1. Define Artificial Intelligence and explain why it is important to study AI.

Artificial Intelligence (AI) is a broad field of computer science focused on creating systems or agents that can perform tasks that typically require human intelligence. These tasks include learning, reasoning, problem-solving, perception, understanding natural language, and (in some contexts) creativity.

Importance of Studying AI:

- **Solving Complex Problems:** AI can find patterns and solutions in data sets that are too large or complex for the human mind to process, leading to breakthroughs in science, medicine (e.g., drug discovery), and engineering.
- **Automation and Efficiency:** AI automates repetitive or dangerous tasks, increasing productivity, reducing errors, and improving safety in areas like manufacturing, logistics, and data analysis.
- **Understanding Intelligence:** By trying to build intelligent systems, we gain a deeper understanding of our own cognitive processes, learning, and decision-making.
- **New Capabilities and Services:** AI powers new technologies that have become integral to modern life, such as search engines, recommendation systems, virtual assistants (Siri, Alexa), and accessibility tools for people with disabilities.

---

2. Explain the four main approaches that define AI: Thinking Humanly, Acting Humanly, Thinking Rationally, and Acting Rationally.

These four approaches represent different goals and philosophies for defining what AI is:

1. **Thinking Humanly (The Cognitive Modeling Approach):**
    - **Goal:** To build systems that *think* in the same way humans do.
    - **Method:** This requires understanding the internal workings of the human mind, often by drawing from cognitive psychology and neuroscience.
    - **Evaluation:** The system's success is measured by how well its internal reasoning process matches psychological models of human thought.
2. **Acting Humanly (The Turing Test Approach):**
    - **Goal:** To build systems that *act* indistinguishably from a human.
    - **Method:** This approach focuses on external behavior. The "Turing Test" is the classic example, where a human interrogator tries to determine if they are communicating with a human or a machine.
    - **Evaluation:** The system is successful if it can fool the interrogator. This requires capabilities like natural language processing, knowledge representation, and learning.
3. **Thinking Rationally (The "Laws of Thought" Approach):**

    - **Goal:** To build systems that think using the principles of formal logic.
    - **Method:** This approach is based on creating systems that use logic and syllogisms to derive "correct" conclusions from a set of premises.
    - **Evaluation:** The system's success is measured by its ability to perform correct logical inference.
4. **Acting Rationally (The Rational Agent Approach):**
    - **Goal:** To build systems that **act to achieve the best expected outcome** given their available information.
    - **Method:** This agent acts "rationally" by making decisions that maximize its **performance measure**. This is more general than "thinking rationally" because it can handle uncertainty.
    - **Evaluation:** The system is successful if it consistently makes the best possible choices to achieve its goals. This is the **dominant approach in modern AI**.

---

3. Discuss the main challenges faced in building intelligent systems.

- **Knowledge Representation:** How to store vast amounts of complex, abstract, and common-sense knowledge in a way that is usable for reasoning.
- **Reasoning under Uncertainty:** The real world is not black-and-white. Systems must be able to make good decisions using incomplete, ambiguous, or probabilistic information.
- **Learning:** Developing systems that can learn from data, adapt to new situations, and generalize their knowledge effectively.
- **Common Sense:** Programming the vast, unspoken set of assumptions and knowledge that humans use to navigate the world.
- **Computational Complexity:** Many AI problems are "NP-hard," meaning the time required to find a perfect solution explodes as the problem size grows.
- **Explainability (XAI):** As AI models (especially deep learning) become "black boxes," a major challenge is building systems that can explain *why* they reached a particular conclusion.
- **Ethics, Safety, and Alignment:** Ensuring that AI systems are fair, unbiased, robust, and that their objectives are aligned with human values.

---

4. Differentiate between Weak AI and Strong AI with examples.

Feature	**Weak AI (Artificial Narrow Intelligence - ANI)**	**Strong AI (Artificial General Intelligence - AGI)**

Definition	AI designed and trained to perform a **specific, narrow task**.	A theoretical type of AI that possesses the full range of human cognitive abilities.
Capability	It **simulates** human intelligence for its specific function. It is not conscious or self-aware.	It would have genuine consciousness, understanding, and the ability to learn and apply its intelligence to **any** problem.
Goal	To create a tool that is useful for a single purpose.	To create a machine with the same intellectual capacity as a human being.
Status	**Exists today.** This is the basis for all current AI applications.	**Hypothetical.** It does not currently exist.
Examples	• Siri/Alexa • Spam filters • Google Search • Facial recognition software	• Data (Star Trek) • HAL 9000 (2001: A Space Odyssey) • The robots in *Westworld*

---

5. Describe the concept of Rationality in AI and explain how Rational Agents are evaluated using a Performance Measure.

Concept of Rationality

In AI, **rationality** does not mean "thinking logically" or "being all-knowing." An agent is considered **rational** if it selects actions that are expected to **maximize its performance measure**, given the information it has.

This means a rational agent:

1. Uses its **percepts** (what it senses from the environment).
2. Uses its **built-in knowledge**.
3. Chooses the action that it *believes* will lead to the best possible score.

Evaluating with a Performance Measure

A **Performance Measure** is the objective **criterion for success** for an agent. It's an external metric that evaluates how well the agent is doing its job.

- **How it's used:** We, the designers, define the performance measure to specify *what* we want the agent to achieve, not *how* to achieve it.
- **Evaluation:** The agent's *rationality* is evaluated *by* this measure.
- **Example:** For a **vacuum cleaner agent**, the performance measure might be:
    - +10 points for every gram of dirt vacuumed.
    - -1 point for every unit of energy consumed.
    - -50 points for running over the cat.

A rational vacuum agent will then choose its actions (move left, move right, suck) to maximize this total score.

---

6. Briefly trace the historical milestones in the development of Artificial Intelligence

- **1940s-1950s (The Birth):**
    - **1950:** Alan Turing's paper "Computing Machinery and Intelligence" introduces the "Turing Test".
    - **1956:** The **Dartmouth Workshop** officially coins the term "Artificial Intelligence" and establishes it as a field.
- **1950s-1960s (Early Enthusiasm):**
    - Development of early programs like the Logic Theorist and checkers-playing programs.
    - John McCarthy invents the LISP programming language.
- **1970s (The First "AI Winter"):**
    - Optimism faded due to the difficulty of common-sense reasoning and computational limits. Funding was cut.
- **1980s (The Boom of Expert Systems):**
    - AI rebounds with "Expert Systems," which are knowledge-based programs designed to replicate human expertise in a narrow domain (e.g., MYCIN).
- **Late 80s - Early 90s (The Second "AI Winter"):**
    - Expert systems proved expensive to build and brittle. The market for them collapsed.
- **1990s - 2000s (Modern AI Takes Shape):**
    - Shift toward **Machine Learning** and **probabilistic reasoning**.
    - **1997:** IBM's Deep Blue defeats world chess champion Garry Kasparov.
- **2010s - Present (The Deep Learning Revolution):**
    - The availability of **Big Data** and powerful **GPU hardware** allows for the success of **Deep Learning**.
    - **2012:** AlexNet wins the ImageNet competition, demonstrating a massive leap in computer vision.
    - **2016:** Google DeepMind's AlphaGo defeats world Go champion Lee Sedol.
    - Rise of large language models (LLMs) like Transformers and GPT.

---

7. Define an Intelligent Agent. Explain the structure of agents and environments using the PEAS framework.

Intelligent Agent

An **Intelligent Agent** is any entity that **perceives** its environment through **sensors** and **acts** upon that environment through **actuators** in pursuit of a goal. The agent's "intelligence" lies in its **agent function**, which maps its history of percepts to a chosen action.

- **Agent = Architecture + Program**

PEAS Framework

PEAS is a framework used to define the task environment for an intelligent agent.

- **P - Performance Measure:** The criteria used to evaluate the agent's success.
- **E - Environment:** The "world" in which the agent operates.
- **A - Actuators:** The "body parts" the agent uses to *act* on the environment.
- **S - Sensors:** The "senses" the agent uses to *perceive* the environment.

---

8. Construct PEAS specifications for the following:

(i) Autonomous Taxi Driver

	Specification
Performance	**Safety**, **Legality**, **Speed**, **Comfort**, **Profit**.
Environment	Roads, traffic, pedestrians, customers, other vehicles, weather, road signs.
Actuators	Steering wheel, accelerator, brake, indicators (signals), horn, display.
Sensors	Cameras, **LIDAR**, **RADAR**, GPS, sonar, odometer, accelerometer, microphone.

(ii) Weather Forecasting System

	Specification
Performance	**Accuracy** of the forecast, **Timeliness**, **Coverage**.
Environment	The Earth's atmosphere, oceans, landmasses; streams of historical and real-time data.
Actuators	A display (website, app), a system for sending alerts.
Sensors	Weather satellites, ground-based weather stations, radar systems, weather balloons, ocean buoys.

---

9. Differentiate between Reflex, Model-based, Goal-based, and Utility-based Agents with examples.

Agent Type	**How it Works**	**Example**
Simple Reflex Agent	**Acts *only* on the current percept.** Uses simple "Condition-Action" rules. It has **no memory** of the past.	• A simple thermostat (IF temp < 68° THEN turn on heat).
Model-Based Reflex Agent	**Maintains an internal "model" (state) of the world.** It uses its model and the current percept to act. This allows it to handle *partial observability*.	• A self-driving car that **remembers** a truck was in its blind spot.
Goal-Based Agent	**Acts to achieve a specific "goal" state.** It uses its model to *plan* and *search* for a sequence of actions that will lead to the goal.	• A navigation app (Google Maps) finding a route to a destination.

Utility-Based Agent	**Acts to maximize its "utility" (happiness).** This is used when there are **conflicting goals** or uncertainty. It uses a **utility function** to assign a numeric value to the "desirability" of a state.	• A taxi agent deciding which ride to accept, balancing **profit**, **time**, and **distance**.

---

10. Formulate a PEAS description for an Intelligent Tutoring System.

	Specification
Performance	**Student learning** (e.g., improved scores), **student engagement**, **topics mastered**, **efficiency**.
Environment	The student, a database of lessons, questions, and learning materials.
Actuators	**Display** (showing lessons, problems, hints, feedback), keyboard/mouse prompts.
Sensors	**Keyboard/Mouse** (for student's answers), **quiz results**, timers (time to answer).

---

11. Explain the components of a well-defined problem with an example.

A problem is "well-defined" for a search algorithm if it has these four components:

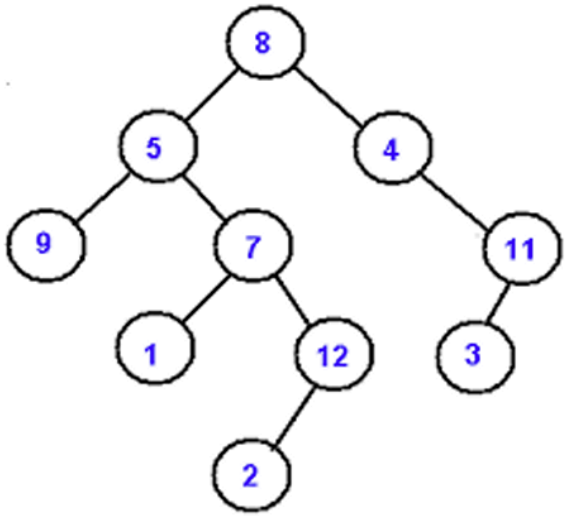
1. **Initial State:** The state from which the agent starts.
2. **Actions (or Successor Function):** A description of the possible actions available to the agent in any given state.
3. **Goal Test:** A function that determines whether a given state is a "goal state" (a solution).
4. **Path Cost:** A function that assigns a numeric cost to a path.

Example: The 8-Puzzle

- **1. Initial State:** A specific, jumbled configuration of the 8 tiles (e.g., [[7, 2, 4], [5, 0, 6], [8, 3, 1]]).
- **2. Actions:** Move the blank space (0) in one of four directions: **Up, Down, Left, or Right**.
- **3. Goal Test:** Checks if the current state's configuration matches the goal (e.g., [[1, 2, 3], [4, 5, 6], [7, 8, 0]]).
- **4. Path Cost:** Each move has a cost of 1. The total path cost is the total number of moves.

---

12. Apply Breadth-First-Search (BFS) to the following graph starting from node 8 to goal node 3. Show all steps, including Open and Closed lists.



BFS explores level by level using a **FIFO (First-In, First-Out) queue**.

- **Start:** 8
- **Goal:** 3

Step	Node Expanded	Closed List (Explored)	Open List (Frontier)
1	-	[ ]	[**8**]

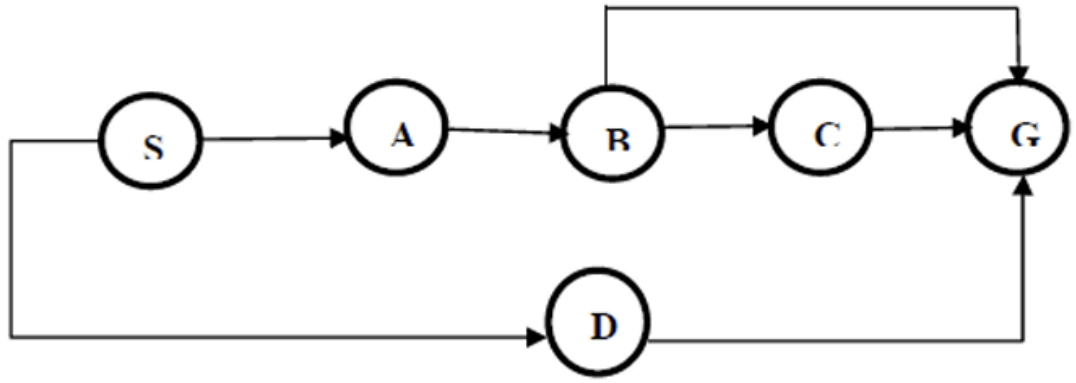
2	**8**	[8]	[5, 4]
3	**5**	[8, 5]	[4, 9, 7]
4	**4**	[8, 5, 4]	[9, 7, 11]
5	**9**	[8, 5, 4, 9]	[7, 11]
6	**7**	[8, 5, 4, 9, 7]	[11, 1, 12]
7	**11**	[8, 5, 4, 9, 7, 11]	[1, 12, **3**]
8	**1**	[8, 5, 4, 9, 7, 11, 1]	[12, **3**]
9	**12**	[8, 5, 4, 9, 7, 11, 1, 12]	[**3**, 2]
10	**3**	[8, 5, 4, 9, 7, 11, 1, 12, 3]	[2]

- **Goal Found!**
- **Order of Node Expansion:** 8, 5, 4, 9, 7, 11, 1, 12, 3
- **Final Path (by tracing parents):** 3 $\rightarrow$ 11 $\rightarrow$ 4 $\rightarrow$ 8

---

13. Apply Depth-First Search (DFS) on the following graph with Source as S and Goal as G, and compare results with BFS.

13. Apply Depth-First Search (DFS) on the following graph with Source as S and Goal as G, and compare results with BFS.



DFS explores as deeply as possible first, using a **LIFO (Last-In, First-Out) stack**.

DFS Trace

(Assuming children are explored in alphabetical order)

1. **Start at S.** Push S. Stack: [S]
2. **Pop S.** Find children: A, D. Push D, then A. Stack: [**A**, D]
3. **Pop A.** Find children: B. Push B. Stack: [**B**, D]
4. **Pop B.** Find children: C, G. Push G, then C. Stack: [**C**, G, D]
5. **Pop C.** Find children: G. Push G. Stack: [**G**, G, D]
6. **Pop G. GOAL FOUND!**

- **DFS Path Found:** S $\rightarrow$ A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ G
- **DFS Expansion Order:** S, A, B, C, G

Comparison with BFS

- **BFS Trace (FIFO Queue):**
    1. Queue: [S]
    2. Dequeue S. Enqueue A, D. Queue: [A, D]
    3. Dequeue A. Enqueue B. Queue: [D, B]
    4. Dequeue D. Enqueue G. Queue: [B, G]
    5. Dequeue B. Enqueue C, G. Queue: [G, C, G]
    6. Dequeue G. **GOAL FOUND!**
- **BFS Path Found:** S $\rightarrow$ D $\rightarrow$ G

Comparison:

- **DFS** found the path **S $\rightarrow$ A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ G** (length 4).
- **BFS** found the path **S $\rightarrow$ D $\rightarrow$ G** (length 2).

- In this case, BFS found the **shortest path** (in terms of number of edges), while DFS found a longer path.

---

14. Explain Depth-Limited Search and Iterative Deepening Search with examples.

Depth-Limited Search (DLS)

- **Concept:** DLS is a modification of Depth-First Search (DFS) that avoids its problem of infinite loops. It does this by imposing a pre-defined **depth limit (L)**.
- **How it works:** It performs a standard DFS, but it treats any node at depth L as if it has no successors.
- **Properties:**
    - **Not Complete:** If the solution is at a depth $d > L$, DLS will fail to find it.
    - **Good Space Complexity:** $O(bL)$.
- **Example:** In a graph, if you set $L=3$, the search will explore paths like A $\rightarrow$ B $\rightarrow$ C, but it will **stop** and backtrack at C, refusing to explore any children of C (which would be at depth 4).

Iterative Deepening Search (IDS)

- **Concept:** IDS combines the **space-efficiency of DFS** with the **completeness and optimality of BFS** (for unit step costs).
- **How it works:** It runs DLS repeatedly with progressively larger depth limits ($L=0, 1, 2, ...$).
    - Run DLS with $L=0$.
    - If no solution, discard all nodes and run DLS with $L=1$.
    - If no solution, discard all nodes and run DLS with $L=2$.
    - ...It continues this until a solution is found at depth d.
- **Properties:**
    - **Complete:** Yes.
    - **Optimal:** Yes (for unit step costs), because it finds the shallowest solution first.
    - **Excellent Space Complexity:** $O(bd)$.
    - **Good Time Complexity:** $O(b^d)$. It may seem wasteful, but the cost of re-generating upper-level nodes is insignificant compared to the cost of exploring the last level.

---

15. Define heuristic function. How does it help in Informed Search?

Definition

A **heuristic function**, denoted **$h(n)$**, is a function that **estimates the cost of the cheapest path from a given node n to a goal state.**

The heuristic is an "educated guess." It is *not* the true cost and it is *not* the cost from the start (that's $g(n)$). It is an estimate of the *remaining* cost.

- **Example (Route-Finding):** $h(n)$ = The straight-line (Euclidean) distance from city n to the goal city.
- **Example (8-Puzzle):** $h(n)$ = The "Manhattan distance" (sum of the horizontal and vertical distances of each tile from its goal position).

How it Helps in Informed Search

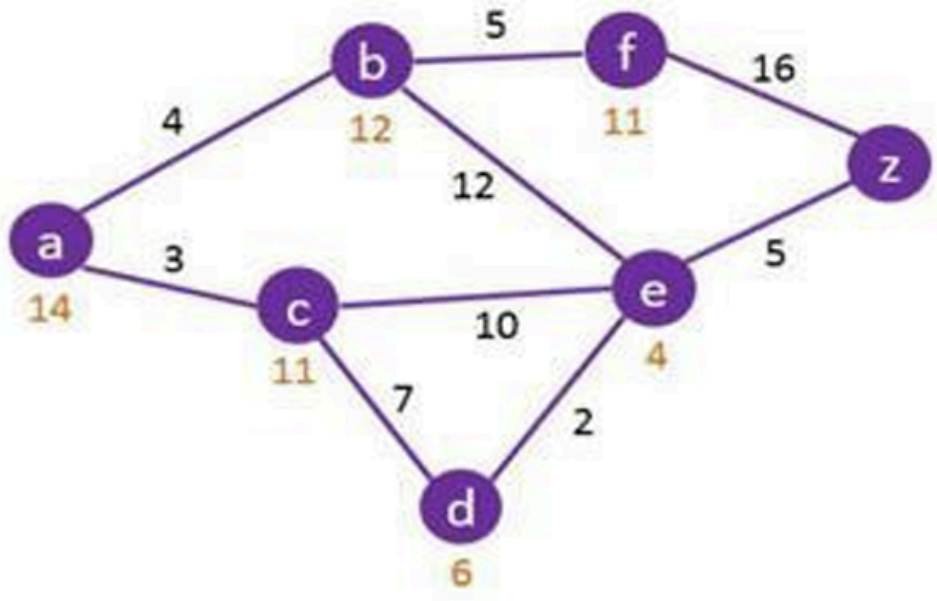
A heuristic function provides **domain-specific knowledge** to a search algorithm, which is what "informs" the search.

1. **Guides the Search:** Instead of exploring blindly, an informed search uses the heuristic to prioritize nodes that *seem* to be closer to the goal.
2. **Improves Efficiency:** By expanding promising nodes first, the algorithm can find a solution much faster, effectively "pruning" large, unpromising sections of the search tree.

---

16. For the following graph, apply Best-First Search and show the order of node expansion and final path to goal.

16. For the following graph, apply Best-First Search and show the order of node expansion and final path to goal.



Greedy Best-First Search expands the node that *appears* to be closest to the goal, based only on the **heuristic value $h(n)$**. The edge weights are ignored.

- **Start:** a

- **Goal:** z
- **Heuristic Values:** $h(a)=14$, $h(b)=12$, $h(c)=11$, $h(d)=6$, $h(e)=4$, $h(f)=11$, $h(z)=0$ (assumed for goal).

We use a **Priority Queue** ordered by the lowest $h(n)$.

1. **Start at 'a'.**
    - **Open List:** [{**a**, $h=14$}]
2. **Expand 'a' ($h=14$).**
    - Find children: 'b' ($h=12$) and 'c' ($h=11$).
    - **Open List:** [{**c**, $h=11$}, {b, $h=12$}]
3. **Expand 'c' ($h=11$).**
    - Find children: 'e' ($h=4$) and 'd' ($h=6$).
    - **Open List:** [{**e**, $h=4$}, {d, $h=6$}, {b, $h=12$}]
4. **Expand 'e' ($h=4$).**
    - Find children: 'z' ($h=0$), 'b' ($h=12$), 'd' ($h=6$).
    - **Open List:** [{**z**, $h=0$}, {d, $h=6$}, {b, $h=12$}]
5. **Expand 'z' ($h=0$).**
    - **GOAL FOUND!**
- **Order of Node Expansion:** a, c, e, z
- **Final Path (by tracing parents):** z $\leftarrow$ e $\leftarrow$ c $\leftarrow$ a
- **Final Path: a $\rightarrow$ c $\rightarrow$ e $\rightarrow$ z**

---

17. Explain A* Search algorithm in detail.

A^* (A-star) is an informed search algorithm that finds the lowest-cost path from a *start* node to a *goal* node.

Core Idea

A^* works by maintaining a priority queue of paths to explore. It always expands the path that has the lowest **estimated total cost**, calculated by the evaluation function:

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

- **$g(n)$ (Cost-to-Come):** The **actual, known cost** of the path from the start_node to the current node n.
- **$h(n)$ (Heuristic):** The **estimated cost** of the cheapest path from n to the goal_node.
- **$f(n)$ (Total Estimated Cost):** The sum of the past cost and the future *estimated* cost.

Data Structures

1. **Open List (or Frontier):** A **priority queue** that stores all nodes that have been *generated* but not yet *expanded*. Nodes are sorted by their $f(n)$ value (lowest $f(n)$ at the front).

2. **Closed List (or Explored Set):** A **set** that stores all nodes that have already been *expanded*. This prevents re-expanding nodes and getting stuck in cycles.

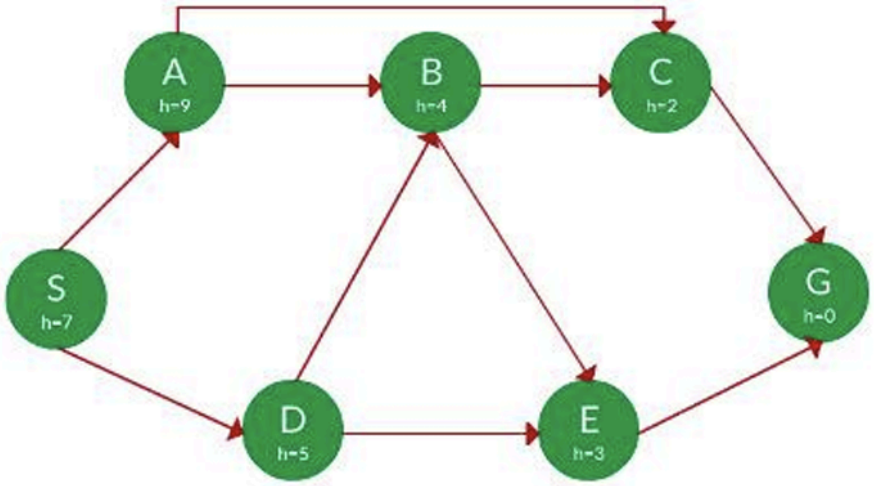
Algorithm Steps

1. **Initialize:**
    - Create the **Open List** and add the start_node to it.
    - Set the start_node's $g(n)=0$ and $f(n) = h(start\_node)$.
    - Create an empty **Closed List**.
2. Loop: While the Open List is not empty:
   a. Select Node: Pop the node n with the lowest $f(n)$ value from the Open List.
   b. Goal Check: If n is the goal_node, stop! Reconstruct the path by following parent pointers.
   c. Expand Node: Move n from the Open List to the Closed List.
   d. Generate Successors: For each successor (neighbor) of n:
   i. Check Closed List: If successor is in the Closed List, ignore it.
   ii. Calculate Cost: new_g = g(n) + cost(n, successor)
   iii. Check Open List:
   * If successor is NOT in the Open List: Add it. Set its parent to n, set its $g(n) = new\_g$, and calculate its $f(n) = new\_g + h(successor)$.
   * If successor IS ALREADY in the Open List: Check if this new path is better. If new_g < g(successor), update the successor's $g(n)$ to new_g, update its parent to n, and recalculate its $f(n)$.
3. **Failure:** If the Open List becomes empty, no path exists.

---

18. Apply A* Search to find the shortest path from node A to node G for the graph below.

18. Apply A* Search to find the shortest path from node A to node G for the graph below. Clearly show g(n), h(n), and f(n) values for each node.



A* search uses $f(n) = g(n) + h(n)$.

(Assuming a uniform step cost of 1 for each edge, as costs are not given.)

- **Start:** A
- **Goal:** G
- **Step Cost:** 1

Step	Expand Node	g(n)	h(n)	f(n)	Open List (Frontier)	Closed (Explored)
1	-	-	-	-	[{**A**, $f=9$} ($g=0, h=9$)]	[ ]
2	**A**	0	9	9	[{**B**, $f=5$} ($g=1, h=4$)]	[A]
3	**B**	1	4	5	[{**C**, $f=4$} ($g=2, h=2$), {E, $f=5$} ($g=2, h=3$)]	[A, B]
4	**C**	2	2	4	[{**G**, $f=3$} ($g=3, h=0$), {E, $f=5$} ($g=2, h=3$)]	[A, B, C]
5	**G**	3	0	3	[{E, $f=5$} ($g=2, h=3$)]	[A, B, C, G]

- **Goal Found!**
- **Shortest Path:** A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ G
- **Total Cost:** $g(G) = 3$

---

19. Differentiate between Uninformed and Informed Search strategies.

Feature	Uninformed Search (Blind Search)	Informed Search (Heuristic Search)
Knowledge	Only uses the problem definition: initial state, actions, and goal test.	Uses problem-specific knowledge in the form of a **heuristic function** ($h(n)$).
Strategy	Explores the state space systematically (e.g., by depth, by breadth).	Explores the state space by prioritizing nodes that *seem* to be closer to the goal.
Efficiency	Generally less efficient. Can be very slow for large problems.	Generally much more efficient. The heuristic guides the search.
Examples	• Breadth-First Search (BFS) • Depth-First Search (DFS) • Uniform-Cost Search (UCS)	• Greedy Best-First Search • **A* Search**

---

20. Write the advantages and limitations of each search algorithm: BFS, DFS, UCS, Greedy, and A*.

Algorithm	Advantages	Limitations
BFS (Breadth-First)	• **Complete:** Always finds a solution. • **Optimal:** Finds the *shallowest* solution.	• **Time/Space Complexity:** Bad ($O(b^d)$). Stores the entire frontier.
DFS (Depth-First)	• **Space Complexity:** Excellent ($O(bm)$). Only stores the current path.	• **Not Complete:** Can get stuck in infinite loops. • **Not Optimal:** Finds the *first* solution, not the shortest.

UCS (Uniform-Cost)	• **Complete:** Yes. • **Optimal:** Always finds the solution with the *lowest path cost*.	• **Time/Space Complexity:** Bad ($O(b^{1 + C^*/\epsilon})$). Explores based on cost.
Greedy BFS	• **Fast (in practice):** Often finds *a* solution quickly by following the heuristic.	• **Not Complete:** Can be misled by the heuristic. • **Not Optimal:** Makes "greedy" choices.
A*	• **Complete:** Yes. • **Optimal:** Yes, **if** the heuristic $h(n)$ is **admissible**.	• **Time Complexity:** Can be exponential in the worst case. • **Space Complexity:** Bad ($O(b^d)$). Stores all explored nodes.

(Where b = branching factor, d = depth of solution, m = max depth, $C^$ = cost of optimal solution, ϵ = min step cost)

---

22. Explain the role of heuristics and admissibility in A* Search.

Role of the Heuristic ($h(n)$)

In A^* search, the algorithm evaluates nodes using: **$f(n) = g(n) + h(n)$**

The **role of $h(n)$ is to guide the search**. A^* always expands the node with the **lowest $f(n)$ value**. A good heuristic (one that is close to the true cost) will direct A^* to explore nodes along the optimal path, allowing it to find the goal much faster than an uninformed search.

Role of Admissibility

A heuristic $h(n)$ is **admissible** if it **never overestimates** the true cost to reach the goal.

- **Condition:** $h(n) \le h^*(n)$ for all nodes n, where $h^*(n)$ is the *true* cost from n to the goal.
- **Example:** In route-finding, the **straight-line distance** is admissible because the *actual* road distance can never be *less* than the straight-line distance.
- **Why it is crucial:** Admissibility is what **guarantees that A* is optimal**. If $h(n)$ is admissible, A^* will *always* find the lowest-cost path.

---

23. Explain the properties of heuristic search: Completeness, Optimality, Time and Space Complexity.

These four properties are the standard way we evaluate any search algorithm:

1. **Completeness:**
    - **Question:** "Does the algorithm always find a solution if one exists?"
    - **Heuristic Context:** A^* is complete. Greedy BFS is **not complete**.
2. **Optimality:**
    - **Question:** "Does the algorithm always find the *best* (lowest-cost) solution?"
    - **Heuristic Context:** A^* is **optimal** *if* its heuristic $h(n)$ is **admissible**. Greedy BFS is **not optimal**.
3. **Time Complexity:**
    - **Question:** "How long does it take to find a solution?"
    - **Heuristic Context:** In the worst case, both are exponential ($O(b^d)$). However, a high-quality heuristic can make the *practical* time much faster.
4. **Space Complexity:**
    - **Question:** "How much memory does the algorithm need?"
    - **Heuristic Context:** This is the biggest weakness of A^* and Greedy BFS. They must store the entire frontier in memory, leading to exponential space complexity ($O(b^d)$).

---

24. Compare Greedy Best-First Search and A* Search with respect to heuristic accuracy.

Feature	**Greedy Best-First Search**	**A* Search**
Evaluation Function	**$f(n) = h(n)$**	**$f(n) = g(n) + h(n)$**
What it Prioritizes	The node that *seems* closest to the goal. (Short-sighted)	The node on the path that *seems* to have the lowest *total solution cost*. (Balanced)
Impact of an *Inaccurate* Heuristic	**Very high impact.** A misleading heuristic can easily cause the algorithm to go	**More robust.** The $g(n)$ term "anchors" the search in reality. As the $g(n)$ cost of a wrong path grows, A^* will

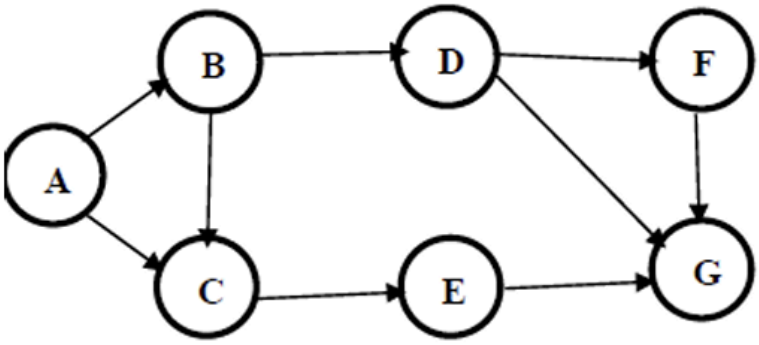
	down a very long, suboptimal path.	eventually be "pulled back" to explore more promising paths.
Optimality	**Never guaranteed.**	**Guaranteed**, as long as the heuristic is *admissible*.

In short: Greedy BFS *blindly trusts* the heuristic. A^* *uses the heuristic as a guide* but balances it with the real cost $g(n)$ to ensure optimality.

---

25. Apply Best First Search and A* Search to find the shortest path from node A to node G for the graph below.

25. Apply Best First Search and A* Search to find the shortest path from node A to node G for the graph below. Clearly show g(n), h(n), and f(n) values for each node.



A	5
B	6
C	4
D	3
E	3
F	1
G	0

*(Assuming a uniform step cost of **1** for each edge, as costs are not given.)*

- **Start:** A
- **Goal:** G
- **Heuristics:** $h(A)=5, h(B)=6, h(C)=4, h(D)=3, h(E)=3, h(F)=1, h(G)=0$

Part 1: Greedy Best-First Search (evaluates $h(n)$)

1. **Open List:** [{**A**, $h=5$}]
2. **Expand A ($h=5$).** Children: B ($h=6$), C ($h=4$).
    - **Open List:** [{**C**, $h=4$}, {B, $h=6$}]
3. **Expand C ($h=4$).** Children: B.
    - **Open List:** [{**B**, $h=6$}]
4. **Expand B ($h=6$).** Children: D ($h=3$).

    - **Open List:** [{**D**, $h=3$}]
5. **Expand D ($h=3$).** Children: F ($h=1$), G ($h=0$).
    - **Open List:** [{**G**, $h=0$}, {F, $h=1$}]
6. **Expand G ($h=0$). GOAL FOUND!**

- **Path Found (Greedy):** A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ G

Part 2: A* Search (evaluates $f(n) = g(n) + h(n)$)

Step	Expand Node	g(n)	h(n)	f(n)	Open List (Frontier)	Closed (Explored)
1	-	-	-	-	[{**A**, $f=5$} ($g=0, h=5$)]	[ ]
2	**A**	0	5	5	[{**C**, $f=5$} ($g=1, h=4$), {B, $f=7$} ($g=1, h=6$)]	[A]
3	**C**	1	4	5	[{**B**, $f=7$} ($g=1, h=6$)] *Path to B via C (g=2, f=8) is worse*	[A, C]
4	**B**	1	6	7	[{**D**, $f=5$} ($g=2, h=3$)]	[A, C, B]
5	**D**	2	3	5	[{**G**, $f=3$} ($g=3, h=0$), {F, $f=4$} ($g=3, h=1$)]	[A, C, B, D]
6	**G**	3	0	3	[{F, $f=4$} ($g=3, h=1$)]	[A, C, B, D, G]

- **Goal Found!**
- **Shortest Path (A*):** A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ G
- **Total Cost:** $g(G) = 3$