Koneru Lakshmaiah Education Foundation
(Category -1, Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Accredited by **NAAC** as '**A++**' Grade University ❖Approved by AICTE ❖ ISO 9001-2015 Certified
Campus: Green Fields, Vaddeswaram - 522 502, Guntur District, Andhra Pradesh, INDIA.
Phone No. 0863 - 2399999; www.klef.ac.in; www.klef.edu.in; www.kluniversity.in
Admin Off: 29-36-38, Museum Road, Governorpet, Vijayawada - 520 002. Ph: +91 - 866 -2577715, Fax: +91-866-2577717.

DEPARTMENT OF AI&DS

23AD2001O – Artificial Intelligence and Machine Learning

2024-25 EVEN SEMESTER

CO-1

HOME ASSIGNMENT

1. **Define agents and compare different types of agents and environments with a suitable example.**

ANSWER

An **agent** is an entity that perceives its environment through sensors and acts based on those perceptions using actuators.

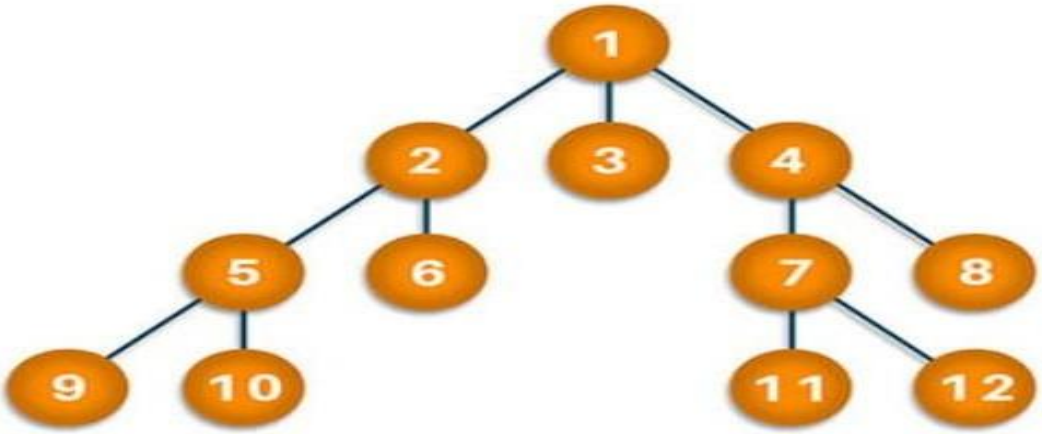
Types of Agents:

1. **Simple Reflex Agents:** Act based on current perceptions (e.g., a thermostat).
2. **Model-based Reflex Agents:** Keep an internal model of the environment (e.g., robot vacuum).
3. **Goal-based Agents:** Act to achieve specific goals (e.g., GPS navigation).
4. **Utility-based Agents:** Maximize satisfaction or utility (e.g., stock trading algorithms).
5. **Learning Agents:** Learn and improve from experience (e.g., Netflix recommendation system).

Types of Environments:

1. **Fully Observable** vs. **Partially Observable** (e.g., chess vs. self-driving car).
2. **Deterministic** vs. **Stochastic** (e.g., robot on flat surface vs. picking up a ball).
3. **Episodic** vs. **Sequential** (e.g., email classification vs. chess).
4. **Static** vs. **Dynamic** (e.g., puzzle game vs. autonomous car).
5. **Discrete** vs. **Continuous** (e.g., Tic-Tac-Toe vs. driving a car).

2. **Given a Graph, show how the search takes place using BFS, DFS and IDDFS Algorithm / Approach. Develop step-by-step execution and explanation of the BFS and DFS algorithms. The explanation must showcase the states of data structures (open, closed, visited, etc.) at each step where node "1" is the initial state and node "11" is the destination state.**



ANSWER

Breadth-First Search (BFS)

BFS Characteristics:

- Explores all neighbors of a node before moving to the next level.
- Uses a queue data structure for traversal.

Steps:

1. **Initialization:**
   - Open (Queue): [1]
   - Visited: {}
   - Path: []
2. **Step 1:** Dequeue `1` (current node). Add its neighbors to the queue.
   - Open: [2, 3, 4]
   - Visited: {1}
   - Path: [1]
3. **Step 2:** Dequeue `2`. Add its neighbors (5, 6).
   - Open: [3, 4, 5, 6]
   - Visited: {1, 2}
   - Path: [1, 2]

4. **Step 3:** Dequeue `3`. No new neighbors to add.
    - Open: [4, 5, 6]
    - Visited: {1, 2, 3}
    - Path: [1, 2, 3]
5. **Step 4:** Dequeue `4`. Add its neighbors (7, 8).
    - Open: [5, 6, 7, 8]
    - Visited: {1, 2, 3, 4}
    - Path: [1, 2, 3, 4]
6. **Step 5:** Dequeue `5`. Add its neighbors (9, 10).
    - Open: [6, 7, 8, 9, 10]
    - Visited: {1, 2, 3, 4, 5}
    - Path: [1, 2, 3, 4, 5]
7. **Step 6:** Dequeue `6`. No new neighbors.
    - Open: [7, 8, 9, 10]
    - Visited: {1, 2, 3, 4, 5, 6}
    - Path: [1, 2, 3, 4, 5, 6]
8. **Step 7:** Dequeue `7`. Add its neighbor (11).
    - Open: [8, 9, 10, 11]
    - Visited: {1, 2, 3, 4, 5, 6, 7}
    - Path: [1, 2, 3, 4, 5, 6, 7]

9. **Step 8:** Dequeue `8`. Add its neighbor (12).
    - Open: [9, 10, 11, 12]
    - Visited: {1, 2, 3, 4, 5, 6, 7, 8}
    - Path: [1, 2, 3, 4, 5, 6, 7, 8]
10. **Step 9:** Dequeue `9`, then `10`. No new neighbors.
    - Open: [11, 12]
    - Visited: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
11. **Step 10:** Dequeue `11` (destination reached).
    - Open: [12]
    - Visited: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
    - Path: [1, 2, 3, 4, 5, 6, 7, 11]

Depth-First Search (DFS)

DFS Characteristics:

- Explores as far as possible along a branch before backtracking.
- Uses a stack data structure for traversal.

Steps:

1. **Initialization:**
   - Open (Stack): [1]
   - Visited: {}
   - Path: []
2. **Step 1:** Pop `1`. Add its neighbors (4, 3, 2) to the stack.
   - Open: [4, 3, 2]
   - Visited: {1}
   - Path: [1]
3. **Step 2:** Pop `2`. Add its neighbors (6, 5).
   - Open: [4, 3, 6, 5]
   - Visited: {1, 2}
   - Path: [1, 2]
4. **Step 3:** Pop `5`. Add its neighbors (10, 9).
   - Open: [4, 3, 6, 10, 9]
   - Visited: {1, 2, 5}
   - Path: [1, 2, 5]

5. **Step 4:** Pop `9`. No new neighbors.
    - Open: [4, 3, 6, 10]
    - Visited: {1, 2, 5, 9}
    - Path: [1, 2, 5, 9]
6. **Step 5:** Pop `10`. No new neighbors.
    - Open: [4, 3, 6]
    - Visited: {1, 2, 5, 9, 10}
    - Path: [1, 2, 5, 9, 10]
7. **Step 6:** Pop `6`. No new neighbors.
    - Open: [4, 3]
    - Visited: {1, 2, 5, 6, 9, 10}
    - Path: [1, 2, 5, 6, 9, 10]
8. **Step 7:** Pop `3`. No new neighbors.
    - Open: [4]
    - Visited: {1, 2, 3, 5, 6, 9, 10}
    - Path: [1, 2, 5, 6, 9, 10, 3]
9. **Step 8:** Pop `4`. Add its neighbors (8, 7).
    - Open: [8, 7]
    - Visited: {1, 2, 3, 4, 5, 6, 9, 10}
    - Path: [1, 2, 5, 6, 9, 10, 3, 4]

10. **Step 9:** Pop `7`. Add its neighbor (11).
    - Open: [8, 11]
    - Visited: {1, 2, 3, 4, 5, 6, 7, 9, 10}
    - Path: [1, 2, 5, 6, 9, 10, 3, 4, 7]
11. **Step 10:** Pop `11` (destination reached).
    - Open: [8]
    - Visited: {1, 2, 3, 4, 5, 6, 7, 9, 10, 11}
    - Path: [1, 2, 5, 6, 9, 10, 3, 4, 7, 11]

Iterative Deepening Depth-First Search (IDDFS)

IDDFS Characteristics:

- Combines the benefits of BFS and DFS by gradually increasing depth limits.
- Repeated DFS for each depth level until the destination is found.

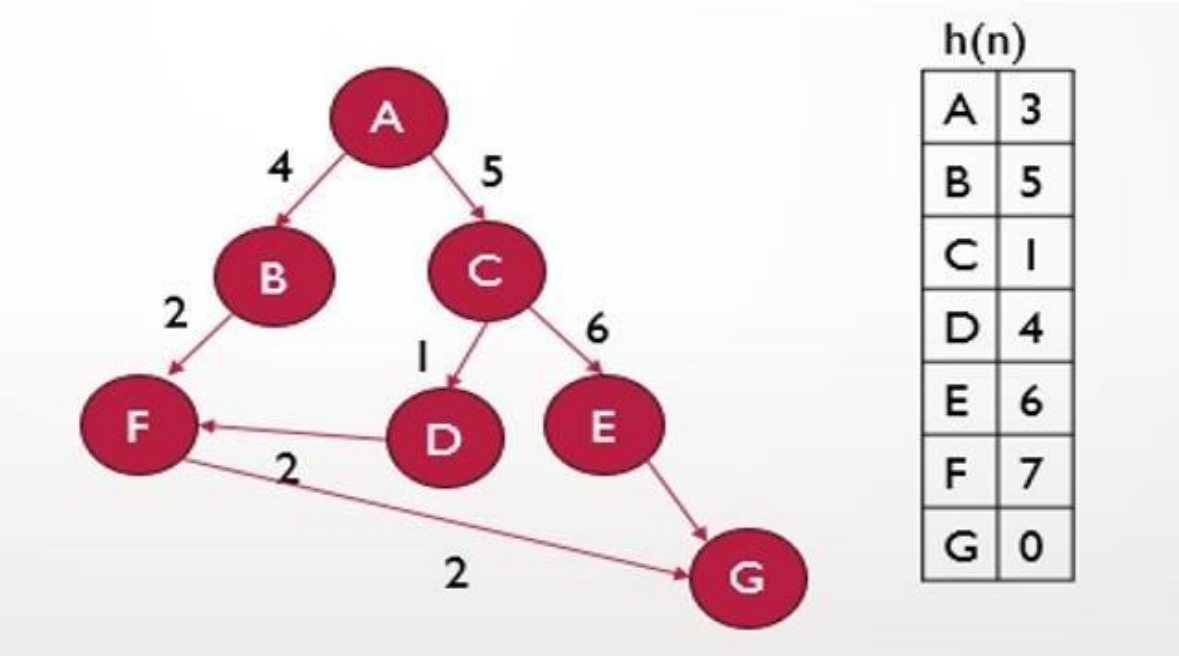
Depth Levels:

1. **Depth 1:** Explore `1`. Destination not reached.
   - Visited: {1}
2. **Depth 2:** Explore `1 → 2 → 3 → 4`. Destination not reached.
   - Visited: {1, 2, 3, 4}
3. **Depth 3:** Explore `1 → 2 → 5 → 6 → 3 → 4 → 7 → 8`. Destination not reached.
   - Visited: {1, 2, 3, 4, 5, 6, 7, 8}
4. **Depth 4:** Explore `1 → 2 → 5 → 9 → 10 → 6 → 3 → 4 → 7 → 11`. Destination reached.

Summary:

- **BFS Path:** [1, 2, 3, 4, 5, 6, 7, 11]
- **DFS Path:** [1, 2, 5, 6, 9, 10, 3, 4, 7, 11]
- **IDDFS:** Matches DFS at the shallowest depth when `11` is reached.

3. Explain A* and Greedy Best First Search algorithms. Find the optimum path within the following graph using both approaches



h(n)	
A	3
B	5
C	I
D	4
E	6
F	7
G	0

ANSWER

Explanation of A* and Greedy Best First Search:

1. **A* Search Algorithm:**
   - Combines the cost to reach a node ($g(n)$) and the heuristic estimate to the goal ($h(n)$).
   - Total cost $f(n) = g(n) + h(n)$.
   - Ensures optimal and complete solutions by balancing actual cost and heuristic.
2. **Greedy Best First Search:**
   - Only uses the heuristic value ($h(n)$).
   - Prioritizes nodes with the smallest $h(n)$, focusing on reaching the goal quickly.
   - May not guarantee the optimal path.

Steps to Find the Optimum Path:

1. A Search Algorithm*

- **Step 1:** Start at `A` with $f(A) = g(A) + h(A) = 0 + 3 = 3$. Add neighbors `B` and `C` to the open set.
  - $f(B) = g(B) + h(B) = 4 + 5 = 9$
  - $f(C) = g(C) + h(C) = 5 + 1 = 6$
- **Step 2:** Choose `C` (smallest f). Add its neighbors `D` and `E`:
  - $f(D) = g(D) + h(D) = 6 + 4 = 10$
  - $f(E) = g(E) + h(E) = 11 + 6 = 17$
- **Step 3:** Choose `D` (smallest f). Add its neighbor `G`:
  - $f(G) = g(G) + h(G) = 8 + 0 = 8$
- **Step 4:** Choose `G` (goal reached). Path: **A → C → D → G**. Total cost: **8**.

2. Greedy Best First Search

- **Step 1:** Start at `A`. Choose neighbor with smallest h:
  - `C` ($h(C) = 1$).
- **Step 2:** Move to `C`. Choose neighbor with smallest h:
  - `D` ($h(D) = 4$).
- **Step 3:** Move to `D`. Choose neighbor with smallest h:
  - `G` ($h(G) = 0$).
- **Step 4:** Goal reached. Path: **A → C → D → G**.

4. Explain the MIN-Max algorithm using a step-by-step approach

ANSWER

The **Min-Max algorithm** is used in two-player, zero-sum games to find the optimal move by maximizing the player's score and minimizing the opponent's score.

Steps:

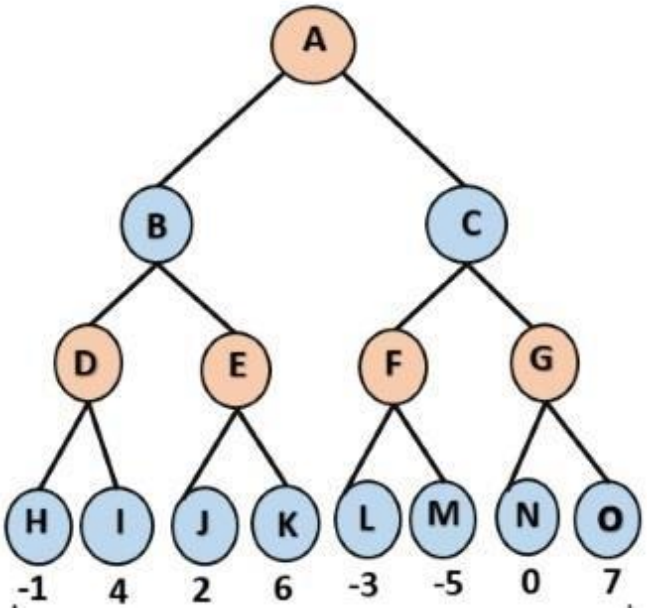
1. **Build the Game Tree:** Represent all possible moves from the current game state.
2. **Assign Terminal Values:** Leaf nodes are assigned values: +1 for a win, -1 for a loss, 0 for a draw.
3. **Evaluate Nodes:**
   - **Maximizing Player** (Player 1) chooses the highest value from child nodes.
   - **Minimizing Player** (Player 2) chooses the lowest value from child nodes.
4. **Backpropagate Values:** Propagate values up the tree, selecting the optimal moves at each level.
5. **Choose Optimal Move:** The root node value determines the best move for the current player.

Example:

In Tic-Tac-Toe:

- Player 1 (maximizer) aims to get the highest value (win = +1).
- Player 2 (minimizer) aims to get the lowest value (win for Player 1 = -1).

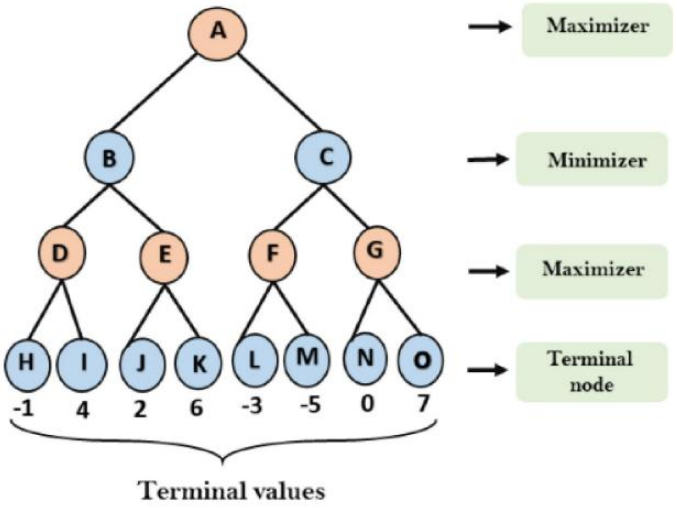
5. Prune the following Graph using the Alpha-Beta pruning method.



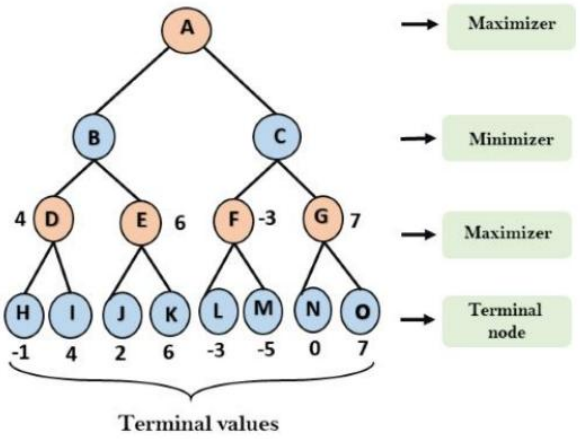
ANSWER

Step-1:

- In the first step, the algorithm **generates the entire game-tree** and apply the utility function to get the utility values for the terminal states.
- In the below tree diagram, let's take A is the initial state of the tree.
- Suppose :
  - maximizer takes first turn which has worst-case initial value =- infinity, and
  - minimizer will take next turn which has worst-case initial value = +infinity.



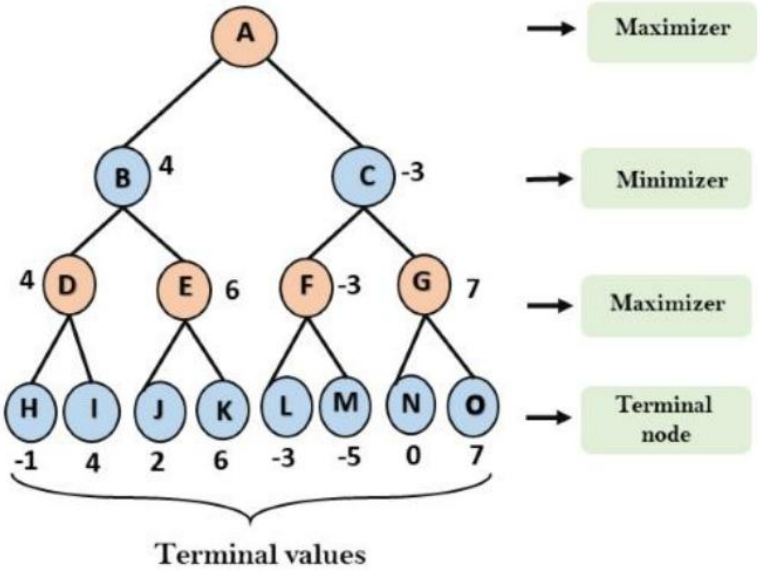
- **Step 2:**
- Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.
- For node D max(-1, -∞) => max(-1,4) = 4
- For Node E max(2, -∞) => max(2, 6) = 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) => max(0, 7) = 7



Step 3:

In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3

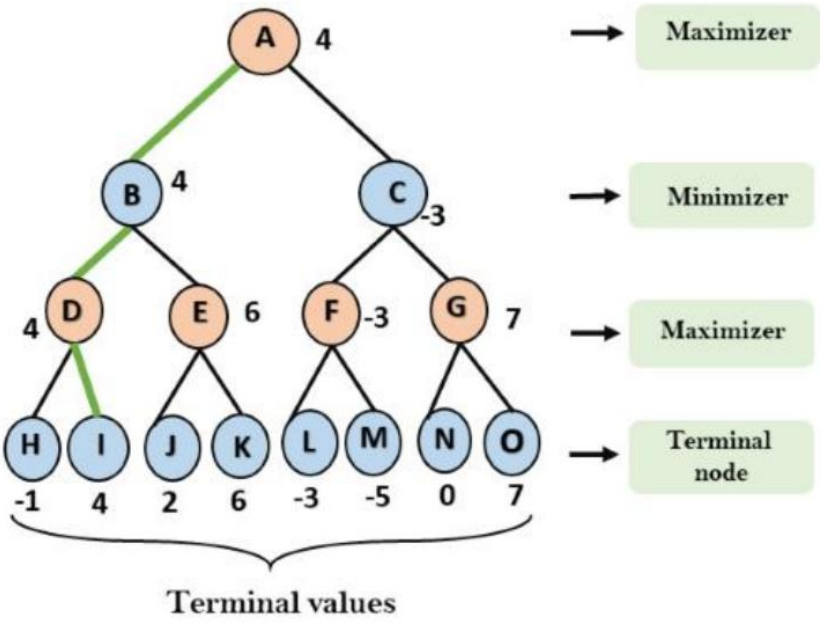


Step 4:

Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.

In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.