AI-Series-1-Important-Topics

For more notes visit

https://rtpnotes.vercel.app

1. Agent types and structure

What is an Agent?

- Think of an "agent" as a robot or a computer program that can "see" what's happening around it and "do" things to change or respond to its surroundings.
    - **Operates in an environment**: The agent exists in a world or space, which could be a physical place (like a room for a robot) or a virtual space (like a game for an AI).
    - **Perceives its environment through sensors**: It gathers information about the environment using sensors. For example, a robot might use cameras to see, or a program might look at the data it's given.
    - **Acts upon its environment through actuators/effectors**: After processing the information it gets, the agent makes decisions and takes actions to change or interact with the environment. For example, a robot moves its arms, or a program changes the output on the screen.
    - **Has goals**: The agent is always trying to achieve some goals or objectives, like cleaning a room or winning a game.
- In simpler terms, an agent is like a smart helper that watches, thinks, and acts to accomplish something in the world it operates in.

Actuators and Effectors

Actuators and **effectors** are devices or components that allow an agent (like a robot or a system) to take action and interact with its environment.

- **Actuators**: These are the mechanisms that physically make something happen. In robots, for example, actuators are like muscles—they control movements such as turning wheels, moving arms, or opening grippers. Actuators convert electrical signals from the agent into physical actions, like moving, rotating, or manipulating objects.
- **Effectors**: These are the parts of the agent that cause the actual change in the environment. In a robot, effectors might be things like wheels, robotic arms, or grippers. **The actuators control the effectors**. For example, an actuator moves a robot’s arm, and the **arm** (the effector) is what interacts with the object or the environment.

So basically:

- **Actuators** make the motion or action happen.
- **Effectors** are the parts that directly interact with the environment to cause a result.

For example, in a drone:

- The **actuators** are the motors that spin the propellers.
- The **effectors** are the spinning propellers that lift the drone off the ground and move it around.

Intelligent Agents

An **intelligent agent** is like a smart, independent system that senses its surroundings, makes decisions, and takes actions to achieve a goal. It uses **sensors** to gather information and **actuators** to interact with the environment.

- **Autonomous**: It operates on its own without constant human guidance.
- **Goal-oriented**: It works toward specific goals, like keeping a room at the right temperature or navigating a car safely.
- **Learning**: An intelligent agent can learn from its environment and improve its decisions over time.

Example: A thermostat
A thermostat is an intelligent agent that senses the room temperature (using sensors) and turns the heater on or off (using actuators) to maintain the desired temperature.

Rational Agent

A **rational agent** is an agent that makes decisions with a clear goal in mind and tries to achieve the best possible outcome based on what it knows.

- **Clear Preferences**: A rational agent knows what it prefers or what it is trying to achieve (its goal).
- **Models Uncertainty**: It understands that it might not know everything, so it makes decisions based on the best possible information.
- **Maximizes Performance**: The agent aims to take actions that lead to the best results, according to a performance measure. For example, in a game, this could mean winning, and in real life, it could mean solving a problem efficiently.
- **Acts Rationally**: It always tries to take the best possible action to reach its goal, given the situation it is in.

Difference between Intelligent and Rational Agents

- **Learning**:
  - **Intelligent Agent**: Can learn from its experiences and improve over time.
  - **Rational Agent**: Focuses on acting in the best way based on current information, regardless of learning.
- **Decision-Making**:
  - **Intelligent Agent**: May adapt or try different actions, even if they're not always the best.
  - **Rational Agent**: Always takes the most logical and goal-focused action available.
- **Goal**:
  - **Intelligent Agent**: Seeks to achieve its goals through learning and adapting.
  - **Rational Agent**: Seeks to achieve its goals by maximizing performance in every situation.

Structure of an AI Agent

- The structure of an AI agent explains how an agent works by combining two key parts: the architecture and the agent program.

1. Architecture:

- This is the **hardware** or **platform** on which the agent runs. It could be a physical machine (like a robot) or a computer system. Think of it as the "body" of the agent that allows it to function.

2. Agent Function:

- The **agent function** is like the brain behind the agent's actions. It takes the **percept** (what the agent senses from the environment) and decides what action to take in response. It's represented as:
  - f: P -> A* (where "P" is the percept and "A" is the action).

3. Agent Program:

- The **agent program** is the actual **code or software** that makes the agent function work. It runs on the architecture and translates the agent's observations into actions.

Formula:

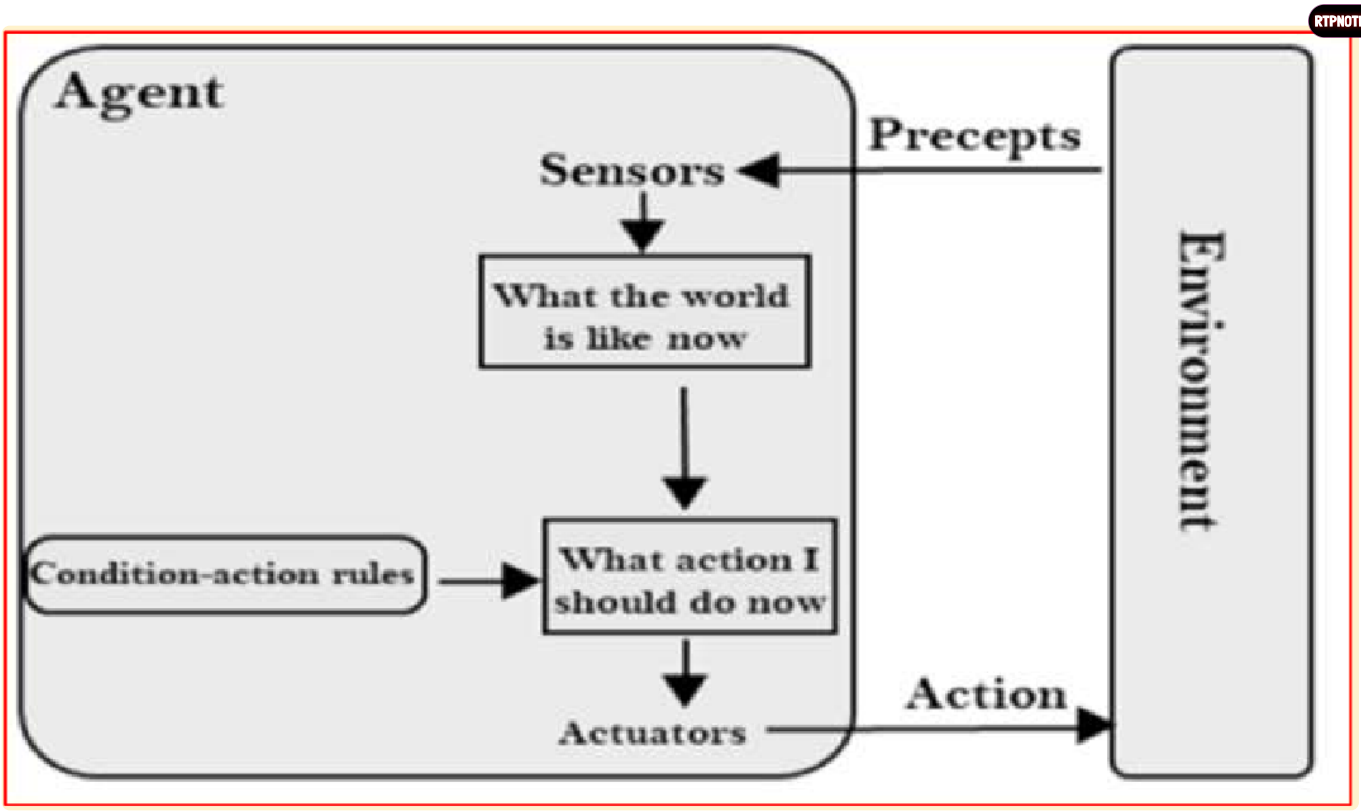
- **Agent = Architecture + Agent Program**: An AI agent is a combination of the physical structure (architecture) and the intelligent software (agent program).

Types of AI Agents

1. Simple Reflex Agent

A **Simple Reflex Agent** is a type of artificial agent that works by reacting to its current situation (called the "percept") without considering any history or future consequences.

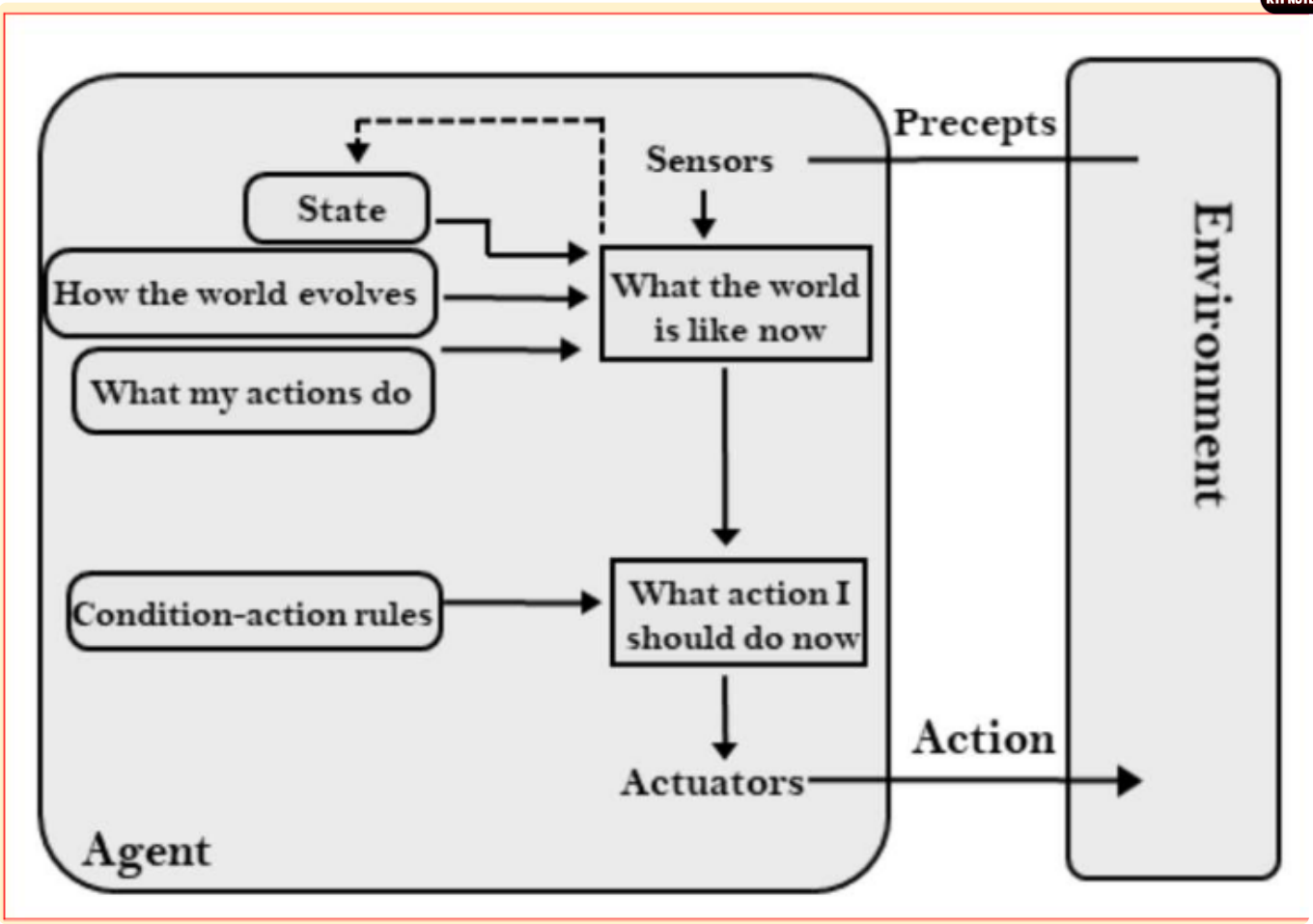
- **Choose actions only based on the current percept**: The agent decides what to do just by looking at the current situation it's in, without worrying about past events or future outcomes.
- **Rational only if a correct decision is made based on the current percept**: The agent can be called "smart" or "rational" only if it can make the right decision based solely on what it currently observes.
- **Their environment is completely observable**: The agent can see or sense everything it needs to know about its environment right now.
- **Condition-Action Rule**: The agent follows simple rules that connect what it sees (condition) to what it should do (action). For example, "If the light is red (condition), then stop (action)."



2. Model-based reflex agent

A Model-based Reflex Agent is a more advanced type of agent that can make better decisions by understanding the world and keeping track of changes

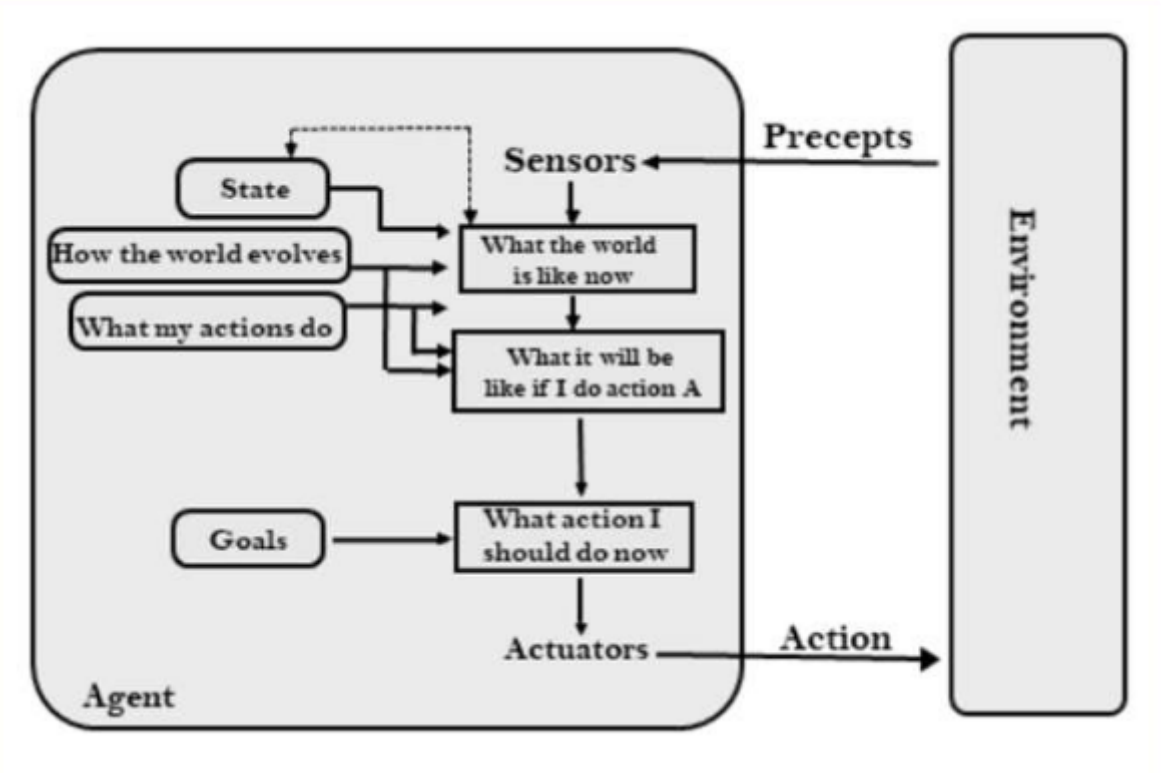
- **A model of the world to choose their actions**: Instead of just reacting to what it sees in the moment, this agent builds a mental picture (or "model") of how the world works to guide its actions. It also keeps track of what's happening, even if it can't see everything at once.
- **Model**: This is the agent's understanding of how things usually happen in the world. For example, it knows that if it drops something, it will fall.
- **Internal State**: The agent remembers things that it can’t currently see. This memory is built up from everything it has observed so far (called "percept history"). For example, if the agent has walked into another room, it remembers where it came from, even if it can’t see it now.
- **Updating the state**: To keep this memory accurate, the agent needs to:
  - Understand how the world changes over time.
  - Know how its own actions will change the world around it.



3. Goal-based agents

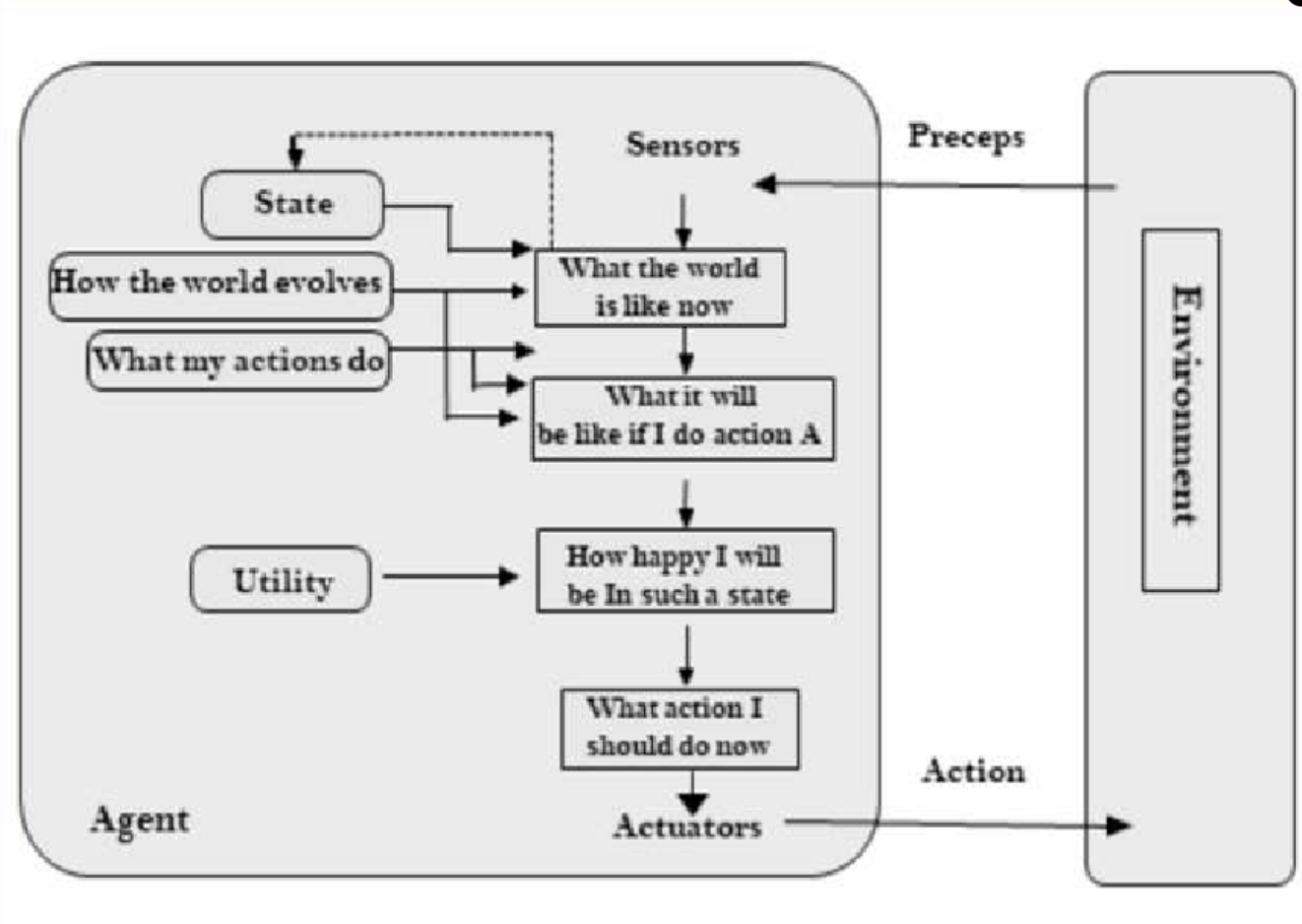
- Goal-based agents are a type of AI that make decisions to achieve specific goals. Unlike reflex agents, they don't just react to situations; they use knowledge to plan their actions.
- This makes them more flexible, as you can change their behavior by adjusting their goals.

- A "goal" simply describes the outcome the agent is trying to achieve.



4. Utility Based Agent

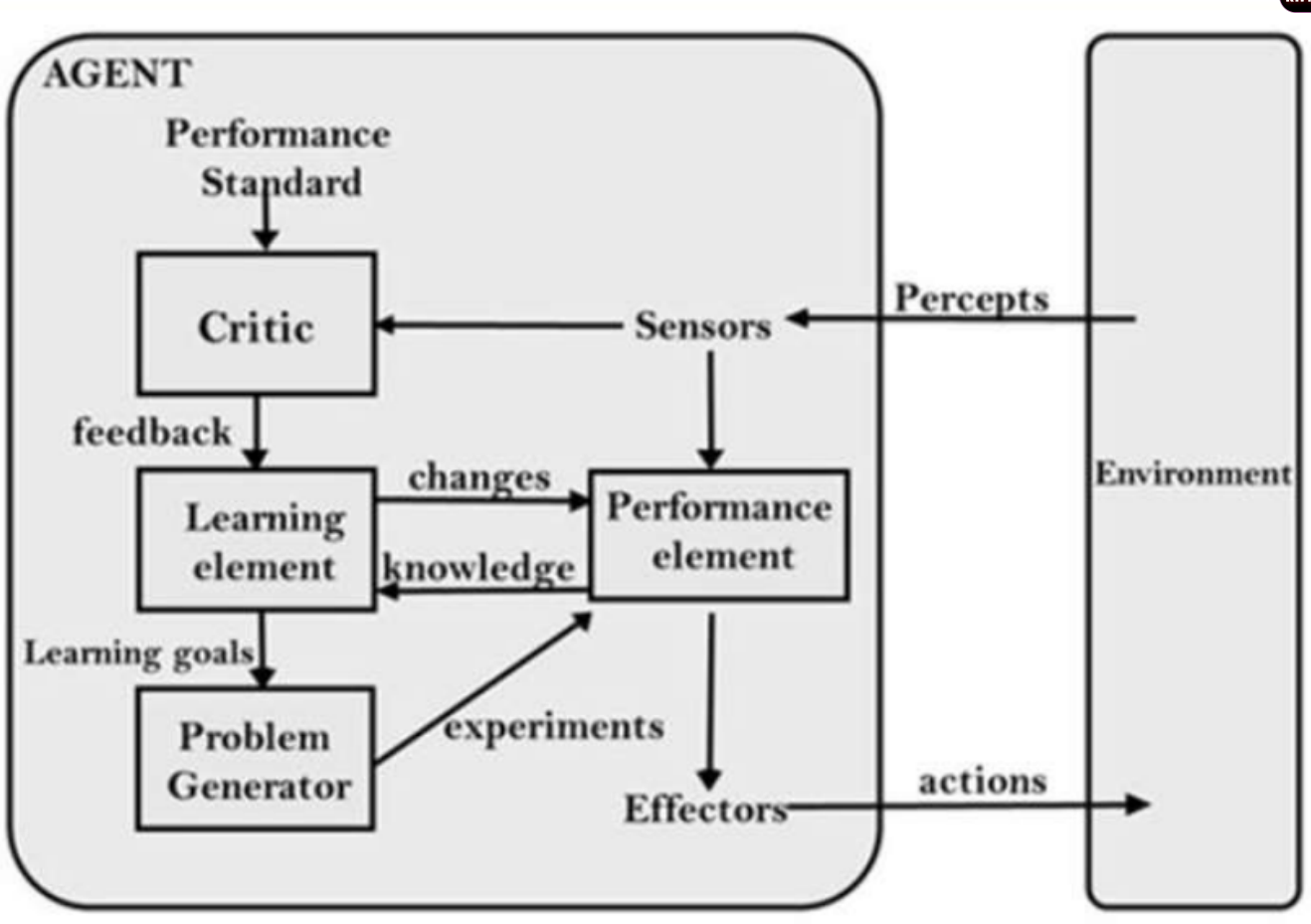
- **Utility-based agents** decide what to do by ranking different outcomes, or states, based on how useful or beneficial (utility) they are.
- Unlike goal-based agents, they don’t just aim for one goal but **balance multiple goals that might conflict**.
- When some goals can't be fully achieved, utility-based agents choose the action that gives the **best possible result**.
- They also factor in uncertainty, considering both the chances of success and how important each goal is.



5. Learning agent

A learning agent is an AI that improves its performance by learning from past experiences. It starts with basic knowledge and then adapts automatically as it learns. It has four main parts:

1. **Learning element**: This part learns from the environment and makes improvements.
2. **Critic**: It provides feedback to the learning element, showing how well the agent is doing compared to a set standard.
3. **Performance element**: This part chooses the actions the agent will take.
4. **Problem generator**: It suggests new actions that help the agent explore and gain more useful experiences.



2. AI

Artificial Intelligence (AI) is the field of computer science focused on creating machines that can think and act intelligently, similar to how humans do.

Levels of AI:

1. **Narrow AI**: **Machines perform specific tasks better** than humans, such as facial recognition or playing chess. Current AI research is focused here.
2. **General AI**: A machine that can perform **any intellectual task at the same level** as a human, though this has not been achieved yet.
3. **Strong AI**: Machines that **surpass human intelligence in many areas,** being able to perform tasks better than humans across a wide range of domains.

Goals of Artificial Intelligence (AI):

1. **Create Expert Systems**: Develop systems that can perform tasks intelligently, learning, explaining, and advising users, much like a human expert.
2. **Implement Human Intelligence in Machines**: Build machines that can think, learn, understand, and behave like humans.

Types of AI:

- **General-purpose AI**: Like the robots seen in science fiction, which can perform a wide range of tasks. However, replicating the complexity of the human brain is extremely difficult, and we still don't fully understand how it works.
- **Special-purpose AI**: Focuses on specific tasks, which is more achievable. Examples include AI used in chess or poker programs, logistics planning, voice recognition, web search, data mining, medical diagnosis, and self-driving cars.

3. Heuristic Function

Informed Search Algorithms:

- In computer science, particularly in AI, **search algorithms** help find the solution to problems by exploring different options.
- **Uninformed search algorithms** explore the problem space without any extra information they don’t know which direction is better and just try every possibility.
- **Think of it like trying to find your friend in a big mall without a map**; you'd check every corner one by one.
- **Informed search algorithms** are smarter. They use extra information or "clues" to guide them in the right direction.
- For example, **if you're looking for your friend in a mall and you know they are near a certain store,** you'd head there first instead of wandering aimlessly. These clues come from something called a **heuristic**.

What is a Heuristic Function?

- A **heuristic function** is like an estimation tool that helps an algorithm decide which path is likely to get to the goal faster.

- **Imagine you're driving to a city and you have a rough idea of how far you are from your destination**—this rough idea is what the heuristic function does for the algorithm. It helps choose the most promising path.
- The **heuristic function (h(n))** gives an estimate of how close the current state (n) is to the **goal**.
- It doesn't always give the perfect solution, but it tries to find a good one **quickly**.
- The function outputs a positive number, representing an **estimated cost to reach the goal from the current point**.
- **Example**
  - if you're in a maze and the goal is to reach the exit, **the heuristic could be the "straight-line distance" from your current position to the exit**, even though walls might make the actual path longer.
- *_h(n) <= h*(n)_*
  - **h(n)** is the heuristic estimate, and *_h*(n)_* is the actual cost to reach the goal.
  - The estimate from the heuristic should always be less than or equal to the real cost, meaning **it should not overestimate the difficulty of reaching the goal.**

Pure Heuristic Search:

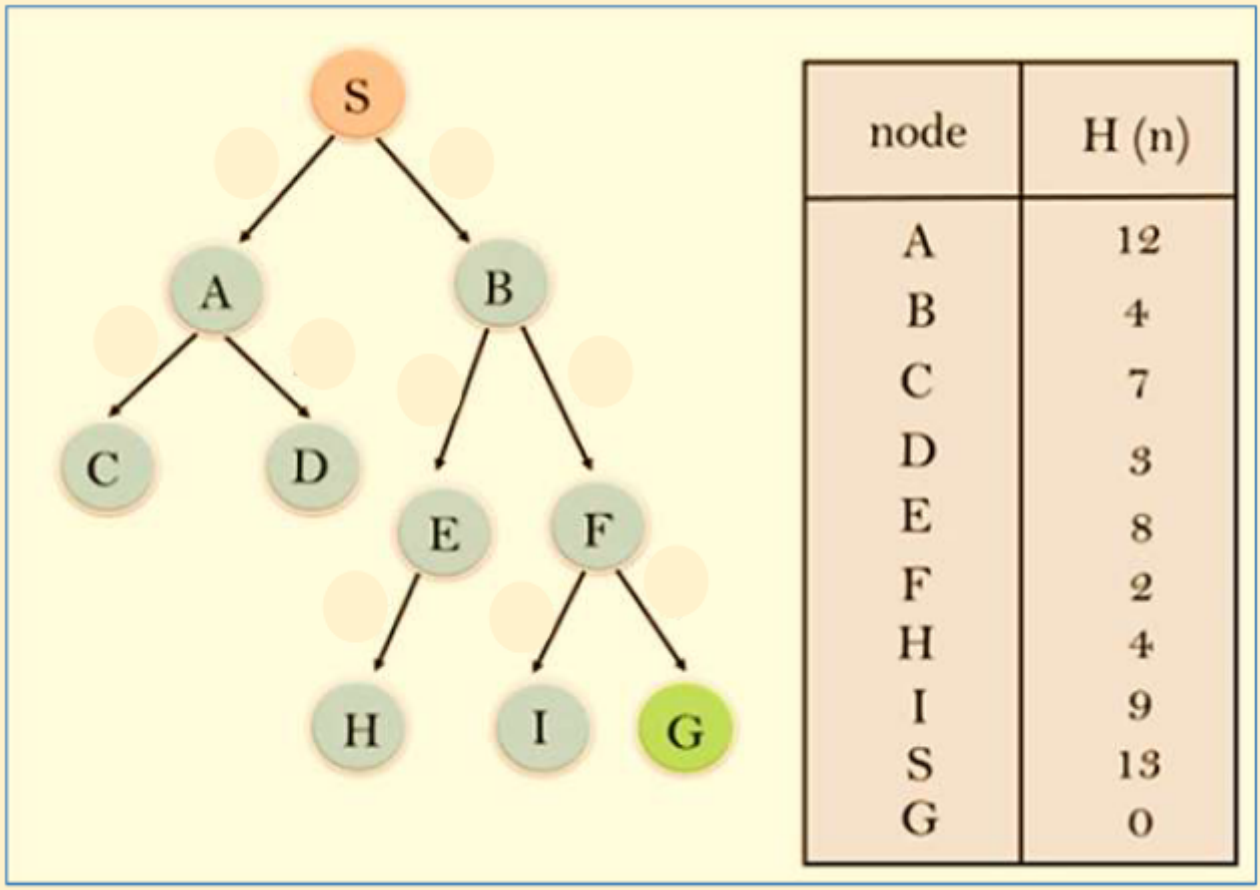
- This is a simple algorithm that selects the next step based on the **heuristic value** alone.
- It keeps two lists:
  - **OPEN list**: contains nodes that are yet to be explored.
  - **CLOSED list**: contains nodes that have already been explored.
- At each step, it picks the node with the smallest heuristic value and expands it. This process repeats until the goal is found.

Best-First Search (Greedy Search):

- It tries to combine the strengths of two common search strategies: **depth-first search** (which goes deep into one path) and **breadth-first search** (which explores all options at the same level first).
- The algorithm uses the heuristic function to pick the "best" option at every step.
- It expands the node that seems closest to the goal based on the heuristic value.

- For example, **if you're navigating a city and you think a particular road seems fastest, you'll take that road even if it may not lead to the goal directly.**

Example



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

- Here you can see the nodes (A,B,C ...G) and their heuristic values (12,4,7 ... 0)
- **Our goal is to reach from S to G.**
- First we start from S
  - Here A and B are successor nodes
  - We visited S, so we put it to closed List
  - We discovered 2 successor Nodes A and B, so adding them to Open List
  - Open List = [A,B]
  - Closed List = [S]
- From A and B, the one with smaller heuristic value is B (Its value is 4)
  - We are visiting B, so modifying the lists
  - Open List = [A]
  - Closed List = [S,B]

- Visiting B
  - The successor nodes are E and F
  - Open List = [A,E,F]
  - Closed List = [S,B]
- From E and F, the one with smaller heuristic value is F (Its value is 2)
  - Visiting F
  - Open List = [A,E]
  - Closed List = [S,B,F]
- Visiting F
  - The successor nodes are I and G
  - Open List = [A,E,I,G]
  - Closed List = [S,B,F]
- Visiting G (Heuristic is 0 and its our goal)
  - Open List = [A,E,I,G]
  - Closed List = [S,B,F,G]
- So our final solution path is
  - S -> B -> F -> G

A* Algorithm

- A* (A-star) algorithm is a smart way to find the shortest path between two points. Think of it like navigating a map to find the best route from your house to a friend's place
- But instead of just checking the distance, you also consider how easy or difficult it is to walk along the path.

Key Concepts:

1. **g(n)**: This is the actual cost to reach a point (node) from the starting position.
2. **h(n)**: This is the estimated cost to reach the destination (goal) from that point (node). It’s like a guess based on how far the destination might be.
3. **f(n) = g(n) + h(n)**: A* combines both the actual cost to reach a point and the estimated cost to reach the goal from that point.

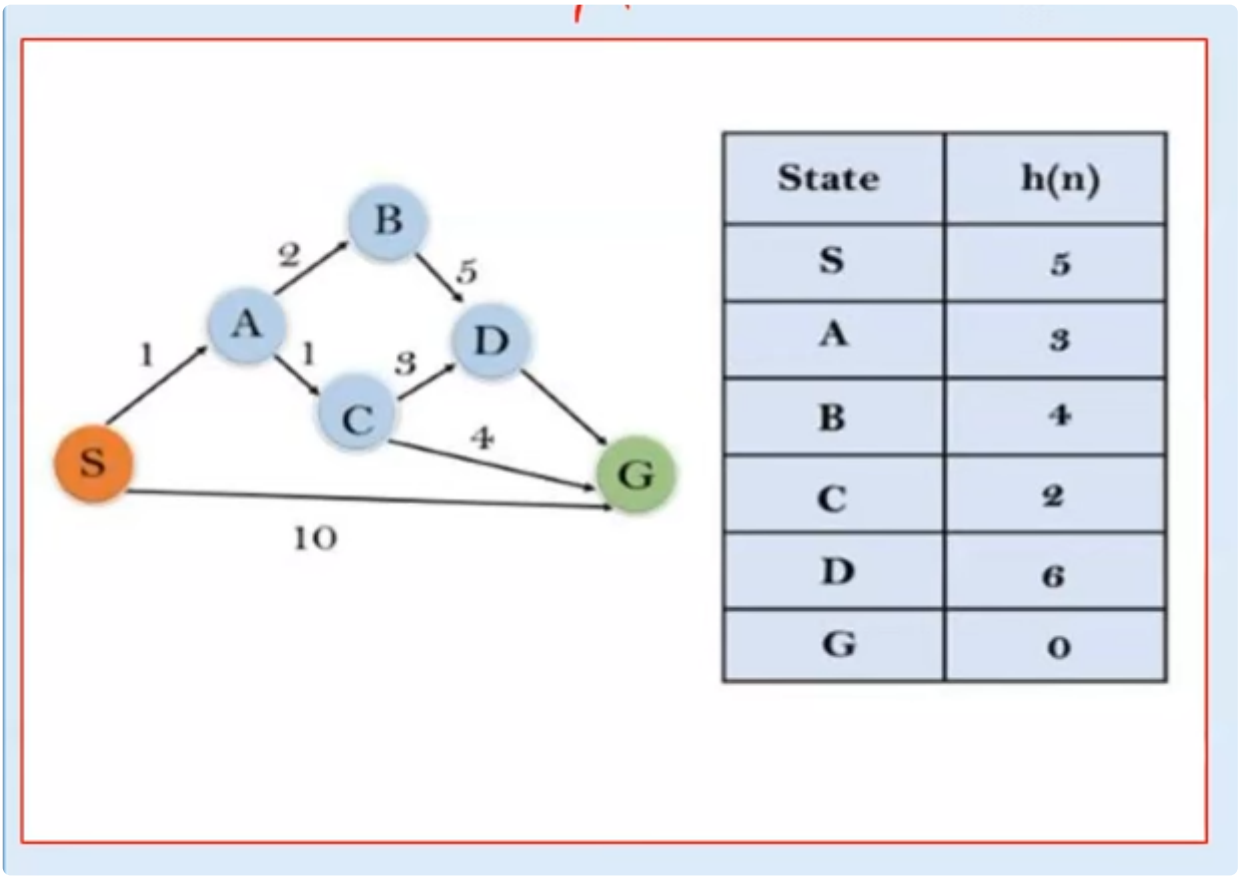
How it Works:

- The algorithm checks all possible routes, but it tries to be smart by expanding the paths that seem most promising. It does this by looking at the sum of the actual distance traveled (g) and the estimated remaining distance (h).
- A* explores the path with the **smallest f(n)**, which balances the known distance and the guess for the remaining distance.

Steps of the Algorithm:

1. Start at the initial point and place it in a list called the **OPEN list** (the list of nodes to be explored).
2. Pick the node from the OPEN list with the smallest f(n) value.
3. If this node is the goal, you're done! If not, generate all possible next steps (successor nodes) from this point.
4. Add these new points to the OPEN list to explore later.
5. Move the current node to a **CLOSED list** (so you don’t check it again).
6. Repeat until you find the goal or no more points can be explored.

Example



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

- Our goal is to go from S to G
  - From S, there are 2 paths
  - S -> A
    - We need to calculate f(n)
    - **f(n) = g(n) + h(n)**
    - g(n) is the distance from S to A (From the graph we can see that the distance is 1)
    - h(n) is the heuristic value (The h(n) value of A is 3 from the table)
    - f(n) = 1+3 = 4
  - S -> G
    - **f(n) = g(n) + h(n)**
    - f(n) = 10+0= 10
  - When comparing f(n) of A and G, A is smaller with the value of 4. So we will check the nodes from A
- From A we have 2 paths
  - S -> A -> B
    - **f(n) = g(n) + h(n)**

        - f(n) = 3 + 4 = 7 (We got that 3 from S-> A = 1 and A->B = 2, S->A->B = 1+2 = 3)
    - S -> A -> C
        - f(n) = 2 + 2 = 4
    - f(n) of C is smaller
- From C we have 2 paths
    - S -> A -> C -> D
        - f(n) = 5 + 6 = 11
    - S -> A -> C -> G
        - f(n) = 6 + 0 = 6
    - f(n) of G is smaller.
- Since we reached G, we need to compare with the earlier nodes to see if they are smaller
    - S -> G f(n) = 10
    - S -> A -> B f(n) = 7
    - S -> A -> C -> G f(n) = 6
- S -> A -> C -> G f(n) = 6 is the smallest
- So our best path is
    - S -> A -> C -> G

Advantages

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages

• It does not always produce the shortest path as it mostly based on heuristics and approximation.
• A *search algorithm has some complexity issues.*
• *The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

4. PEAS

PEAS is a framework that helps describe how an AI agent works by breaking it down into four parts:

- **P (Performance Measure):** This is how we judge the success of the AI's actions. For example, in an automated taxi, performance could be measured by how safely and efficiently it gets passengers to their destination.
- **E (Environment):** This is the world the AI operates in. For a taxi, this would be the city streets, weather, traffic, and pedestrians.
- **A (Actuators):** These are the parts of the AI that allow it to take action. In a taxi, the actuators would be the steering wheel, brakes, and accelerator.
- **S (Sensors):** These help the AI gather information about the environment. In the taxi's case, sensors could include cameras, GPS, and radar.

5. Iterative deepening

Iterative Deepening Depth-First Search (IDDFS) is a search algorithm that combines the strengths of both Depth-First Search (DFS) and Breadth-First Search (BFS).

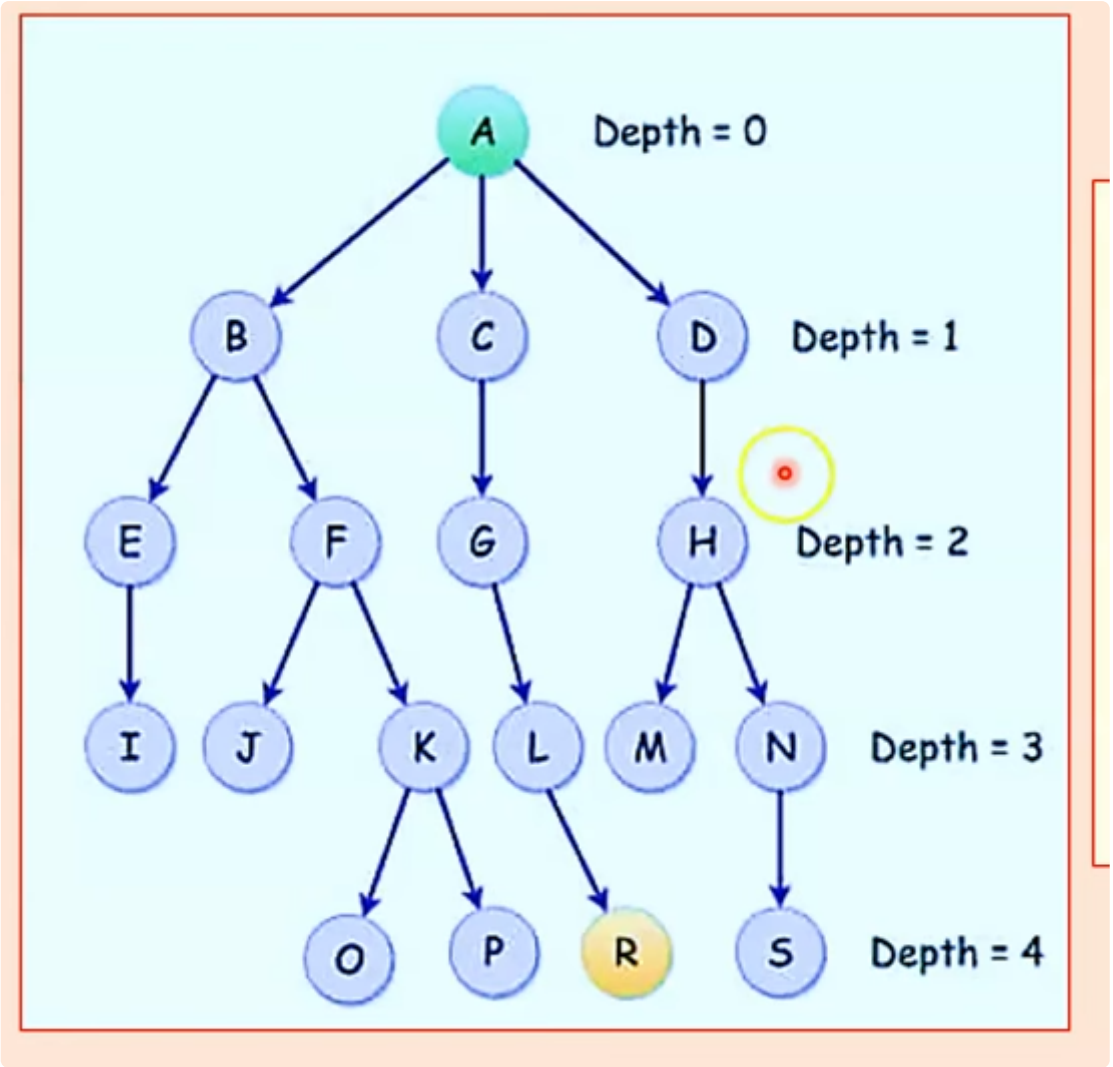
How it works:

- IDDFS works by performing a series of Depth-First Searches, each with an increasing "depth limit."
- It starts with a shallow search and gradually goes deeper until it finds the goal (or solution).
- For each iteration, it performs DFS up to a certain depth. If the goal isn't found, it increases the depth and repeats the search.
- **Where it's helpful:**
  - It's useful in situations where the search space is large, and we don't know how deep the solution is (for example, in game trees or puzzles).
- **Advantages:**
  - **Memory-efficient** like DFS.
  - **Fast to find a solution** like BFS.
- **Disadvantages:**

- It **repeats some work** from previous iterations, since every time it increases the depth limit, it has to search the same nodes as before.

- **Time complexity:**
  - If the branching factor is `b` (how many children each node has), and `d` is the depth of the goal node, the time complexity is around `O(b^d)`.

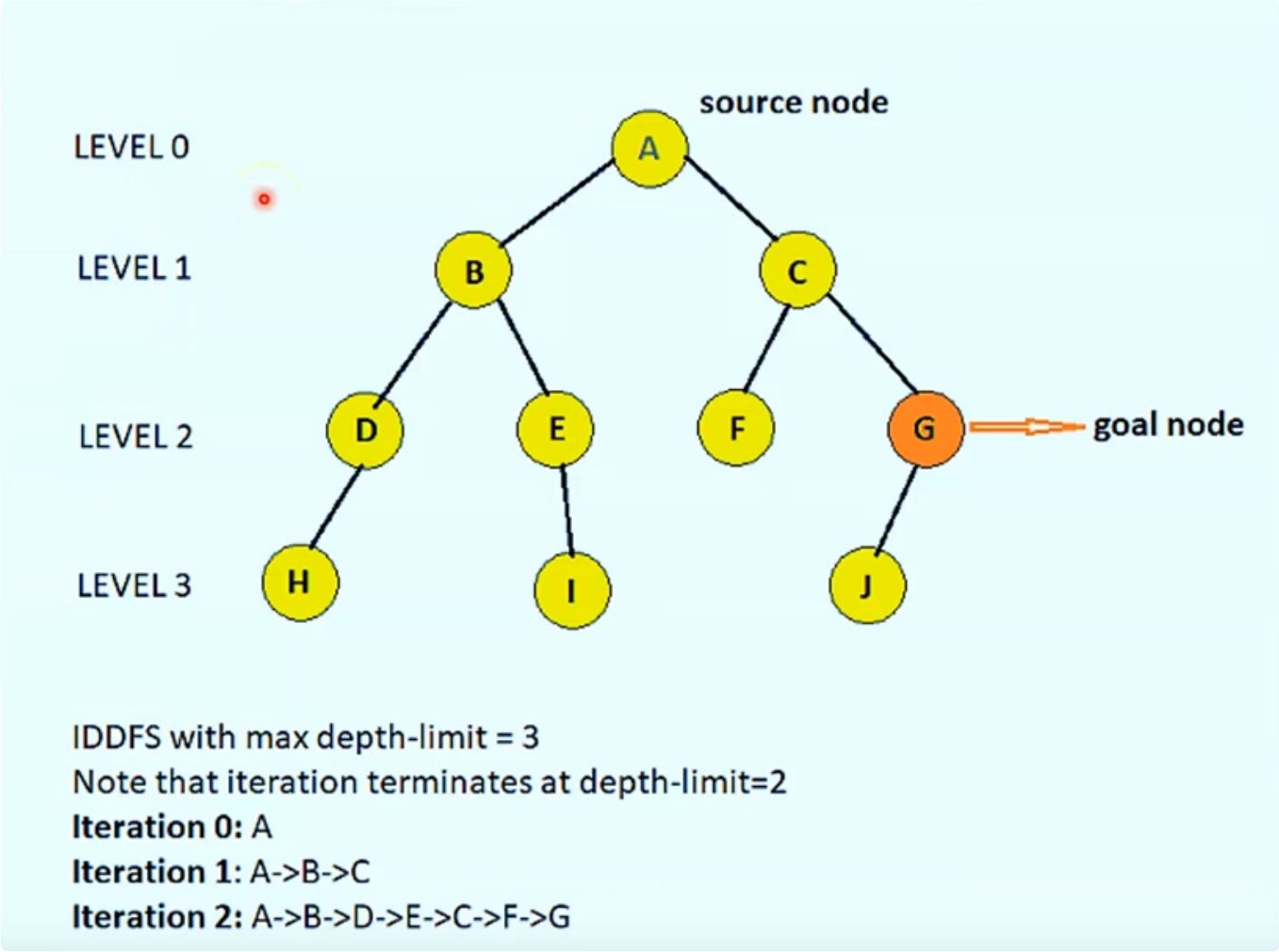
Example 1



- Our goal is to reach R
- As you can see there are different levels of depth, like depth = 0, depth = 1 ... depth = 4
- We will traverse this tree by each depth level
- Depth level 0
  - Only A is there
  - A

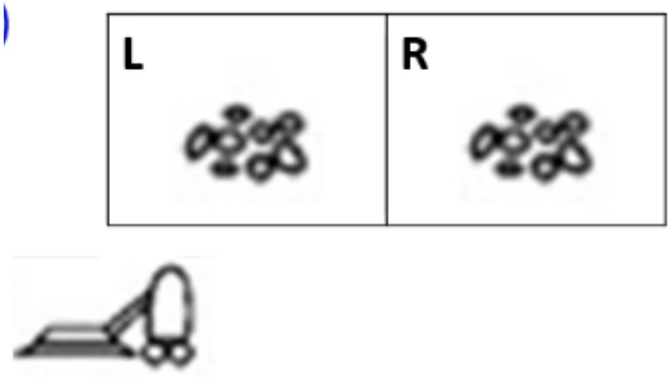
- Depth level 1
  - We will go from Left to right and go upto the depth level
  - A,B,C,D
- Depth Level 2
  - A,B,E,F,C,G,D,H
- Depth Level 3
  - A,B,E,I,F,J,K,C,G,L,D,H,M,N
- Depth Level 4
  - A,B,E,I,F,J,K,O,P,C,G,L,R,D,H,M,N,S

Example 2



6. Vaccum cleaner

Vacuum World Problem



- There are two dirty square blocks and we need to clean these dirty square blocks by using a vaccum cleaner
- Percepts: Location and Content
  - Eg (L,Dirty)
- Possible actions for vacuum cleaner
  - Move Left (L)
  - Move Right (R)
  - Suck (S)
  - No Operation (NoOp)

Initial State

- Vacuum Agent can be in any state(Left or Right)

Successor function

- Successor function generates legal states resulting from applying the 3 actions {Left, RIght and Suck}

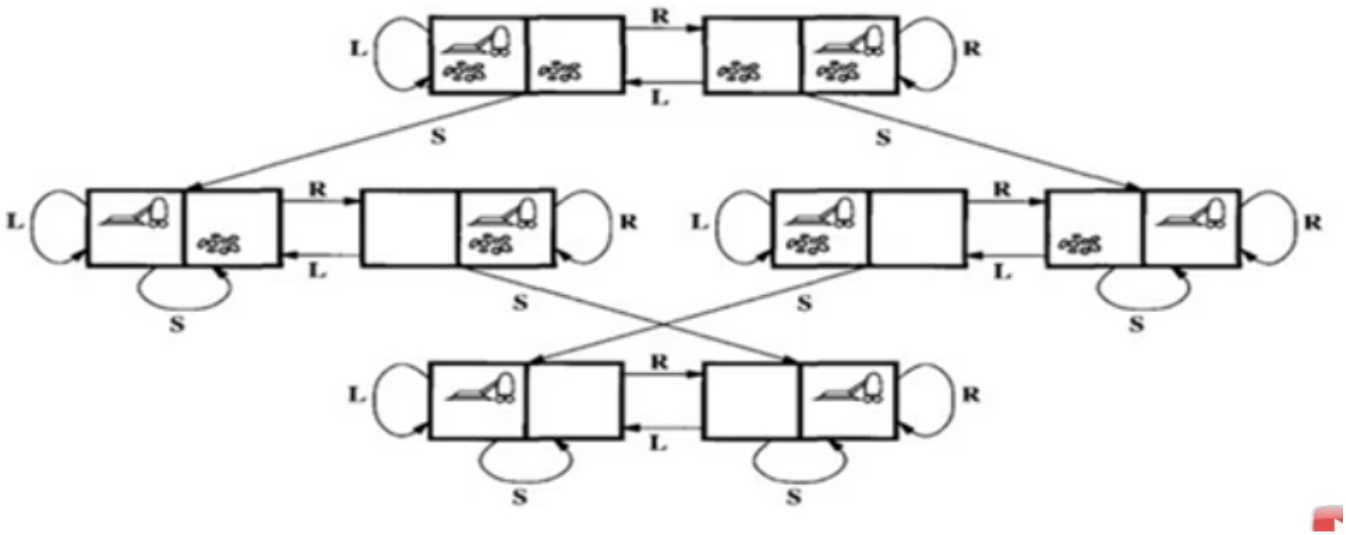
Goal Test

- Checks whether all squares are clean

Path Cost

- Each step costs 1, so the path cost is the sum of steps in the path from Initial state to goal state

Transition Diagram



- First 2 boxes are dirty
  - We can do the following operations
    - Suck the dirt
    - Move left
    - Move right
  - Suppose we Suck the dirt
    - Now one box is clean
    - We can move right
  - Moving Right
    - There is dirt on the second box, which can be sucked
  - After sucking the dirt, we have 2 clean squares
- Since we did this in 3 steps, the path cost is 3

7. Tree search and graph search

Aspect	Tree Search Algorithm	Graph Search Algorithm
Structure	Operates on a tree (no cycles, hierarchical structure)	Operates on a graph (may contain cycles)

Aspect	Tree Search Algorithm	Graph Search Algorithm
Cycle Handling	Does not handle cycles (no need in trees)	Handles cycles by keeping track of visited nodes
Memory Usage	Lower memory usage (no need to store visited nodes)	Higher memory usage (must track visited nodes)
Node Revisit	Assumes each node is visited only once	Prevents revisiting nodes by maintaining a visited set
Efficiency in Cycles	Inefficient in cyclic structures (can revisit nodes)	Efficient in cyclic structures (avoids revisiting)
Applicability	Best suited for acyclic or hierarchical problems	Best suited for general graphs, including cyclic ones
Common Example	Depth-First Search (DFS) for trees	Breadth-First Search (BFS) with cycle detection

8. Uniform cost search Algorithm

- This algorithm is used to traverse the tree which have weight on the edge
- Each edge have different weight
- The goal is to find the path from root to goal node having lowest path cost
- It is implemented by priority queue, gives maximum priority to lowest weighted edge

Advantages

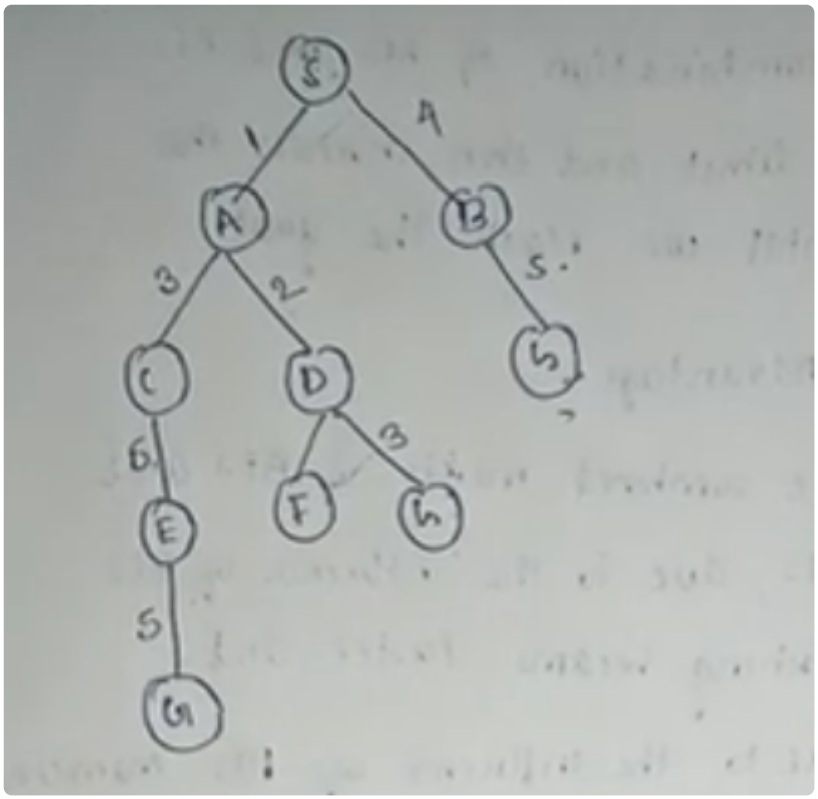
- It is optimal if we choose the path with least cost

Disadvantages

- It does not care about the number of steps, only care about the path cost

Example

To Reach G from S



- We have to follow the lowest path cost
- The possibilities
  - S -> A -> C -> E -> G
    - 1 + 3 + 6 + 5 = 15
  - S -> A -> D -> G
    - 1 + 2 + 3 = 6
  - S -> B -> G
    - 4 + 5 = 9
- Among these, the path with shortest path cost is
- S -> A -> D -> G (cost = 6)

9. Admissible and Consistent Heuristics

Admissible Heuristic

- An admissible heuristic is a function that estimates the cost from the current state to the goal, but it never overestimates this cost.

- In other words, it is "optimistic." For example, if you’re trying to find the shortest path to a destination, an admissible heuristic will always **guess either the exact distance or something less, but never more.**
- Think of it like planning a road trip: if your GPS tells you the trip will take no more than 5 hours, you know you’ll either arrive in exactly 5 hours or sooner—but not later.

Consistent Heuristic (Monotonic)

- A consistent heuristic, has an additional property: the estimated cost to reach the goal must always be less than or equal to the estimated cost from the current node to a neighbor, plus the actual cost to move from the current node to that neighbor.