SCHEME AND SOLUTION

Internal Assessment Test I
NOVEMBER 2024

Sub:	Artificial Intelligence					Sub Code:	BCS515B	Branch:	ISE		
Date:	08-11-2024	Duration:	90 min's	Max Marks:	50	Sem/Sec:	V- A,B,C			OBE	
									MARKS	CO	RBT
1	a. Explain all four different approaches to AI in detail. **1. Thinking Humanly: The Cognitive Modeling Approach** • **Goal**: This approach aims to create systems that can think like humans by simulating human cognitive processes. • **Focus**: Cognitive scientists use this approach to build AI models that mirror how the human brain processes information, often taking inspiration from psychology and neuroscience. • **Example**: Cognitive modeling often involves developing algorithms that simulate tasks humans perform, like memory recall or perception, and might include neural networks that emulate brain processes. • **Challenge**: Since human thought processes are complex and not fully understood, creating accurate cognitive models can be difficult and resource-intensive. **2. Thinking Rationally: The Laws of Thought Approach** • **Goal**: This approach seeks to create systems that make logically correct decisions, relying on formal rules and logical reasoning. • **Focus**: Based on logical deduction and the idea that intelligence involves following rules of reasoning, this approach uses principles of logic to define AI. • **Example**: Early AI research focused on building systems that apply formal logic, such as theorem provers, which are designed to solve problems by applying logical rules. • **Challenge**: Formalizing human knowledge and reasoning into precise logic statements is difficult, and real-world problems are often too complex to be solved purely by logic. **3. Acting Humanly: The Turing Test Approach** • **Goal**: The objective here is to create machines that can act in ways indistinguishable from human behavior. • **Focus**: This approach, inspired by the Turing Test proposed by Alan Turing, evaluates a machine's ability to exhibit behavior that, to an observer, appears human. • **Example**: Chatbots and conversational AI are examples, as they try to mimic human conversation and interactions. • **Challenge**: While acting humanly doesn't require understanding or thought in the human sense, it's challenging to create machines capable of human-like responses to a broad range of unpredictable scenarios. **4. Acting Rationally: The Rational Agent Approach** • **Goal**: This approach aims to develop systems that act rationally, meaning they perform actions that maximize their chances of success. • **Focus**: A rational agent takes actions based on information available and expected outcomes, aiming for the highest utility or benefit. • **Example**: Self-driving cars are designed to act rationally, making decisions based on a mix of real-time data, safety concerns, and efficiency.								6+4	CO1	L1

- **Challenge**: Defining what is "rational" or "optimal" can be complex, especially in dynamic and uncertain environments.

b. Explain different properties of task environment.

Fully Observable vs. Partially Observable

- **Fully Observable**: In a fully observable environment, the agent has complete information about the state of the environment at each point in time. This means the agent can access all relevant data needed for decision-making without uncertainty.
- **Partially Observable**: In a partially observable environment, the agent has limited information, often due to noisy or incomplete sensor data. This requires the agent to make decisions based on partial knowledge, possibly predicting or inferring missing details.

Example: Chess is fully observable (all pieces and positions are visible), while driving a car in fog is partially observable (limited visibility affects the agent's perception).

2. Deterministic vs. Stochastic

- **Deterministic**: In a deterministic environment, any action taken by the agent has a predictable outcome with no randomness involved. The agent's actions always result in expected results, making planning straightforward.
- **Stochastic**: In a stochastic environment, actions can lead to different outcomes due to randomness or uncertainty, which makes the environment unpredictable. Here, the agent might need strategies to handle variability and incorporate probabilities.

Example: Solving a mathematical puzzle is deterministic, while dealing with traffic is stochastic due to the unpredictable behavior of other drivers.

3. Episodic vs. Sequential

- **Episodic**: In an episodic environment, the agent's actions are divided into separate, independent episodes. Each episode does not depend on the previous one, so the agent doesn't need to consider the past when making decisions.
- **Sequential**: In a sequential environment, current actions affect future ones. The agent must consider the sequence of actions and the resulting state after each action.

Example: Image recognition tasks are episodic (each image can be classified independently), while chess is sequential since each move impacts future moves and the overall game outcome.

4. Static vs. Dynamic

- **Static**: A static environment remains unchanged while the agent is deliberating. The agent doesn't need to worry about the environment changing between decision-making steps.
- **Dynamic**: In a dynamic environment, the environment can change while the agent is choosing or executing an action, requiring it to react quickly or adapt its strategy continuously.

Example: Solving a crossword puzzle is static (no changes occur as you think about the answers), while driving is dynamic, as road conditions and other vehicles change in real-time.

5. Discrete vs. Continuous

- **Discrete**: In a discrete environment, there are a finite number of distinct states, actions, or time intervals. The agent's actions and decisions are taken in fixed steps or increments.
- **Continuous**: A continuous environment has a vast or infinite number of possible states or time intervals, requiring the agent to operate in a fluid, real-time manner.

		Example: A turn-based game like chess is discrete, while controlling a robotic arm (which moves in smooth, continuous motions) is continuous. **6. Single Agent vs. Multiagent** • **Single Agent**: In a single-agent environment, the agent is the only one acting within the environment, so it doesn't need to consider the influence of other agents' actions. • **Multiagent**: In a multiagent environment, multiple agents interact, either as competitors or collaborators. Agents need to strategize based on other agents' actions, often involving cooperation, competition, or negotiation. **Example**: Playing a solo puzzle game is single-agent, while a multiplayer game like soccer is multiagent, as each player's actions affect others.			
2	a.	What are PEAS descriptors? Give PEAS descriptors for some common AI applications. **PEAS** descriptors (Performance measure, Environment, Actuators, and Sensors) are used to define the specifications and requirements of an intelligent agent for a particular task. **Performance Measure**: Defines the criteria to evaluate the success or efficiency of the agent. It specifies what constitutes a "good" or "successful" outcome for the agent's task. **Environment**: Refers to the surroundings or context within which the agent operates. This includes all external factors and conditions that may affect the agent's behavior. **Actuators**: The components through which the agent interacts with the environment. Actuators allow the agent to perform actions or make changes to its surroundings. **Sensors**: The means by which the agent perceives its environment. Sensors gather information from the surroundings, which the agent uses to make decisions. **<u>A list of AI applications with their respective PEAS descriptors:</u>** 1. **Self-Driving Car** 2. **Household Cleaning Robot** 3. **Spam Email Filter** 4. **Autonomous Drone for Surveillance** 5. **Personal Voice Assistant (e.g., Siri, Alexa)**	5+5	CO1	L1

	b.	Differentiate the environment types : i)Fully observable Vs partially observable ii)Singe agent Vs Multiagent iii)Deterministic Vs stochastic iv) Static Vs Dynamic
		i) Fully Observable vs. Partially Observable
		• **Fully Observable**: The agent has complete information about the environment at all times. All relevant aspects of the environment's current state are available for the agent to perceive.
		  ○ **Example**: Chess, where the agent can see the entire board and all pieces.
		• **Partially Observable**: The agent has limited information about the environment, often due to noisy sensors or restricted visibility. The agent must make decisions based on incomplete data.
		  ○ **Example**: Driving a car in fog, where visibility is reduced, and some information (like distant obstacles) is not immediately perceivable.
		--- 	
		ii) Single Agent vs. Multiagent
		• **Single Agent**: The environment contains only one agent, so it doesn't need to consider the actions or decisions of other agents.			

	◦ **Example**: A robot vacuum cleaner in a home, where it is the only decision-making agent. • **Multiagent**: The environment includes multiple agents, which may cooperate, compete, or interact in complex ways. The agent must consider other agents' actions and may need strategies to account for these interactions. ◦ **Example**: A game of soccer, where players (agents) must anticipate and respond to opponents and teammates. --- **iii) Deterministic vs. Stochastic** • **Deterministic**: The outcome of any action is predictable and has no uncertainty. Each action leads to a single, expected result, making it easier to plan. ◦ **Example**: Solving a mathematical problem, where each step has a known, predictable outcome. • **Stochastic**: Actions lead to uncertain outcomes due to randomness or unpredictability. The agent needs to handle variability and might use probabilities to predict possible results. ◦ **Example**: Weather forecasting, where various factors introduce uncertainty, leading to probabilistic predictions. --- **iv) Static vs. Dynamic** • **Static**: The environment remains unchanged while the agent is deliberating. It doesn't evolve or alter independently of the agent's actions, making it simpler for planning. ◦ **Example**: A crossword puzzle, where the puzzle does not change as the agent thinks about solutions. • **Dynamic**: The environment can change while the agent is in the process of making or executing a decision, often requiring the agent to adapt in real-time. ◦ **Example**: A self-driving car navigating through traffic, where the environment (other vehicles, pedestrians) changes continuously.			
3	Provide a step-by-step illustration of how breadth-first search[BFS] works with example and pseudo code Breadth-First Search (BFS) is a search algorithm used for traversing or searching tree or graph data structures. It explores all nodes at the present depth level before moving on to nodes at the next depth level. BFS is typically implemented using a queue to keep track of nodes to visit next.	10	CO2	L2

When all actions have the same cost, an appropriate strategy is **breadth-first search**, in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on. This is a systematic search strategy that is therefore complete even on infinite state spaces. We could implement breadth-first search as a call to BEST-FIRST-SEARCH where the evaluation function $f(n)$ is the depth of the node—that is, the number of actions it takes to reach the node.

However, we can get additional efficiency with a couple of tricks. A first-in-first-out queue will be faster than a priority queue, and will give us the correct order of nodes: 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. In addition, *reached* can be a set of states rather than a mapping from states to nodes, because once we've reached a state, we can never find a better path to the state. That also means we can do an **early goal test**, checking whether a node is a solution as soon as it is *generated*, rather than the **late goal test** that best-first search uses, waiting until a node is popped off the queue. Figure 3.8 shows the progress of a breadth-first search on a binary tree, and Figure 3.9 shows the algorithm with the early-goal efficiency enhancements.

Breadth-first search always finds a solution with a minimal number of actions, because when it is generating nodes at depth d, it has already generated all the nodes at depth $d-1$, so if one of them were a solution, it would have been found. That means it is cost-optimal

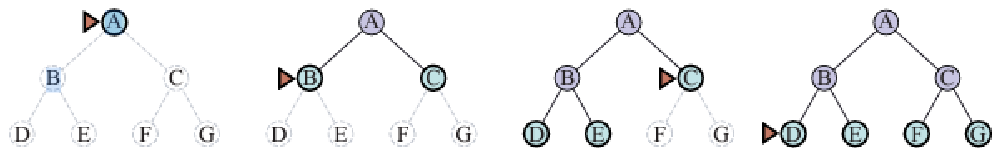


Figure 3.8 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

for problems where all actions have the same cost, but not for problems that don't have that property. It is complete in either case. In terms of time and space, imagine searching a uniform tree where every state has b successors. The root of the search tree generates b nodes, each of which generates b more nodes, for a total of b^2 at the second level. Each of these generates b more nodes, yielding b^3 nodes at the third level, and so on. Now suppose that the solution is at depth d. Then the total number of nodes generated is

$$1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

All the nodes remain in memory, so both time and space complexity are $O(b^d)$. Exponential bounds like that are scary. As a typical real-world example, consider a problem with branching factor $b = 10$, processing speed 1 million nodes/second, and memory requirements of 1 Kbyte/node. A search to depth $d = 10$ would take less than 3 hours, but would require 10 terabytes of memory. *The memory requirements are a bigger problem for breadth-first search than the execution time.* But time is still an important factor. At depth $d = 14$, even with infinite memory, the search would take 3.5 years. In general, *exponential-complexity search problems cannot be solved by uninformed search for any but the smallest instances.*

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

4	a. Explain the components and architecture of a problem-solving agent.	4+6	CO2	L3

- A **problem-solving agent** is a *goal-based agent* and use *atomic representations.*
  - In atomic representations, states of the world are considered as wholes, with no internal structure vi the problem solving algorithms.
- *Intelligent agents* are supposed to maximize their *performance measure*. Achieving this is som simplified if the agent can adopt a **goal** and aim at satisfying it.
- **Problem formulation** is the process of deciding what actions and states to consider, given a *g*
- The process of looking for a sequence of actions that reaches the goal is called **search**.
- A ***search algorithm*** takes a problem as input and returns a **solution** in the form of an action sequence.
- Once a *solution* is found, the carrying actions it recommends is called the **execution phase**.

- A problem-solving agent has three phases:
  - **problem formulation, searching solution and executing actions in the solution.**

A **problem** can be defined by five components:
- **initial state, actions, transition model, goal test, path cost.**

INITIAL STATE: The **initial state** that the agent starts in.

ACTIONS: A description of the possible **actions** available to the agent.

- Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s.
- Each of these actions is **applicable** in s.

TRANSITION MODEL: A description of what each action does is known as the **transition mod**

- A function RESULT(s,a) that returns the state that results from doing action a in state s.
- The term **successor** to refer to any state reachable from a given state by a single
- The **state space** of the problem is the set of all states reachable from the *initial state* by any sequ of actions.
- The *state space* forms a **graph** in which the nodes are states and the links between nodes are acti
- A **path** in the state space is a sequence of states connected by a sequence of actions.

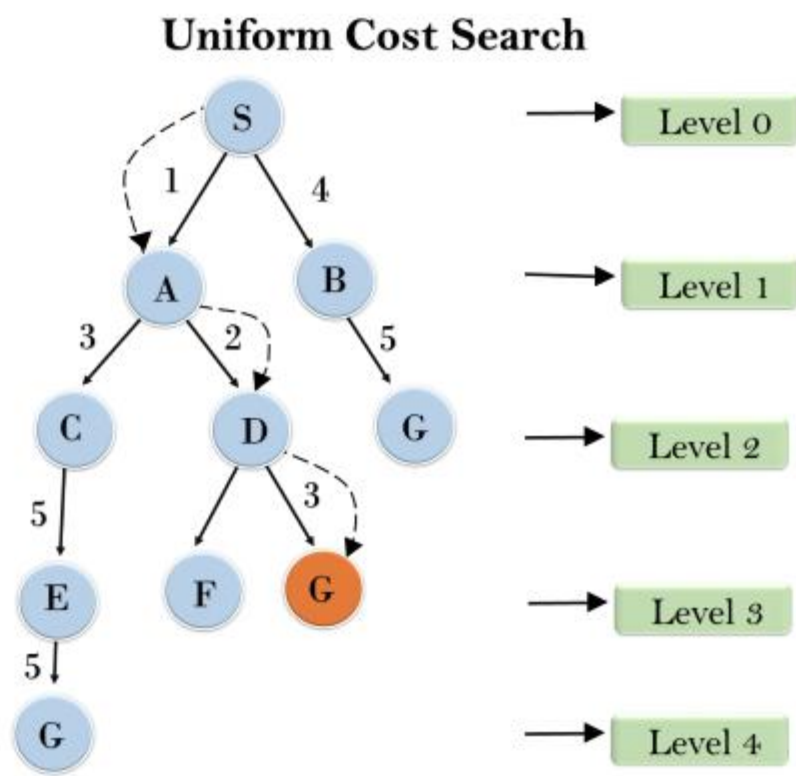
b. Identify situations where uniform cost search performs better than other algorithms and explain with example.

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

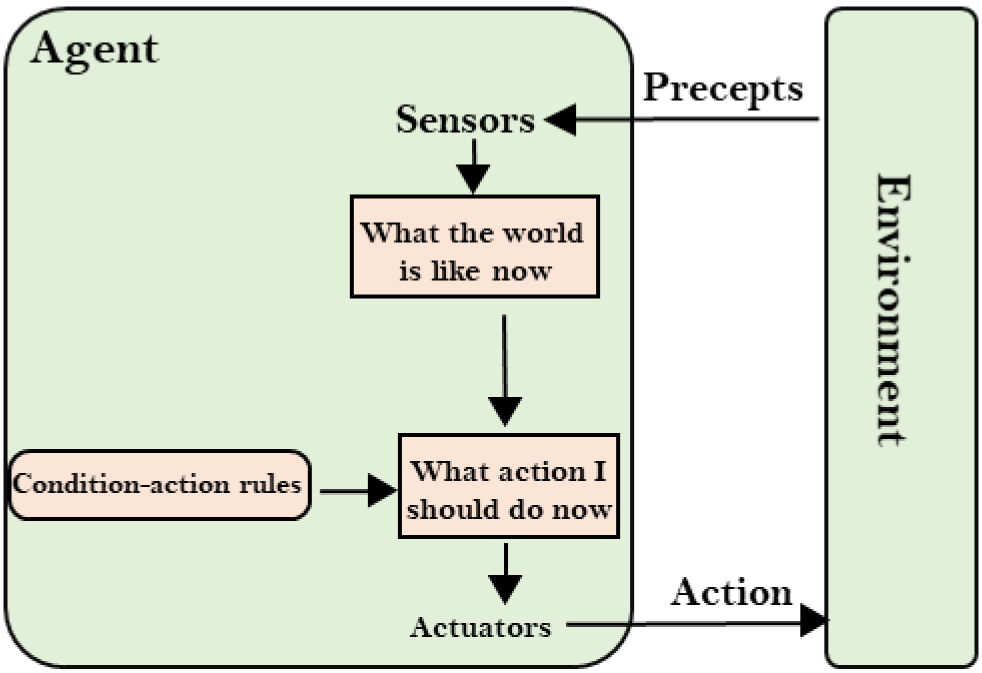
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.
- It is an efficient when the edge weights are small, as it explores the paths in an order that ensures that the shortest path is found early.
- It's a fundamental search method that is not overly complex, making it accessible for many users.
- It is a type of comprehensive algorithm that will find a solution if one exists. This means the algorithm is complete, ensuring it can locate a solution whenever a viable one is available. The algorithm covers all the necessary steps to arrive at a resolution.

Disadvantages:

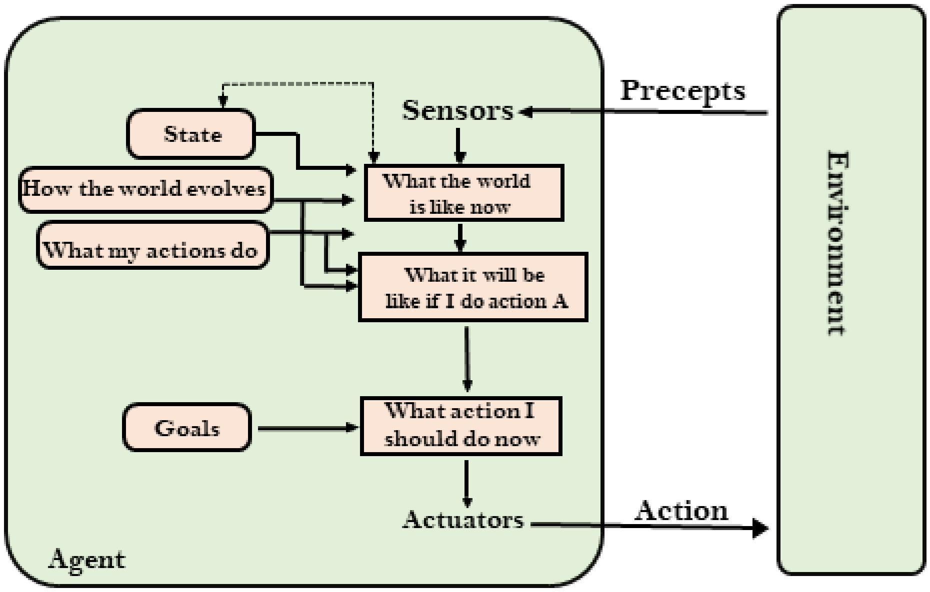
	- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop. - When in operation, UCS shall know all the edge weights to start off the search. - This search holds constant the list of the nodes that it has already discovered in a priority queue. Such is a much weightier thing if you have a large graph. Algorithm allocates the memory by storing the path sequence of prioritizes, which can be memory intensive as the graph gets larger.With the help of Uniform cost search we can end up with the problem if the graph has edge's cycles with smaller cost than that of the shortest path. - The Uniform cost search will keep deploying priority queue so that the paths explored can be stored in any case as the graph size can be even bigger that can eventually result in too much memory being used. Example:  			
5	List types of agents. Differentiate between simple reflex agents and goal-based agents in the context of problem-solving. gents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below: - Simple Reflex Agent - Model-based reflex agent - Goal-based agents - Utility-based agent - Learning agent 1. Simple Reflex agent: - The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history. - These agents only succeed in the fully observable environment.	10	CO1	L2

- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
  - They have very limited intelligence
  - They do not have knowledge of non-perceptual parts of the current state
  - Mostly too big to generate and to store.
  - Not adaptive to changes in the environment.



2. Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.

<table>
<tr><td></td><td>
</td><td></td><td></td><td></td></tr>
<tr><td>6</td><td>Evaluate the performance of BFS and DFS for different performance measures.
The performance of tree search strategies is often evaluated based on several key criteria:
1. Completeness
2. Optimality
3. Time Complexity
4. Space Complexity
These criteria help determine the effectiveness of each search strategy based on the nature of the problem and the properties of the graph or tree being explored.
<table>
<tr><th>Criteira</th><th>BFS</th><th>DFS</th></tr>
<tr><td>Completeness</td><td>Complete for finite graphs</td><td>Incomplete for infinite graphs</td></tr>
<tr><td>Optimality</td><td>Optimal for unweighted graphs</td><td>Not optimal</td></tr>
<tr><td>Time Complexity</td><td>O(b^d)</td><td>O(b^m)</td></tr>
<tr><td>Space Complexity</td><td>O(b^d)</td><td>O(b * m)</td></tr>
</table>
• b = branching factor
• d = depth of the shallowest solution
• m = maximum depth of the search tree</td><td>10</td><td>CO2</td><td>L3</td></tr>
</table>