Unit 2: Problem Solving by Search - Theory Answers

1. What is Search Problem? How it is represented using State space model?

A **search problem** in Artificial Intelligence refers to the process of navigating through a **search space** to find a **goal state** from an **initial state**, following specific rules and constraints. Many real-world problems like navigation, puzzle solving, and game playing can be represented and solved as search problems.

Search Problem Components:

- **Initial State:** The starting point of the problem.
- **Goal State:** The target state we aim to reach.
- **Operators:** The set of actions that can be applied to move between states.
- **Search Space:** All the possible states that can be generated by applying operators starting from the initial state.
- **Goal Test:** A way to determine if a given state is a goal state.

State Space Representation: In AI, the search problem is often represented using a **state space model**. It provides a formal description of all possible configurations (states) of the problem.

Key Aspects:

- The state space is defined by all valid states that can be reached from the initial state by applying a set of operators.
- A path from the initial to goal state consists of a sequence of states where each state is derived by applying a rule/operator.

Example - Chess Game:

- **Initial State:** Standard chess setup.
- **Goal State:** Checkmate or draw.
- **Operators:** Legal moves of chess pieces.
- **State Space:** All possible board configurations resulting from legal moves.

Advantages:

- Provides a structured approach to problem solving.
- Helps in applying various search algorithms to find optimal or feasible solutions.

(Refer to slides 2–10 for state space model and examples.)

2. Describe Uninformed Search Algorithms.

Uninformed Search, also known as **Blind Search**, is a category of search algorithms that explore the search space without any domain-specific knowledge. These algorithms do not have information about the goal location beyond the definition of the problem and goal test.

Types of Uninformed Search Algorithms:

1. **Breadth-First Search (BFS):**
    - Explores all nodes at the present depth before moving on to the nodes at the next level.
    - Uses FIFO queue.
    - Guarantees finding the shortest path (minimal steps).
    - **Pros:** Complete, optimal (for unweighted graphs).
    - **Cons:** High memory and time consumption.
2. **Depth-First Search (DFS):**
    - Explores as far as possible along each branch before backtracking.
    - Uses stack (LIFO) or recursion.
    - **Pros:** Less memory, simple to implement.
    - **Cons:** Can get stuck in loops, not guaranteed to find optimal solution.
3. **Iterative Deepening Depth-First Search (IDDFS):**
    - Combines BFS's completeness and DFS's memory efficiency.
    - Repeats DFS to increasing depth limits.
    - **Pros:** Complete, optimal (like BFS), less memory.
    - **Cons:** Repeats previous efforts multiple times.
4. **Bidirectional Search:**
    - Simultaneous searches from initial state and goal state until they meet.
    - **Pros:** Reduces search time significantly.
    - **Cons:** Requires goal state to be known; complex implementation.

Uninformed search is powerful in generic problem-solving scenarios but becomes inefficient for large and complex search spaces.

(Refer to slides 13–29.)

3. Explain Alpha-Beta Pruning in Mini-max Algorithm.

Alpha-Beta Pruning is an optimization technique for the **Mini-Max Algorithm**, used in decision-making in two-player games like chess and tic-tac-toe. It reduces the number of nodes evaluated in the game tree, thereby improving efficiency without compromising the optimal solution.

Mini-Max Algorithm Recap:

- Two players: MAX (tries to maximize score) and MIN (tries to minimize score).
- It uses recursive depth-first search to evaluate game outcomes.

Alpha-Beta Pruning Concept:

- Introduces two parameters:
  - **Alpha (α):** Best value for MAX so far.
  - **Beta (β):** Best value for MIN so far.
- If at any point $\alpha \geq \beta$, that subtree is pruned (not explored further), as it won't influence the final decision.

Advantages of Alpha-Beta Pruning:

- Reduces number of nodes evaluated.
- Speeds up computation by eliminating unnecessary branches.
- Same result as Mini-Max but faster.

Working Steps:

1. Traverse the game tree using depth-first search.
2. Maintain α and β values during traversal.
3. Prune the subtree if $\alpha \geq \beta$.

Example:

- At node E: If MAX finds a value higher than MIN's threshold β, further siblings of that node can be ignored.
- Result: Game tree is explored only where necessary.

Alpha-Beta pruning is essential in real-time games and complex search problems to improve algorithm performance.

(Refer to slides 37–44.)

4. What is A* Algorithm? Explain working with example.**

A* is an informed search algorithm widely used for pathfinding and graph traversal. It uses a **heuristic approach** to determine the most promising path, combining the cost to reach a node and the estimated cost to reach the goal from that node.

Formula:

- **f(n) = g(n) + h(n)**
- **g(n):** Actual cost from start node to current node.

- **h(n):** Heuristic estimate from current node to goal.
- **f(n):** Estimated total cost of the cheapest solution through n.

Key Features:

- Finds optimal solutions if the heuristic is admissible (never overestimates).
- Reduces the number of nodes explored compared to uninformed search.

Example Scenario:

- Navigating from city A to city B.
- g(n): distance from A to current city.
- h(n): straight-line distance from current city to B.
- A* selects the path with the lowest combined estimated cost.

Advantages:

- Optimal and complete.
- Very efficient with a good heuristic.

Disadvantages:

- High memory consumption.

Applications:

- GPS navigation systems.
- Game AI.
- Robotics path planning.

A* remains one of the most efficient and widely used algorithms for pathfinding problems due to its balance between accuracy and performance.