

- **Truth Maintenance System (TMS)**:
  - A TMS is used to maintain the integrity and consistency of the agents' beliefs. It helps in keeping the knowledge base stable and logically coherent.
  - Example: An AI system managing traffic signals in New Delhi uses a TMS to update its beliefs about traffic flow. If new data contradicts previous beliefs (like a sudden road closure), the system adjusts its knowledge base to maintain consistency.

- **Stable Knowledge Base**:
  - A stable knowledge base is one where all accepted data are backed by valid justifications, and any data without support are rejected.
  - Example: An e-commerce recommendation system believes a user likes a product based on their browsing history (valid justification) and disregards random items that the user never interacts with (lack of justification).

- **Well-Founded and Logical Consistency**:
  - The system avoids circular dependencies among beliefs and ensures that no contradictions exist within the knowledge base.
  - Example: A weather forecasting AI for a city like Chennai won't simultaneously hold the belief that it is both the dry season and the monsoon season.

- **Consistency and Completeness**:
  - The knowledge base should not contain contradictory beliefs, and it should be complete, concise, accurate, and efficient.
  - Example: A customer support AI contains complete information about product issues but doesn't hold conflicting information, such as a product being both available and discontinued.

- **Chapter-5 (**APPLICATIONS**):** AI applications - Language Models - Information Retrieval - Information Extraction - Natural Language Processing - Machine Translation - Speech Recognition - Robot - Hardware – Perception - Planning – Moving.

AI applications

- **Gaming**: In India, AI is being utilized to develop sophisticated mobile and online games, with companies like MPL (Mobile Premier League) using AI for enhancing user engagement and improving gaming strategies.
- **Natural Language Processing (NLP)**: Startups like Haptik and Vernacular.ai are harnessing NLP to create chatbots and virtual assistants that understand and respond in multiple Indian languages, improving customer service experiences.
- **Expert Systems**: Agriculture is a major sector in India, and AI-based expert systems are being developed to help farmers with crop selection, pest control, and disease diagnosis, improving yields and sustainability.
- **Vision Systems**: India's traffic and public safety departments are using AI-driven vision systems for surveillance, crowd management, and traffic control, particularly in smart city projects.
- **Speech Recognition**: Organizations like the Indian government's e-governance services are incorporating speech recognition to help citizens interact with services in a variety of Indian languages and dialects.
- **Handwriting Recognition**: Banks and financial institutions in India are employing handwriting recognition technologies to process cheques and forms faster and more accurately.
- **Intelligent Robots**: Healthcare in India is seeing the introduction of AI robots that assist in surgeries and patient care, while industries are using robots for tasks like assembly and warehouse management.

Language Models

- **State Machines**:
  - Think of these like a game of stepping stones where each stone is a state. You can move from one to another (transition) based on certain rules (inputs). Examples include:
    - Finite-State Automata (FSA): These are simple state machines that follow strict rules to move from state to state.
    - Finite-State Transducers (FST): These are like FSAs but can produce outputs, not just state transitions.

- **Rule Systems**:
  - These are sets of "if-then" rules. In linguistics:
    - Regular Grammars: Used for simple structures like validating email formats.
    - Context-Free Grammars: Good for describing the structure of sentences where context isn't needed.
    - Feature-Augmented Grammars: These add extra information to grammatical rules, like gender or number in languages.

- **Logic**:
  - Logic models are like precise language for computers. They help computers understand meanings and implications. Types include:
    - First-Order Logic: Deals with objects and their relationships.
    - Lambda Calculus: A way to express computations, used in functional programming languages and semantics.
    - Feature Structures: Like filling out a form, each feature (like 'color') has a value (like 'red').
    - Semantic Primitives: These are basic concepts used to build more complex meanings.

- **Probabilistic Models**:
  - These models are like weather forecasts for language, assigning probabilities to different linguistic events. Examples are:
    - Weighted Automata: State machines where each transition has a weight, like a cost or likelihood.
    - Markov Models: Predicts a sequence of events, where each event only depends on the previous one.
    - Hidden Markov Models (HMMs): Like Markov models, but the states are hidden and only the outcomes are seen.

- **Vector-Space Models**:
  - Imagine a giant space where each word is a point, and similar words are closer together. This is useful for tasks like finding synonyms or grouping similar words.

Information Retrievel

- Certainly, information retrieval (IR) is a vital component across various domains in computer science, particularly in areas where large datasets are involved. Here's a simplified explanation of your points:

- **Huge Data/Document Collection**:
  - Think of IR systems as a vast library. Just like in a physical library where books are categorized and indexed for easy retrieval, IR systems must manage and sift through vast amounts of digital information to find what the user is looking for.

- **Format of Query with Standard Query Language**:
  - When you want to find information, you ask a question or type a search. In some systems, especially databases, you use a specific language with a particular syntax, like SQL (Structured Query Language), to tell the system exactly what you're looking for.

- **The Generated Result Model**:
  - This is about how the system finds and picks the documents that answer your query. It's like having a smart assistant in the library who understands your question and knows exactly where the answers are in the countless shelves of books.

- **Displaying Results Model**:
  - Once your assistant finds the books, they can show them to you in different ways. Some systems just show you the titles or a brief summary; others give you more detailed previews. And then there are those that might use fancy visual aids, like maps or 3-D models, to present the information in an easy-to-understand manner.

Definitions and Scope of NLP:

- **Natural Language Processing (NLP)**: This field studies the challenges in processing and manipulating natural language, aiming to make computers understand human language.
- **Automatic Processing**: NLP enables the automatic processing of human languages, making it a subset of AI focused on human-computer communication in natural languages like English.

Purpose and Need for NLP:

- **Analyzing Text**: NLP is used to analyze and represent natural texts at various linguistic levels to achieve human-like language processing for diverse tasks.
- **Bridging Communication Gaps**: It serves as a bridge between human communication and machine understanding, facilitating more intuitive interactions.

Key Areas in Language Generation:

- **Content Determination**: Deciding what information should be conveyed in the response.
- **Text Planning**: Organizing the content into a coherent structure before generating the actual text.
- **Realization**: Transforming the structured content into natural language text.

Capabilities of a Full NLU System:

- **Paraphrasing**: Rewording the input text while retaining the original meaning.
- **Translation**: Converting text into another language.
- **Question Answering**: Providing answers based on the content of the text.
- **Inference**: Making logical deductions from the text.

Applications of Natural Language Processing

1. Machine Translation:

- **Purpose**: Helps translate texts like manuals, support content, or catalogs at reduced costs, enabling access to multilingual information.
- **Challenge**: The main difficulty lies not in translating words but in understanding and conveying the intended meanings of sentences.

2. Automatic Summarization:

- **Need**: Useful in managing information overload by condensing large volumes of data into concise summaries.
- **Applications**: Provides quick overviews of long articles, such as news stories or blog posts, and is useful in extracting sentiment and key points from social media.

- 3. Sentiment Analysis:
  - **Goal**: To detect and understand the sentiments expressed in text, even when these emotions are not explicitly stated.
  - **Use Cases**: Companies utilize this to gauge public opinion on products and services, helping them to tailor marketing strategies and improve customer experiences.

4. Text Classification:

- **Functionality**: Involves categorizing documents into predefined groups to streamline information retrieval and organization.
- **Examples**: Common applications include email spam filtering and sorting content into thematic folders.

5. Question Answering:

- **Overview**: Develops systems that can answer questions posed by humans in a clear and accurate manner.
- Technologies: This includes interfaces like chatbots and virtual assistants, which use NLP to provide timely and contextually relevant responses.

fundamental steps involved in Natural Language Processing (NLP)

1. Morphological Analysis

- **Purpose**: To break down words into their smallest meaningful units, known as morphemes (e.g., "comes" breaks down into "come" + "s").
- **Process**: Words are analyzed to identify their root forms and affixes. This step also involves distinguishing tokens, like punctuation, which are not part of any words.

2. Syntactic Analysis

- **Purpose**: To arrange words into a grammatical structure that makes sense according to the rules of the language.
- **Process**: This involves parsing sentences to identify grammatical structures, determining the role of each word (such as subject, verb, object), and how these words interrelate to form coherent sentences.

3. Semantic Analysis

- **Purpose**: To understand the meanings of individual words and how these meanings combine in sentences to produce overall sentence meanings.
- **Process**: Semantic analysis takes the structures formed during syntactic analysis and assigns meanings to them, ensuring the entire sentence makes sense semantically.

4. Pragmatic Analysis

- **Purpose**: To interpret sentences based on their usage in specific contexts, considering how meaning changes with different situations.
- **Process**: It goes beyond the literal meaning of sentences to consider factors like the speaker’s intent, the listener's perception, and the context in which the conversation occurs. This helps in determining what is meant from what is simply said.

5. Discourse Integration

- **Purpose**: To consider the influence of preceding sentences on the interpretation of the current sentence, and how the current sentence can affect subsequent sentences.
- **Process**: This step looks at the text as a whole, rather than in isolated sentences, ensuring that the interpretation of one sentence is consistent with and influenced by others around it.

AI significantly enhances NLP by allowing machines to understand and interact with human language in a natural and intuitive way. This involves several key capabilities and approaches:

- **Speech Recognition**: AI enables machines to process spoken words, turning speech into text, which is crucial for applications like voice-activated assistants.
- **Natural Communication**: AI programs communicate with humans using natural language, making interactions with machines more user-friendly and accessible.
- **Understanding Language**: AI must understand the structure of language, including vocabulary, syntax (arrangement of words), and semantics (meaning), to effectively process human language.

Approaches to NLU (Natural Language Understanding)

- **Keyword and Pattern Matching**: Simple methods for identifying essential words or patterns to interpret language.
- **Syntactic and Semantic Analysis**: More complex analyses that consider both the structure and meaning of language to generate responses.
- **Real-World Situation Matching**: AI compares inputs to known real-world scenarios to enhance understanding and response accuracy.

Machine Translation (MT)

- Refers to the automated process of translating text from one language (source language) to another (target language) using computer algorithms.
- **Definition**: Machine translation automates the translation of text from one language to another, converting sequences of symbols (words, phrases) from the source language to equivalent sequences in the target language.
- **Purpose**: The primary goal is to enable communication across language barriers without human translators, enhancing accessibility and efficiency in global interactions.

Types of Machine Translation Systems

- **Bilingual MT System**:
  - These systems focus on translating between two specific languages. They are typically unidirectional, meaning they translate from one language to another but not vice versa.
- **Multilingual MT System**:
  - Capable of translating between multiple language pairs. These systems can be either unidirectional or bidirectional, facilitating translations across several languages.

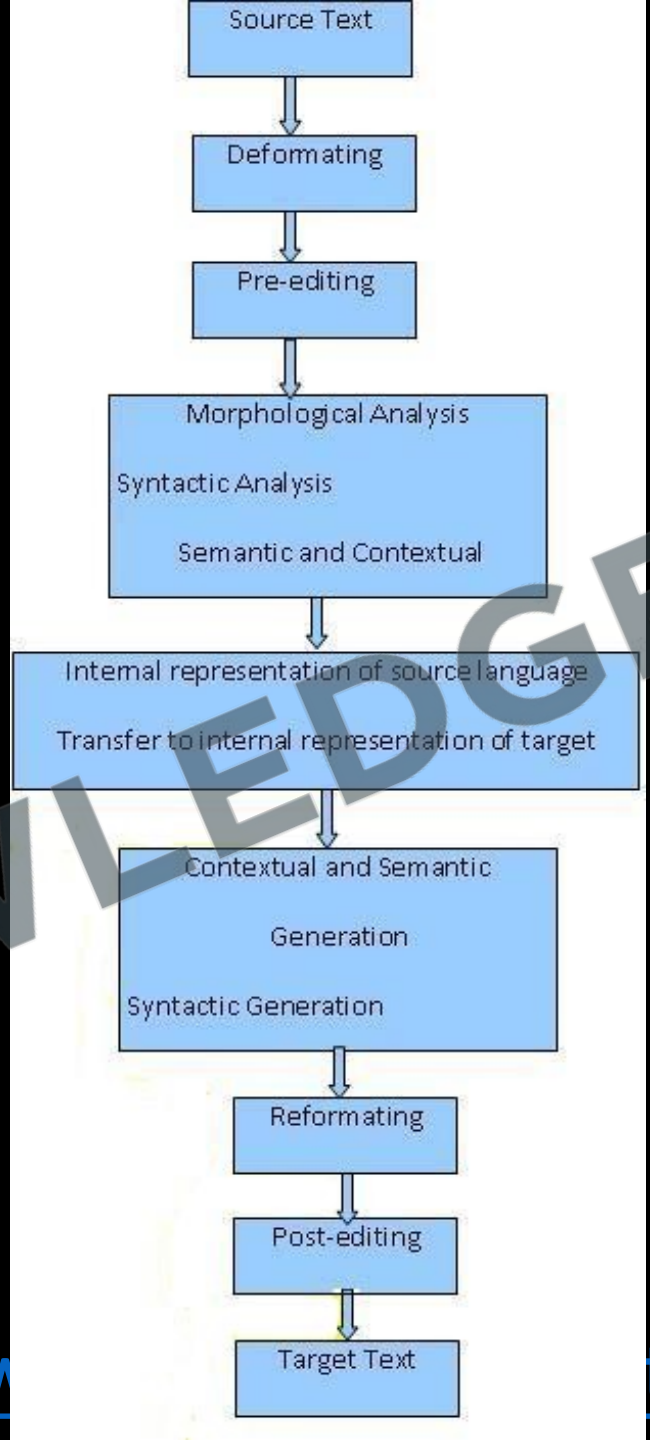
Major Approaches to Machine Translation

- **Direct MT Approach**:
  - **Description**: Direct translation from the source to the target language without intermediate steps.
  - **Example**: Translating "Buenos días" (Spanish) directly to "Good morning" (English) without analyzing or restructuring the sentence's grammatical components.

- **Interlingua Approach**:
  - **Description**: Translation involves converting the source language into an artificial intermediate language (Interlingua) and then into the target language.
  - **Example**: For translating "Bonjour" (French) to "Guten Morgen" (German), the system first translates French into an Interlingua representation like "Greeting(morning)" and then from Interlingua to the German equivalent.

- **Transfer Approach**:
  - **Description**: Involves three steps: analysis of the source text, conversion to a target-language representation, and generation of the target text.
  - **Example**: To translate "Estoy cansado" (Spanish, meaning "I am tired") to English, the approach would first analyze and understand the structure ("[subject] [verb] [adjective]"), map it to an equivalent English structure ("[subject] [verb] [adjective]"), and generate the target text ("I am tired").

- **Empirical (Statistical or Neural) MT Approach**:
  - **Description**: Uses statistical models or neural networks trained on large datasets of bilingual texts to predict translations.
  - **Example**: Google Translate uses neural machine translation (NMT). When translating "Je t'aime" from French to English, it analyzes vast amounts of French-English text data to learn that the most probable translation of "Je t'aime" is "I love you" based on statistical likelihood and context learned from data.



Overview of Speech Recognition

- Speech recognition technology allows computers to recognize spoken language and convert it into a digital format that machines can process. This capability can transform spoken commands or dictations into text or other forms of actionable data, depending on the application.

Commonly Used Algorithms for Speech Recognition:

- **Hidden Markov Models (HMM)**:
  - HMMs are statistical models that assume the probability of each state depends only on the previous state. In speech recognition, they are used to model sequences of speech units (like phonemes or words) and predict the most likely sequence from the audio input.
- **Neural Networks**:
  - Deep learning approaches use neural networks to model complex patterns in data. These networks consist of layers of interconnected nodes that mimic the human brain's structure and function. They process training data, learn to recognize speech patterns, and improve accuracy over time through methods like gradient descent.

Applications of Speech Recognition

- **Automotive**: Integration in vehicles for hands-free navigation, controlling media playback, and other voice-activated features enhance driver safety.
- **Technology**: Virtual assistants (like Siri, Google Assistant, Alexa, Cortana) embedded in smartphones and smart speakers help with tasks such as setting reminders, playing music, and controlling smart home devices through voice commands.
- **Healthcare**: Doctors and nurses use speech-to-text applications for efficient documentation of patient care, improving accuracy and saving time.
- **Sales and Customer Service**: Call centers use speech recognition to transcribe and analyze calls, helping identify trends and common issues. Cognitive bots engage in voice-based customer interactions, improving response times and service quality.
- **Security**: Voice authentication systems provide an additional layer of security, using unique vocal characteristics to verify a user's identity.

Techniques for Speech Recognition

- **Speech Analysis**: This involves analyzing the physical and behavioral characteristics of speech, such as pitch and tone, which can help differentiate speakers and understand spoken commands.
- **Feature Extraction**: Key to improving performance, this technique involves condensing speech into a set of meaningful data points that represent the essential characteristics of the speech signal without the redundancy.
- **Modeling**: Techniques like speaker recognition and identification are used to create models that can recognize and differentiate between different speakers based on their unique vocal features.

Robots, robotics, and how they intersect with artificial intelligence (AI)

- What are Robots?
  - Definition: Robots are artificial agents designed to perform tasks in the real world. They can manipulate objects through actions like picking, moving, modifying, or even destroying them to achieve specific goals, often in repetitive roles to free up human labor.
- What is Robotics?
  - Field Description: Robotics is an interdisciplinary branch that combines electrical engineering, mechanical engineering, and computer science to design, construct, and operate robots. It involves developing systems that can perform complex actions autonomously or semi-autonomously.

Differences Between AI Programs and Robot Systems

- **Environment**: AI programs typically operate in simulated virtual environments, while robots operate in the real, physical world.
- **Inputs**: AI programs often receive inputs as symbols and rules, whereas robots deal with analog signals such as sounds and images.
- **Hardware Needs**: AI programs run on general-purpose computers, while robots require specialized hardware like sensors and actuators to interact with their environment.

Integration of Robots and AI

- **Enhanced Capabilities**: AI equips robots with skills like computer vision, allowing them to navigate and interact dynamically with their environment. AI also enables robots to learn from experiences and make autonomous decisions, using various forms of machine learning.

Integration of Robots and AI

Components of a Robot

- **Power Supply**: Includes batteries, solar power, hydraulic, or pneumatic sources.
- **Actuators**: Convert energy into mechanical motion.
- **Sensors**: Equip robots with the ability to gather real-time data about their surroundings, crucial for tasks requiring environmental interaction.

Robot Locomotion

- **Legged Locomotion**: Uses multiple legs, suitable for varied terrain but power-intensive and complex due to the need for coordination and balance.
- **Wheeled Locomotion**: Involves fewer motors, generally easier to implement, and more stable and power-efficient compared to legged options.
- **Slip/Skid Locomotion**: Similar to a tank, using tracks that allow stability and maneuverability on challenging surfaces.

Mobile Robot Hardware Overview

- A mobile robot integrates hardware and computational components to operate autonomously or semi-autonomously in various environments. Key hardware subsystems of a mobile robot include:
- **Motion**: This subsystem allows the robot to change its position relative to its environment. It includes mechanisms like wheels, legs, or tracks that enable movement.
- **Sensing**: Sensing technologies enable a robot to gather information about its own state and its surroundings. This can include a range of sensors such as cameras, infrared sensors, ultrasonic sensors, and lidar.

- **Reasoning**: This involves processing sensor data to make decisions. The reasoning subsystem often includes onboard computers that execute algorithms for navigation, obstacle avoidance, and task execution.

- **Communication**: Robots often need to communicate with human operators or other machines. This subsystem can include wireless communication technologies like Wi-Fi, Bluetooth, or even satellite communications in more advanced robots.

Categories of Motion of Mobile Robots

- Mobile robots are designed to operate in various environments, each requiring specialized modes of locomotion:
- **Terrestrial**:
  - **Environment**: These robots navigate ground surfaces, leveraging the stability of solid support.
  - **Locomotion**: Terrestrial robots predominantly use wheels for simplicity and efficiency. Other forms include legged robots for walking or climbing, tracked robots for navigating rough terrains, and robots that slither, adapting to diverse ground conditions.

- **Airborne**:
  - **Environment**: Airborne robots are crafted to fly, often emulating the designs of conventional aircraft or avian species.
  - **Locomotion**: This category includes robotic helicopters, fixed-wing aircraft, and inventive forms like automated parachutes and dirigibles. Challenges for these robots include maintaining position and navigating through varying air conditions.

- **Aquatic**:
  - **Environment**: Aquatic robots function in water, suitable for both surface and underwater operations.
  - **Locomotion**: They mainly utilize propellers or jets for movement, essential for conducting tasks in locations that are otherwise inaccessible to humans, such as deep-sea exploration or maintaining underwater structures.

- **Space**:
  - **Environment**: Operating in the extreme conditions of outer space, including microgravity and intense radiation.
  - **Locomotion**: Space robots may operate independently as free-fliers or move across larger vehicles like space stations. Their movement typically relies on thrusters suited for the vacuum of space.

Types of AI in Robotics

- **Weak AI**: Often used for simulated human thought processes in controlled environments, such as virtual assistants (e.g., Siri, Alexa).
- **Strong AI**: Employs sophisticated decision-making capabilities without human oversight, used in autonomous vehicles and advanced robotics.
- **Specialized AI**: Tailored for specific tasks, commonly found in industrial robots performing repetitive actions like painting or assembly.

Applications of Speech Recognition in Robotics

- Used extensively in fields like automotive for voice-activated systems, healthcare for documentation, and customer service for interactive bots and security systems, enhancing both efficiency and safety.