HITEC UNIVERSITY TAXILA
Department of Computer Science
BS Artificial Intelligence Program / BS Cyber Security Program
3rd Semester, Session-2024
Mid Term Examination, Fall 2025

Subject:	**CS-312 Artificial Intelligence**	**Date:**	**Tuesday December 02, 2025**
Max Marks:	**50**	**Max Time:**	**120 Minutes**

Solution File

Question #1	**[CO-1 – C2] [SO-1]**	**(Marks: 5 × 2 = 10)**

1. Demonstrate your understanding of fundamental Artificial Intelligence concepts by differentiating between the following with examples

1.1. Intelligence and Artificial Intelligence

Intelligence is the ability to understand, learn, reason, and apply knowledge to solve problems in daily life.
Daily Life Examples of Intelligence

- Choosing the fastest route to university when traffic is heavy
- Recognizing your friend's voice on the phone
- Learning how to ride a bicycle after falling a few times
- Solving a puzzle by trying different strategies

John McCarthy, who invented the term Artificial Intelligence in **1956**, defines it as

"The science and engineering of making intelligent machines", especially intelligent computer programs.

AI is the study and design of ***Intelligent Agents***, where an intelligent agent is a system that perceives its environment and take actions that maximize its chances of success. AI is concerned with the design of ***intelligence*** in an ***artificial*** device. Intelligence in Machines:

- **Observe** → Sensors/Inputs Devices (e.g., camera, microphone)
- **Reason** → Algorithms (e.g., decision-making models)
- **Decide** → Action (e.g., robot moves, chatbot responds)
- **Learn** → Machine Learning (improving with experience)

AI is about building machines that can mimic this human-like intelligence.

1.2. Evolutionary AI and Artificial Super Intelligence.

Evolutionary AI is a type of Artificial Intelligence inspired by biological evolution (Darwin's theory of natural selection).
How it Works

- Uses genetic algorithms and evolutionary strategies.
- Creates many possible solutions → tests them → keeps the "fittest" ones.
- Over many generations, the system evolves better and smarter solutions.

Key Features

- Learns through trial and error.

- Improves automatically without step-by-step programming.
- Useful when the problem space is huge and complex.

Examples

- Designing antennas for satellites (NASA used Evolutionary AI).
- Optimizing routes in logistics or delivery systems.
- Evolving game strategies in simulations.

Artificial Super Intelligence (ASI)

Artificial Super Intelligence (ASI) refers to a stage where machines not only match but **surpass** human intelligence in every possible aspect like creativity, reasoning, emotional understanding, decision-making, and social skills. It represents the highest level of AI evolution, where machines can outperform humans in all domains, like scientific innovation, art, ethics, and strategic thinking.

Key Characteristics

- **Beyond human capability:** Thinks, learns, and innovates faster than humans.
- **Self-learning and self-improving:** Continuously enhances its own algorithms.
- Possesses consciousness and emotions (theoretical).
- Can predict and solve complex global problems with unimaginable efficiency.
- May raise **ethical and control challenges,** how to ensure it benefits humanity.

Examples (Futuristic / Fictional)

- **Ultron (Avengers)** – self-aware and surpasses human intelligence.
- **Skynet (Terminator)** – autonomous system surpassing human control.
- **AI Singularity Concept** – a hypothetical future point where ASI evolves beyond human understanding.

1.3. Simple Reflex Agent and Goal-Based Agent.

Simple Reflex Agent

- Characteristics
  - Very limited knowledge or intelligence
  - Only works if the environment is fully observable.
  - Lacking history, (operating in partially observable environment)
  - Easily get stuck in infinite loops.

Goal-Based Agents

- Knowing about the current state of the environment is not always enough to decide what to do.
  - For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to.
- The agent needs some sort of goal information that describes situations that are desirable.
  - For example, being at the passenger's destination.
  -

1.4. Stack and Queue

Stack

A Stack is a linear data structure that follows the LIFO (Last In, First Out) principle.

Operations:

- push(x) Adds element x to the top of the stack
- pop() Removes the top element from the stack
- peek() or top() Views the top element without removing it
- isEmpty() Checks if the stack is empty

Examples

1. Undo/Redo Functionality: Used in editors like **MS Word**, **Photoshop,** Each action is **pushed** onto a stack, Pressing Undo **pops** the last action.

2. Backtracking (e.g., Maze Solving, DFS), Stacks are used to track the path.

3. Expression Evaluation & Syntax Parsing: Compilers use stacks for: Infix to postfix conversion

4. Stack of Plates / Books: Last plate/book added is the first to be removed.
5. Browser History (Back Button): Each visited page is **pushed** onto a stack, Pressing "Back" **pops** the current page and shows the previous.
6. Reversing a Word: Push each character onto a stack, then pop to reverse the string.
7. Boxing Items: Items are packed in boxes, last item packed is first to unpack.

Queues
A Queue is a linear data structure that follows the FIFO (First In, First Out) principle.
Operations:

- enqueue(x) Adds element x at the rear
- dequeue() Removes and returns element from front
- front() Returns element at the front (without removing it)
- isEmpty() Checks if the queue is empty

Type of Queues :
1. Simple Queue Standard FIFO queue
2. Circular Queue Last position connects to the first to reuse space
3. Priority Queue Elements dequeued based on **priority**, not just order
4. Double-Ended Queue: Insertions and deletions from both front and rear

1.5. Difference between an optimal solution and a near-optimal solution.

Optimal Solution (Best Possible Solution)
An **optimal solution** is the *best* solution according to a defined cost function. In search algorithms, *optimality* means:

- The solution has the **minimum path cost** (or maximum utility).
- No other solution exists with a lower cost.
- Algorithms like **Uniform Cost Search (UCS)** and **A*** (with *admissible* and *consistent* heuristic) **guarantee optimal solutions**.

Near-Optimal Solution (Almost Best but Not Perfect)
A **near-optimal solution** is close to the optimal solution but:

- Does **not guarantee** minimum cost
- Trades off accuracy for speed, memory, or practicality
- Usually found in heuristic-based or approximate search methods (e.g., Hill-Climbing, Beam Search, Genetic Algorithms)

Question #2	**[CO-1 – C2] [SO-1]**	**(Marks: 4 × 2.5 = 10)**

2. Demonstrate your understanding of basic uninformed search algorithms by answering the following:

2.1. Explain how Breadth-First Search works. Mention one advantage and one disadvantage

Breadth First Search (BFS) is a graph traversal algorithm that explores nodes level by level. It starts from a selected node, called the root or starting point, and visits all its immediate neighbors before moving on to the next level of neighbors. BFS uses a queue data structure to keep track of nodes that need to be explored. As each node is visited, its unvisited neighbors are added to the queue. This process continues until all reachable nodes are visited. BFS is especially useful for finding the shortest path in unweighted graphs because it guarantees that the first time a node is reached, it is through the shortest possible route.

Benefits of BFS

- Finds the **shortest path** in unweighted graphs.
- Easy to implement. Simple and systematic.
- Useful for **level-order traversal** in trees.

Disadvantages

- Requires **more memory** (queue may grow large).
- Slower for deep or infinite search spaces.
- Not suitable for problems where the solution is far from the root.

2.2. Describe Depth First Search and briefly explain how backtracking happens.

Depth First Search (DFS) is an uninformed search algorithm that explores as far as possible along each branch before backtracking. It uses a stack (LIFO) structure, either explicitly (with a stack data structure) or implicitly via recursion.

Advantages

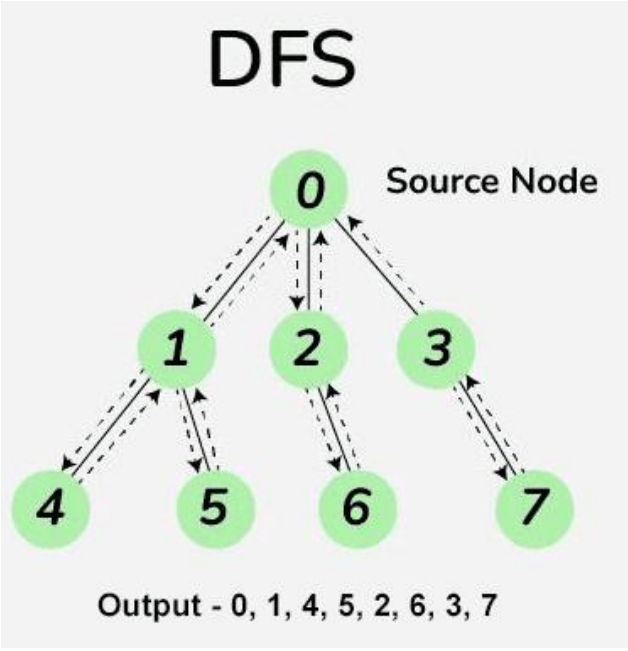
- Uses less memory than BFS (stores only one path).
- Can find a solution without exploring all nodes (if goal is deep).

Disadvantages

- May get **stuck in an infinite loop** in cyclic graphs.
- Not **optimal** — may find longer path first.
- Not **complete** — might miss the goal if depth limit not set.

Applications

- Solving **mazes** or **puzzles** (like N-Queens).
- **Pathfinding** in AI (where full exploration is required).
- Topological sorting and cycle detection in graphs.
- **Backtracking problems** (e.g., Sudoku, crosswords).



2.3. What is Depth Limited Search? Why do we use a depth limit?

Depth-Limited Search (DLS) is a variant of Depth-First Search (DFS) in which the search depth is restricted to a fixed limit (L). It explores the graph like DFS but stops exploring any branch once it reaches the specified depth limit. DFS may get trapped in infinite loops (especially in graphs with cycles or infinite depth). DLS solves this problem by imposing a limit on how deep the algorithm can go, preventing infinite descent.

2.4. What is Uniform Cost Search, and how is it different from Breadth First Search?

Uniform Cost Search (UCS) is a search algorithm that expands the least-cost node first, rather than the lowest node (like BFS). It guarantees finding the optimal (lowest cost) path to the goal if all step costs are non-negative. UCS is basically a Breadth-First Search with costs, where the priority is given to path cost instead of depth. Imagine you're finding the cheapest route between two cities on a map. You don't just look for the shortest number of roads, but the least total travel cost (distance or time). That's what UCS does, it always picks the path with the smallest accumulated cost so far.

Working Principle

- Start from the root node.
- Maintain a priority queue (min-heap) where priority = path cost (g(n)).
- At each step, expand the node with the lowest cost.
- Add successors to the queue with their cumulative path costs.
- Stop when the goal node is reached - this guarantees the minimum cost path.

Question #3	**[CO-2 – C2] [SO-1]**	**(Marks: 10)**

3. A city transportation department wants to deploy an AI-based Smart Bus Management System**. The system should be able to**

3.

a) Detect passenger flow at different stops using live video feeds
b) Predict bus overcrowding before it happens.
c) Adjust bus schedules dynamically based on traffic patterns.
d) Notify drivers and the control center when unsafe driving behavior (e.g., sudden braking, speeding) is detected.
e) Provide real-time updates to passengers waiting at bus stops.

Using your understanding of the PEAS framework, identify and describe the Performance Measure, Environment, Actuators, and Sensors for this Smart Bus Management System. Ensure that your answer clearly connects the PEAS elements to the scenario.

Solution:

PEAS specification for Smart Bus Management System

1) Performance Measure (P) — what "good" looks like

Design performance measures that reflect safety, passenger experience, and operational efficiency. Use a mixture of **primary metrics** (hard targets) and **secondary metrics** (quality / trade-offs).

Primary metrics (examples & targets):

Passenger detection accuracy (for live video):

- Precision ≥ 92% ; Recall ≥ 90% (min) — reduces under/over-count errors.
- Latency: detection ≤ 200 ms per frame (real-time). *(Directly supports f — passenger flow detection.)*

Overcrowding prediction lead time & accuracy (g):

- Predict overcrowding ≥ T minutes before event (T = 3–10 min typical).
- Prediction F1-score ≥ 0.85.

Schedule adherence / on-time performance (h):

- % buses arriving within ±2 min of adjusted timetable ≥ 85% after dynamic adjustments.
- Average passenger waiting time reduced by X% (baseline → target e.g., 25%).

Safety detection performance (i):

- True Positive Rate for unsafe driving events ≥ 95%; False Alarm Rate ≤ 5%.
- Detection latency ≤ 1s for sudden braking events.

Passenger information freshness (j):

- Real-time updates latency ≤ 5 s; availability ≥ 99.5%.

Operational cost / resource constraints:

- Additional vehicle-hours due to dynamic rescheduling ≤ specified budget threshold.

Human-in-the-loop constraints:

- % of automated schedule changes requiring dispatcher approval (configurable, e.g., 10%).

Secondary / aggregate KPIs:

- Overall passenger satisfaction score (survey) ↑
- Reduction in crowding incidents per route per month
- Reduction in emergency braking / accident-related incidents

2) Environment (E) — where the agent operates

The environment is complex, partially observable, dynamic, stochastic, multi-agent.

Characteristics:

- **Partially observable:** sensors (cameras, GPS, traffic feed) give incomplete/noisy info (occlusions, bad weather).
- **Dynamic & stochastic:** traffic, passenger arrival patterns, road incidents evolve over time and involve randomness.
- **Multi-agent:** many buses, drivers, passengers, other vehicles, traffic-control systems interact.
- **Continuous & discrete aspects:** continuous (vehicle speed, passenger counts), discrete (stop IDs, bus IDs, schedule slots).
- **Real-world constraints:** regulatory rules, safety limits, physical capacity (bus seating/standing), network latency, privacy laws.

Relevant to requirements:

- (f) Video streams affected by lighting/occlusion → must handle partial observability.
- (g) Predictions must account for stochastic passenger arrivals and upstream delays.
- (h) Dynamic scheduling must respect traffic patterns and regulatory constraints (e.g., driver shift limits).
- (i) Unsafe driving detection must tolerate sensor noise and false positives.
- (j) Passenger updates must be reliable across heterogeneous devices/networks.

3) Sensors (S) — what the agent perceives

List sensors / data sources and map them to the requirements (f–j).

Primary sensors / data sources

- CCTV / onboard & stop-mounted video cameras (live feeds) → (f) Detect passenger flow
  - Use computer-vision pipelines (counting, tracking, pose) and edge inference.
- **Automatic Passenger Counters (APC) / door sensors** → ground truth for boarding/alighting counts (fusion with video). Supports (f) and (g).
- **GPS & vehicle telematics (speed, acceleration, heading)** → supports (g) overcrowding inference (via delays), (h) traffic-aware rescheduling, and (i) unsafe driving detection (sudden braking, speeding, harsh turns).
- Traffic feeds / third-party APIs (e.g., municipal traffic sensors, road events, congestion indices) → (h) dynamic schedule adaptation.
- Mobile-app / ticketing system data (smartcard taps, mobile app check-ins, reservations) → passenger demand signal for (f), (g), (j).
- **Road/weather sensors / meteorological APIs** → used to adjust predictions (rain reduces boarding rates; affects (g) & (h)).
- **Driver input / manual reports** (via driver app) → corroborates unsafe events (i) and incidents.
- **Control-center dispatch data** (current bus assignments, driver shifts) → needed for (h).
- **Passenger devices & beacons (optional)** → location pings for better (j) arrival estimates.

Sensor fusion approach: combine video + APC + ticketing + GPS to increase robustness and reduce single-sensor failure modes.

4) Actuators (A) — actions the agent can take

Actuators are both **direct controls** and **communication actions** (human-in-the-loop and automated).

Direct / automated actions

- **Dynamic schedule adjustments / dispatch commands** → add/remove buses, change headway on route, deploy reserve vehicles (h). (Executed by dispatch system, not autonomous vehicle control.)
- **In-vehicle alerts to drivers** → auditory/visual warnings for detected unsafe behavior (i). Example: "Reduce speed — hard braking detected." These are advisories, not control.
- **Control-center notifications & escalation** → send incident alerts, suggested route diversions, or stop closures to dispatch/ops (i,h).
- Passenger-facing updates:
  - Mobile-app push notifications, SMS, electronic stop displays, web APIs for real-time ETAs (j).
  - Overcrowding warnings to passengers: "Next bus less crowded in X min" (g,j).
- **Automated gating / access control (where applicable):** restrict boarding if overcrowded and redirect passengers (policy-dependent) (g).
- Traffic signal priority requests (if integrated with city control): request priority at intersections to reduce delay (h).
- **Data logging / audit trails & reporting:** store events for offline analysis (safety/regulatory).

Human-in-loop actuators

- **Dispatcher console commands** to re-route or reschedule based on system suggestions.
- **Driver override** — driver must accept/acknowledge system advisories; system must not take direct vehicle control (unless policy/agreement allows).

5) Mapping PEAS → Scenario requirements (f)–(j)

For clarity, I map each requirement directly:

(f) Detect passenger flow at different stops using live video feeds

- **Sensors:** stop-mounted cameras, APC, ticketing logs.
- **Inference:** CV models (people detection, tracking, re-identification) + temporal aggregation to estimate counts and flow rates.
- **Actuators:** update occupancy models; surface info to dispatch and passenger apps.
- **Performance measures:** accuracy, latency, robustness to occlusion/weather.

(g) Predict bus overcrowding before it happens

- **Sensors:** historical ridership, live counts, GPS delays upstream, weather, special-event calendar.
- **Inference:** short-term demand forecasting (time-series + ML — e.g., LSTM, probabilistic models) that outputs probability of overcrowding per bus in next T minutes.
- **Actuators:** recommend/dynamically insert additional bus; notify passengers; limit boarding.
- **Performance metrics:** lead-time, prediction precision/recall, reduction in overcrowding incidents.

(h) Adjust bus schedules dynamically based on traffic patterns

- **Sensors:** GPS, traffic feeds.
- **Inference:** ETA prediction models and schedule-optimization module (constrained optimization considering driver shifts, fleet size, passenger wait cost).
- **Actuators:** dispatch commands, reschedule notifications, request signal priority.
- **Performance metrics:** schedule adherence, average waiting time, operational cost.

(i) Notify drivers and control center when unsafe driving behavior is detected

- **Sensors:** vehicle IMU (accel/gyro), speedometer, GPS, dash cam video.
- **Inference:** event detection models (threshold-based and ML classifiers) for harsh braking, speeding, lane departure.
- **Actuators:** immediate in-cab alert; control-center alert; post-incident log for training/discipline.
- **Performance metrics:** detection TPR/FPR, false-alarm rate, time-to-alert.

(j) Provide real-time updates to passengers waiting at bus stops

- **Sensors:** GPS + schedule status + passenger flow.
- **Inference:** live ETA computation; predictive arrival considering boarding delays.
- **Actuators:** update mobile app, electronic stop displays, SMS.
- **Performance metrics:** update latency, ETA accuracy (mean absolute error), uptime.

6) System architecture & inference flow (concise)

- **Edge layer:** do lightweight CV (person count) on cameras (edge TPU) to reduce bandwidth and ensure low latency for detection (f).
- **Communication layer:** secure message bus (MQTT / WebSockets) for telemetry.
- Central inference layer:
  - **Real-time stream processors** (detect events, fuse sensors)
  - **Prediction models** (short-term demand forecasting, overcrowding probability)
  - **Decision module / optimizer** (generates schedule adjustments subject to constraints)
- **Action layer:** dispatch API, driver app, passenger API, digital signage.
- **Audit & offline analytics:** for model retraining and safety audits.

7) Constraints, safety, privacy & operational considerations

- **Human authority:** never autonomously control vehicle movement — system should be advisory unless explicitly authorized; drivers/dispatch must be able to override.
- **Privacy:** video processing should redact/avoid storing identifiable faces unless needed; comply with local privacy laws (GDPR-like). Prefer edge processing and only transmit counts/aggregates.
- **Latency vs accuracy trade-offs:** edge CV for low latency but possibly lower accuracy; central models for high accuracy but higher latency. Decide based on safety-criticality.
- **Robustness to failure:** graceful degradation — if video fails, fall back to APC and ticketing.
- **Explainability:** maintain logs and explanation facility for safety incidents (why alert raised).
- **Model monitoring:** drift detection, re-training pipelines, SLA metrics.

8) Concrete example KPIs (for reporting)

- Passenger counts per stop: hourly RMSE < 5 passengers.
- Overcrowding predictions: lead time ≥ 5 min and precision ≥ 0.85.
- Unsafe driving alerts: TPR ≥ 0.95; FPR ≤ 0.05.
- ETA error for passengers: MAE ≤ 30 seconds on average per stop in non-congested conditions.
- System uptime: 99.5% for passenger app APIs.

Question #4	**[CO-1 – C2] [SO-1]**	**(Marks: 2x10 = 20)**

4. Demonstrate your understanding of heuristic-based informed search by answering the following:

4.1. What is a heuristic function in Artificial Intelligence? How does it guide informed search strategies towards the goal more efficiently than uninformed search?

A heuristic function (h(n)) is an estimate of the cost, distance, or effort required to reach the goal state from the current state n. It does not guarantee accuracy, but provides a *directional guess* to speed up the search. Informed search algorithms (such as Best-First Search, A*, Hill Climbing) use this heuristic to prioritize nodes that appear closest to the goal.

How it improves efficiency over uninformed search
Uninformed search (BFS, DFS, UCS):

- Does *not* know anything about how close a state is to the goal.
- Expands states blindly, potentially exploring huge, irrelevant portions of the search space.

Informed search (A*, Greedy Best-First):

- Uses heuristics to rank and choose the most promising nodes first.
- Reduces the number of expansions dramatically.
- Moves toward the goal in a more intelligent and directed manner.

4.2. Describe two important characteristics of heuristic search.

i. A heuristic is **admissible** if it *never overestimates* the true cost to reach the goal:

$$h(n) \leq h^*(n)$$

Admissible heuristics ensure that algorithms like **A*** will always find an **optimal** solution when one exists.

ii. A heuristic is **consistent** if it satisfies:

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

This means the heuristic's estimate is always "smooth," never jumping unrealistically between states. Consistency ensures:

- No re-expansion of nodes is needed
- Algorithms remain efficient and optimal

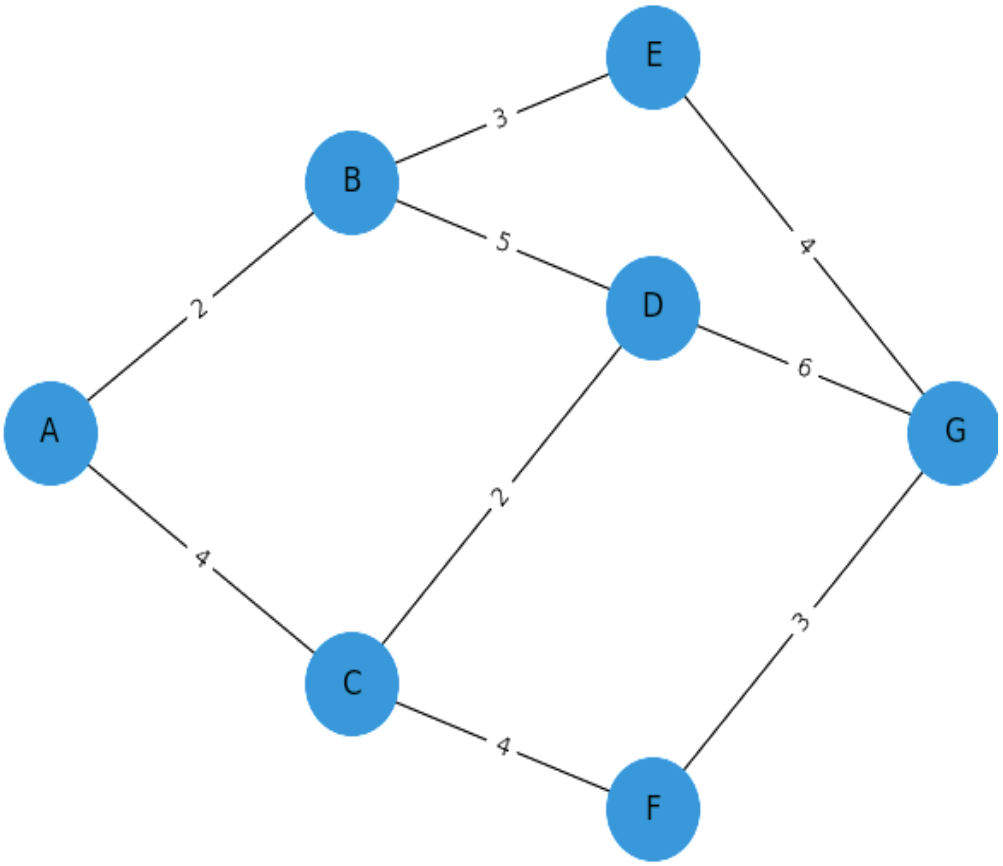
4.3. Mention two real-world applications where heuristic search (such as A*) is commonly used and briefly explain why heuristics are useful there.

Application 1: GPS Navigation / Route Finding
Systems like Google Maps, Waze, and vehicle navigation use **A*** to compute shortest or fastest routes. The heuristic is often the **straight-line distance** (Euclidean or Manhattan distance) to the destination.
Why heuristics help: They guide the algorithm toward roads that bring the user closer to the destination, avoiding unnecessary exploration of distant paths.

Application 2: Game Playing (Chess, Checkers, Go)
Game AI uses **heuristic evaluation functions** to estimate the quality of a board state when the full search tree is too large.
Why heuristics help: They approximate winning likelihood and help the AI choose strong moves without exploring all possible game states (which would be computationally impossible).

b) Consider the following graph, where each edge label represents the step cost. The start node is A and the goal node is G.

Using your understanding of informed search strategies, apply the A* search algorithm on the given graph to find the optimal path from A to G. Clearly show the important steps of A* [values of g(n), h(n), and f(n) = g(n)+h(n)] and write the final optimal path and its total cost.



Heuristic Knowledge

Node → G	h(n)
A → G	10
B → G	8
C → G	5
D → G	7
E → G	3
F → G	6
G → G	0

Solution:

A* is one of the most popular informed search algorithms in Artificial Intelligence. It combines the strengths of Uniform Cost Search (UCS) and Best-First Search (BFS). It uses both actual cost and heuristic estimate to find the optimal (shortest) path efficiently.

A* evaluates each node using the formula:

$$\boldsymbol{f(n) = g(n) + h(n)}$$

Where:

- **g(n)** = cost from **start** node to **current** node
- **h(n)** = estimated cost (heuristic) from **current** node to **goal**
- **f(n)** = total estimated cost of the path through node *n*

A* always expands the node with the **lowest f(n)** value.

Step-by-Step Working

1. Start at the **initial node**.
2. For every neighbor node, calculate:
   $$f(n) = g(n) + h(n)$$
3. Choose the node with the **smallest f(n)**.
4. Repeat until the **goal node** is reached.
5. Keep track of **visited nodes** to avoid loops.

A* = Best balance between *speed* and *accuracy*.

It **uses both past cost and future estimate** to make the most intelligent decision at each step.

Route 1:

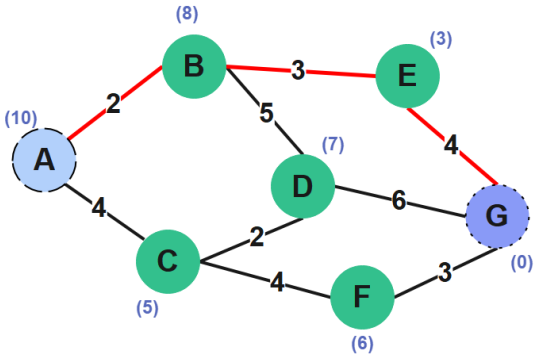
$f(n) = g(n) + h(n)$

$A \rightarrow B = (2) + (8) = 10$

$A \rightarrow B \rightarrow E = (2+3) + (3) = 8$

$A \rightarrow B \rightarrow E \rightarrow G = (2+3+4) + (0) = 9$

$A \rightarrow B \rightarrow E \rightarrow G = 9$



Route 2:

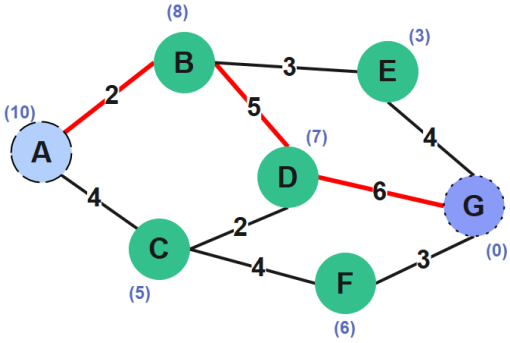
$f(n) = g(n) + h(n)$

$A \rightarrow B = (2) + (8) = 10$

$A \rightarrow B \rightarrow D = (2+5) + (7) = 14$

$A \rightarrow B \rightarrow D \rightarrow G = (2+5+6) + (0) = 13$

$A \rightarrow B \rightarrow D \rightarrow G = 13$



Route 3:

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

$A \rightarrow B = (2) + (8) = 10$

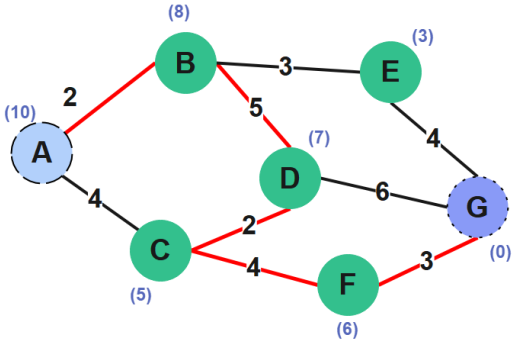
$A \rightarrow B \rightarrow D = (2+5) + (7) = 14$

$A \rightarrow B \rightarrow D \rightarrow C = (2+5+2) + (5) = 16$

$A \rightarrow B \rightarrow D \rightarrow C \rightarrow F = (2+5+2+4) + (6) = 19$

$A \rightarrow B \rightarrow D \rightarrow C \rightarrow F \rightarrow G = (2+5+2+4+3) + (0) = 16$

$A \rightarrow B \rightarrow D \rightarrow C \rightarrow F \rightarrow G = 16$



Route 4:

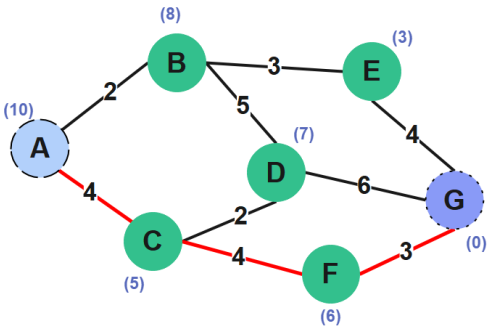
$f(n) = g(n) + h(n)$

$A \to C = (4) + (5) = 10$

$A \to C \to F = (4+4) + (6) = 14$

$A \to C \to F \to G = (4+4+3) + (0) = 11$

$A \to C \to F \to G = 11$



Route 5:

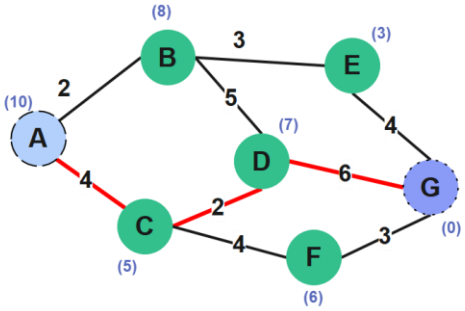
$f(n) = g(n) + h(n)$

$A \to C = (4) + (5) = 10$

$A \to C \to D = (4+2) + (7) = 13$

$A \to C \to D \to G = (4+2+6) + (0) = 12$

$A \to C \to D \to G = 12$



Route 6:

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

$A \to C = (4) + (5) = 10$

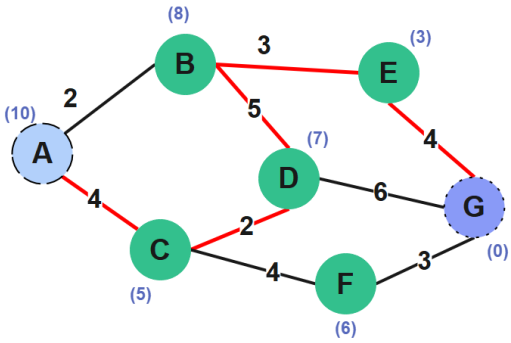
$A \to C \to D = (4+2) + (7) = 13$

$A \to C \to D \to B = (4+2+5) + (8) = 19$

$A \to C \to D \to B \to E = (4+2+5+3) + (3) = 17$

$A \to C \to D \to B \to E \to G = (4+2+5+3+4) + (0) = 18$

$A \to C \to D \to B \to E \to G = 18$



So, the Best Optimized path having lowest value is:

Rout 1: $A \to B \to E \to G = 9$