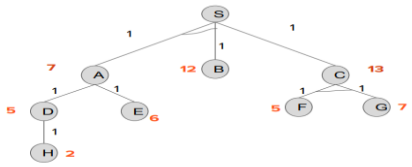
USN | | | | | | | | | | |

Internal Assessment Test 1 – June 2024

Sub:	**Artificial Intelligence**					Sub Code:	BAI402	Branch:	AIML	
Date:	**5/6/2024**	Duration:	90 min	Max Marks:	50	Sem/Sec:	IV /A, B & C		OBE	
	Answer any FIVE FULL Questions							MARKS	CO	RBT
1	What do you mean by AI, list the task domains of AI and discuss its types of agents?							10	CO1	L1
2	Illustrate the algorithm of AO* and find of the shortest path for the following using  AO* in the graph?							10	CO3	L3
3	Describe rationality and PEAS representation with an example.							10	CO3	L3
4a)	State how A* algorithm is differ from Best First Search (BFS).							5	CO2	L2
4b)	Differentiate between DFS and BFS (Breadth First Search) in detail.							5	CO2	L2
5	What are problem solving agents? Give any example and solve it with the help of backtracking or heuristic approach?							10	CO2	L1
6	Explain Iterative deepening depth first search with suitable example.							10	CO2	L2

CI **CCI** **HoD**

----------All the Best----------

CO-PO and CO-PSO Mapping

Course Outcomes		Blooms Level	Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Apply knowledge of agent architecture, searching and reasoning techniques for different applications.	**L1, L2**	**1**	2	2	1	-	-	-	-	-	-	-	-	-	1	-	1	-
CO2	Compare various Searching and Inferencing Techniques.	**L2, L3**	**2,3,4**	2	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-
CO3	Develop knowledge base sentences using propositional logic and first order logic	**L2**	**3, 4**	3	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-
CO4	Describe the concepts of quantifying uncertainty	**L2**	**5**	3	2	2	-	-	-	-	-	-	-	-	-	-	-	1	-
CO5	Use the concepts of Expert Systems to build applications.	**L2, L3**	**4,5**	3	2	3	-	1	-	-	-	-	-	-	-	1	-	1	-

SCHEME & SOLUTIONS

1) What do you mean by AI, list the task domains of AI and discuss its types of agents? (10) CO-1
5M AI definition and task
5M Types of Agents

What is AI?

Artificial Intelligence (AI) refers to the simulation of human intelligence processes by machines, particularly computer systems. These processes include learning (the acquisition of information and rules for using it), reasoning (using rules to reach approximate or definite conclusions), and self-correction. AI aims to create systems capable of performing tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation.

Task Domains of AI

AI can be applied in various task domains, including but not limited to:

1. **Natural Language Processing (NLP):**
   - Machine translation
   - Sentiment analysis
   - Speech recognition
   - Text summarization
2. **Computer Vision:**
   - Image recognition
   - Object detection
   - Facial recognition
   - Medical image analysis
3. **Robotics:**
   - Autonomous vehicles
   - Industrial robots
   - Drones
   - Service robots
4. **Expert Systems:**
   - Medical diagnosis
   - Financial analysis
   - Customer support systems
   - Fraud detection
5. **Machine Learning:**
   - Predictive analytics
   - Recommendation systems
   - Anomaly detection
   - Natural language generation
6. **Planning and Scheduling:**
   - Logistics and supply chain management
   - Workforce scheduling
   - Route optimization
   - Resource allocation
7. **Game Playing:**
   - Chess and Go
   - Real-time strategy games
   - Board games
   - Video games

8. **Autonomous Systems:**
    - Self-driving cars
    - Autonomous drones
    - Home automation systems
    - AI in IoT devices

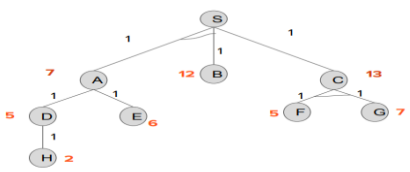
Types of AI Agents

AI agents can be classified based on their capabilities and functionalities:

1. **Reactive Agents:**
    - **Characteristics:** These agents do not have memory or a model of the world. They respond to specific inputs with predefined outputs.
    - **Examples:** Simple rule-based systems, such as an automated vacuum cleaner that changes direction when it hits an obstacle.
2. **Model-Based Reflex Agents:**
    - **Characteristics:** These agents maintain an internal state to track the part of the world they cannot see. They use this model to make decisions.
    - **Examples:** Thermostats that adjust heating based on temperature readings and pre-set schedules.
3. **Goal-Based Agents:**
    - **Characteristics:** These agents act to achieve specific goals. They consider future consequences of their actions and make decisions that bring them closer to their goals.
    - **Examples:** Pathfinding algorithms like A* (A-star) that find the shortest path from a start point to a goal.
4. **Utility-Based Agents:**
    - **Characteristics:** These agents aim to maximize a utility function that measures the agent's satisfaction. They can handle trade-offs and uncertainties to achieve the best outcome.
    - **Examples:** Recommendation systems that suggest products to maximize customer satisfaction and engagement.
5. **Learning Agents:**
    - **Characteristics:** These agents improve their performance over time by learning from their experiences. They have four main components: a learning element, a performance element, a critic, and a problem generator.
    - **Examples:** Machine learning models, such as reinforcement learning agents that learn optimal policies through rewards and punishments.
6. **Multi-Agent Systems:**
    - **Characteristics:** These systems involve multiple interacting agents, which can be cooperative, competitive, or both. They can solve problems that are beyond the capabilities of a single agent.
    - **Examples:** Swarm robotics, distributed sensor networks, and market-based trading systems

2) **Illustrate the algorithm of AO* and find of the shortest path for the following using AO* in the graph?** (10) CO-3

5M for the algo+ 5M for numerical solving



ANSWER:---

1. **Initialize**:
   - Start at the root node SSS.
   - Use a heuristic to estimate the cost from each node to the goal.
2. **Expand the Most Promising Node**:
   - Expand the node with the lowest estimated cost (cost to reach the node + heuristic cost).
3. **Update Costs**:
   - Update the cost of reaching each expanded node based on the newly discovered paths.
   - If the cost of a new path is less than the previously known cost, update the path.
4. **Repeat Until Goal is Reached**:
   - Continue expanding the most promising node until the goal node is reached and no cheaper paths are available.

Step-by-Step Illustration Using the Given Graph

Let's go through the AO* algorithm step-by-step for the provided graph. The numbers on the edges are the costs to move from one node to another, and the numbers in red next to the nodes are the heuristic costs.

1. **Start at Node SSS**:
   - Current node: SSS
   - Successors of SSS: A,B,CA, B, CA,B,C
   - Heuristic values: h(A)=7,h(B)=12,h(C)=13h(A) = 7, h(B) = 12, h(C) = 13h(A)=7,h(B)=12,h(C)=13
   - Total costs: f(A)=1+7=8,f(B)=1+12=13,f(C)=1+13=14f(A) = 1 + 7 = 8, f(B) = 1 + 12 = 13, f(C) = 1 + 13 = 14f(A)=1+7=8,f(B)=1+12=13,f(C)=1+13=14
   - Select node AAA (since it has the lowest cost of 8).
2. **Expand Node AAA**:
   - Current node: AAA
   - Successors of AAA: D,ED, ED,E
   - Heuristic values: h(D)=5,h(E)=6h(D) = 5, h(E) = 6h(D)=5,h(E)=6
   - Total costs: f(D)=2+5=7,f(E)=2+6=8f(D) = 2 + 5 = 7, f(E) = 2 + 6 = 8f(D)=2+5=7,f(E)=2+6=8
   - Select node DDD (since it has the lowest cost of 7).
3. **Expand Node DDD**:
   - Current node: DDD
   - Successors of DDD: HHH
   - Heuristic value: h(H)=2h(H) = 2h(H)=2
   - Total cost: f(H)=3+2=5f(H) = 3 + 2 = 5f(H)=3+2=5
   - Select node HHH (since it has the lowest cost of 5).
4. **Expand Node HHH**:
   - Current node: HHH
   - Node HHH is a leaf node with no successors.

At this point, we have reached the leaf node HHH with the total cost path from SSS as S→A→D→HS \rightarrow A \rightarrow D \rightarrow HS→A→D→H. We will now backtrack and update the costs to find the optimal path.

5. **Backtracking and Updating**:
   - Path cost from SSS to HHH: 1(S→A)+1(A→D)+1(D→H)=31 (S \rightarrow A) + 1 (A \rightarrow D) + 1 (D \rightarrow H) = 31(S→A)+1(A→D)+1(D→H)=3
   - Node HHH has no additional costs.
6. **Checking Other Paths**:
   - Check paths through BBB and CCC:
     - S→BS \rightarrow BS→B: Cost = 1 + 12 = 13
     - S→C→FS \rightarrow C \rightarrow FS→C→F: Cost = 1 + 1 + 5 = 7 (not cheaper)
     - S→C→GS \rightarrow C \rightarrow GS→C→G: Cost = 1 + 1 + 7 = 9 (not cheaper)

After comparing all the paths, the shortest path found using the AO* algorithm is:

S→A→D→HS \rightarrow A \rightarrow D \rightarrow HS→A→D→H with a total cost of **3**

3.Describe rationality and PEAS representation with an example. (10) CO-3
5 M rationality + 5M PEAS representation

Rationality in AI

Rationality in AI refers to the ability of an agent to make decisions that maximize its performance measure, given the knowledge it has, the perceptual information it receives, and its actions. A rational agent is one that acts to achieve the best expected outcome. Rationality does not mean perfection; it is about making the best possible decision based on the available information and the agent's goals.

PEAS Representation

PEAS stands for Performance measure, Environment, Actuators, and Sensors. It is a framework used to define the task environment of an intelligent agent. Here's what each component represents:

1. **Performance Measure**:
   - This is the criterion used to evaluate the agent's behavior. It defines what constitutes success for the agent.
2. **Environment**:
   - This is the context within which the agent operates. It includes everything that the agent can interact with or be affected by.
3. **Actuators**:
   - These are the mechanisms through which the agent can affect the environment. They are the means by which the agent takes actions.
4. **Sensors**:
   - These are the mechanisms through which the agent perceives the environment. They provide the agent with information about the state of the environment.

Example: Autonomous Vacuum Cleaner

Let's use an autonomous vacuum cleaner as an example to illustrate rationality and the PEAS representation.

Rationality

For an autonomous vacuum cleaner, being rational means that it should clean the floor efficiently, covering all dirty areas while avoiding obstacles and optimizing its path to save energy.

PEAS Representation

1. **Performance Measure**:
   - Cleanliness of the floor (percentage of dirt removed).
   - Efficiency (time taken to clean, energy consumption).
   - Coverage (percentage of area covered).
   - Avoidance of obstacles (minimizing collisions).
2. **Environment**:
   - The layout of the house (rooms, furniture, obstacles).
   - Types of floors (carpet, hardwood, tile).
   - Presence of dirt and debris.

3. **Actuators**:
    - Wheels or tracks for movement.
    - Suction mechanism for cleaning.
    - Brushes for sweeping.
    - Possibly, a beeper or light for signaling (e.g., when stuck).
4. **Sensors**:
    - Bump sensors to detect obstacles.
    - Dirt sensors to detect dirty areas.
    - Cliff sensors to detect stairs or drop-offs.
    - Optical sensors to map the environment.
    - Gyroscope and accelerometer for navigation.

Illustration

1. **Performance Measure**:
    - The vacuum cleaner should aim to maximize the cleanliness of the floor, minimize cleaning time and energy consumption, and avoid bumping into furniture or falling down stairs.
2. **Environment**:
    - The vacuum operates in a typical household environment, which can vary in layout, have different types of flooring, and contain various obstacles like furniture and toys.
3. **Actuators**:
    - The vacuum cleaner uses its wheels to move around, its suction and brushes to clean the floor, and possibly a beeper to signal if it needs assistance.
4. **Sensors**:
    - It uses bump sensors to detect and avoid obstacles, dirt sensors to find dirty spots, cliff sensors to avoid falling, and optical sensors to navigate and create a map of the house.

Rational Agent Behavior

Given this PEAS description, a rational vacuum cleaner would behave as follows:

- **Perceive** its environment using sensors to detect dirt, obstacles, and boundaries.
- **Plan** its path to cover the entire floor area efficiently, avoiding obstacles and returning to its charging station when needed.
- **Act** by moving to dirty areas and cleaning them, avoiding obstacles, and ensuring it doesn't fall down stairs or get stuck.

4a) State how A* algorithm is differ from Best First Search (BFS). (5) CO-2

2.5 for A* & 2.5 for BFS

The A* algorithm and Best-First Search (BFS) are both graph search algorithms used to find the shortest path from a start node to a goal node. However, they differ in how they prioritize which node to explore next. Here's a detailed comparison of the two:

Best-First Search (BFS)

1. **Heuristic Function (h(n)h(n)h(n))**:
    - BFS uses a heuristic function to estimate the cost from the current node nnn to the goal node.
    - The priority of each node is based solely on this heuristic estimate.
2. **Selection Criteria**:
    - Nodes are selected for expansion based solely on the heuristic function h(n)h(n)h(n).
    - It prioritizes nodes that appear to be closest to the goal based on the heuristic.

3. **Cost Consideration**:
    - BFS does not consider the actual cost to reach a node from the start node.
    - This can lead to paths that seem promising based on the heuristic but are actually longer in terms of actual cost.
4. **Completeness and Optimality**:
    - BFS is not guaranteed to find the shortest path if the heuristic function is not admissible (i.e., it overestimates the true cost).
    - It is not complete unless the heuristic is perfect.

A* Algorithm

1. **Evaluation Function (f(n)f(n)f(n))**:
    - A* uses an evaluation function f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n) where:
        - g(n)g(n)g(n) is the actual cost to reach node nnn from the start node.
        - h(n)h(n)h(n) is the heuristic estimate of the cost to reach the goal from node nnn.
    - This combines both the actual cost to reach the node and the estimated cost to reach the goal.
2. **Selection Criteria**:
    - Nodes are selected for expansion based on the combined cost f(n)f(n)f(n).
    - This ensures that both the distance traveled so far and the estimated distance to the goal are considered.
3. **Cost Consideration**:
    - A* considers both the actual cost to reach a node and the estimated cost to reach the goal.
    - This helps in finding a more balanced path that is likely to be optimal.
4. **Completeness and Optimality**:
    - A* is guaranteed to find the shortest path if the heuristic function h(n)h(n)h(n) is admissible (i.e., it never overestimates the true cost) and consistent (i.e., it satisfies the triangle inequality).
    - It is complete and optimal under these conditions.

Key Differences

1. **Heuristic Usage**:
    - BFS uses only the heuristic h(n)h(n)h(n) to prioritize nodes.
    - A* uses both the actual cost g(n)g(n)g(n) and the heuristic h(n)h(n)h(n) in its evaluation function f(n)f(n)f(n).
2. **Path Optimality**:
    - BFS does not guarantee an optimal path unless the heuristic is perfect.
    - A* guarantees an optimal path if the heuristic is admissible and consistent.
3. **Efficiency**:
    - A* is generally more efficient than BFS in finding the shortest path because it uses a more informed search strategy that considers both the past cost and future cost estimate.
    - BFS might explore paths that appear promising but are not optimal due to reliance only on the heuristic.

Example

Imagine navigating a maze where the goal is to find the shortest path from the entrance to the exit:

- **Best-First Search (BFS)**:
    - You might prioritize paths that seem to lead directly to the exit based on your visual estimation (heuristic). This can lead you down paths that look promising but end up being dead ends or longer routes.
- *A Algorithm**:
    - You would consider both the distance you have already traveled and your visual estimation to the exit. This balanced approach helps you avoid paths that might look good initially but are actually longer, and thus you find the optimal path more efficiently.

4b) Differentiate between DFS and BFS (Breadth First Search) in detail. (5) CO-2
2.5 M for DFS+ 2.5 M for BFS

Depth-First Search (DFS) and Breadth-First Search (BFS) are fundamental algorithms used to traverse or search through graphs and trees. They differ significantly in their approach and the way they explore nodes. Here's a detailed comparison of the two:

Depth-First Search (DFS)

1. **Method of Exploration**:
   - DFS explores as far as possible along each branch before backtracking. It goes deep into the graph, following one branch to its end before moving on to the next branch.
2. **Data Structure**:
   - DFS typically uses a stack (LIFO - Last In, First Out) to keep track of the nodes to be explored. This can be implemented using either a recursive function call stack or an explicit stack data structure.
3. **Traversal Order**:
   - DFS can be implemented in different ways (preorder, inorder, and postorder for trees), but generally, it visits a node, explores one of its children, and continues until it reaches a leaf, then backtracks to explore other children.
4. **Memory Usage**:
   - DFS is memory efficient for large graphs if implemented recursively, as it only needs to store a path from the root to a leaf node plus the nodes on the current path.
   - The space complexity is O(h)O(h)O(h), where hhh is the maximum depth of the search tree.
5. **Completeness and Optimality**:
   - DFS is not guaranteed to find the shortest path in graphs with cycles or multiple paths.
   - It is not complete for infinite-depth search spaces unless modifications (like depth-limited DFS) are made.
   - DFS is complete for finite search spaces and can be used for finding solutions in deep problem spaces if the solution is not necessarily the shortest path.
6. **Time Complexity**:
   - The time complexity of DFS is O(V+E)O(V + E)O(V+E), where VVV is the number of vertices and EEE is the number of edges in the graph.

Breadth-First Search (BFS)

1. **Method of Exploration**:
   - BFS explores all nodes at the present depth level before moving on to nodes at the next depth level. It expands the frontier evenly across the breadth of the graph.
2. **Data Structure**:
   - BFS uses a queue (FIFO - First In, First Out) to keep track of the nodes to be explored.
3. **Traversal Order**:
   - BFS starts from the root node and explores all its neighboring nodes before moving to the next level of neighbors.
4. **Memory Usage**:
   - BFS can consume a lot of memory if the branching factor is high because it needs to store all nodes at the current depth level in the queue.
   - The space complexity is O(bd)O(b^d)O(bd), where bbb is the branching factor and ddd is the depth of the shallowest solution.
5. **Completeness and Optimality**:

- BFS is guaranteed to find the shortest path in unweighted graphs because it explores all nodes at the current depth level before going deeper.
- It is complete and will always find a solution if one exists, given that the search space is finite.

6. **Time Complexity**:
   - The time complexity of BFS is O(V+E)O(V + E)O(V+E), similar to DFS.

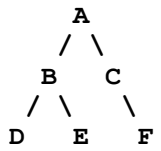
Key Differences

1. **Order of Node Expansion**:
   - **DFS**: Deep before wide. Expands a node's deepest descendant before exploring siblings.
   - **BFS**: Wide before deep. Explores all nodes at the present depth level before going deeper.
2. **Data Structure Used**:
   - **DFS**: Stack (can be implicit with recursion).
   - **BFS**: Queue.
3. **Memory Requirements**:
   - **DFS**: Low memory usage if the graph is deep but narrow.
   - **BFS**: Can have high memory usage for graphs with a high branching factor.
4. **Path Finding**:
   - **DFS**: Not guaranteed to find the shortest path.
   - **BFS**: Guaranteed to find the shortest path in unweighted graphs.
5. **Completeness**:
   - **DFS**: Not complete for infinite or large search spaces without modifications.
   - **BFS**: Complete for finite search spaces and unweighted graphs.

6. **Use Cases**:
   - **DFS**: Suitable for problems where the solution is deep in the tree or graph, such as puzzle solving with a deep solution path.
   - **BFS**: Suitable for finding the shortest path or exploring nodes at the same depth level, such as social networking sites finding the shortest connection path between two individuals.

Example

Consider a simple graph:



Starting at node AAA:

- **DFS** (Preorder traversal example):
  - Start at AAA, go to BBB, then to DDD. DDD has no children, backtrack to BBB, then go to EEE. Backtrack to AAA, then go to CCC, and finally to FFF.

  The order of traversal: A, B, D, E, C, F.

- **BFS**:
  - Start at AAA, explore BBB and CCC. Then, from BBB, explore DDD and EEE. Finally, from CCC, explore FFF.

  The order of traversal: A, B, C, D, E, F.

5.What are problem solving agents? Give any example and solve it with the help of backtracking or heuristic approach? (10) CO-2

5 M for problem solving+ 5 M for example with solutions

Problem Solving Agents

Problem-solving agents are a type of intelligent agent designed to solve specific problems by searching through a state space to find a solution. They operate by systematically exploring possible states of the problem until they find a goal state that satisfies the problem's requirements. These agents typically use search algorithms to explore the state space and determine the optimal or satisfactory solution.

Components of Problem-Solving Agents

1. **Initial State**:
   - The starting point of the problem.
2. **Actions**:
   - The set of actions available to the agent that can change the current state.
3. **Transition Model**:
   - A description of what each action does; defines the resulting state from taking an action in a given state.
4. **Goal Test**:
   - A test to determine whether a given state is a goal state.
5. **Path Cost**:
   - A function that assigns a cost to each path, used to compare different solutions.

Example: 8-Puzzle Problem

The 8-puzzle problem consists of a 3x3 grid with 8 numbered tiles and one empty space. The goal is to rearrange the tiles from a given initial configuration to a specified goal configuration by sliding the tiles into the empty space.

Initial State

Let's assume the initial state is:

```
1 2 3
4 5 6
7 8 _
```

And the goal state is:`1 2 3`

```
4 5 6
7 8 _
```

Solving with Heuristic Approach (A* Algorithm)

The A* algorithm is a heuristic-based search algorithm that combines the cost to reach the current state and the estimated cost to reach the goal state. We use the Manhattan distance as the heuristic function h(n)h(n)h(n), which is the sum of the absolute differences of the tiles' positions from their goal positions.

1. **Initial State**:
   - The puzzle's initial configuration.
2. **Actions**:
   - Move the empty space (denoted as _) up, down, left, or right.
3. **Transition Model**:
   - Defines how the puzzle configuration changes when an action is taken.
4. **Goal Test**:
   - Check if the current configuration matches the goal configuration.
5. **Path Cost**:
   - The cost of each move is 1.

Let's solve a simpler configuration for demonstration:

Initial State:

```
1 2 3
4 5 6
7 _ 8
```

Goal State:

```
1 2 3
4 5 6
7 8 _
```

Steps using A* Algorithm:

1. **Initial Node**:
   - State: `1 2 3 / 4 5 6 / 7 _ 8`
   - g(n)=0g(n) = 0g(n)=0 (cost from the start to the current node)
   - h(n)=1h(n) = 1h(n)=1 (Manhattan distance: tile 8 is 1 move away)
   - f(n)=g(n)+h(n)=0+1=1f(n) = g(n) + h(n) = 0 + 1 = 1f(n)=g(n)+h(n)=0+1=1
2. **Generate Successors**:
   - Move _ left:
     - State: `1 2 3 / 4 5 6 / 7 8 _`
     - g(n)=1g(n) = 1g(n)=1 (one move made)
     - h(n)=0h(n) = 0h(n)=0 (goal state reached)
     - f(n)=1+0=1f(n) = 1 + 0 = 1f(n)=1+0=1
   - Move _ up:
     - State: `1 2 3 / 4 _ 6 / 7 5 8`
     - g(n)=1g(n) = 1g(n)=1
     - h(n)=2h(n) = 2h(n)=2 (tiles 5 and 8 are 1 move each away from their goal)
     - f(n)=1+2=3f(n) = 1 + 2 = 3f(n)=1+2=3
3. **Select Node with Lowest f(n)f(n)f(n)**:
   - Choose state `1 2 3 / 4 5 6 / 7 8 _` (since f(n)=1f(n) = 1f(n)=1 is the lowest).
4. **Goal Test**:
   - Check if the selected node is the goal state. It is, so the algorithm terminates.

Solution

The steps to solve the puzzle are:

1. Move _ left, resulting in the goal state.

The A* algorithm efficiently finds the solution by using the heuristic to prioritize nodes that are closer to the goal, minimizing the number of moves needed to reach the goal state.

Solving with Backtracking

Backtracking is a general algorithm for finding all (or some) solutions to computational problems, notably constraint satisfaction problems. It incrementally builds candidates to the solutions and abandons each partial candidate as soon as it determines that this candidate cannot possibly be completed to a valid solution.

However, backtracking is less efficient for problems like the 8-puzzle due to the vast number of possible configurations, making heuristic-based approaches like A* more suitable.

6.Explain Iterative deepening depth first search with suitable example. (10) CO-2
5 M for definition and features + 5 M for example

Iterative Deepening Depth-First Search (IDDFS)

Iterative Deepening Depth-First Search (IDDFS) is an algorithm that combines the space efficiency of Depth-First Search (DFS) with the completeness of Breadth-First Search (BFS). IDDFS performs a series of depth-limited searches, gradually increasing the depth limit until the goal is found. Each depth-limited search is performed using DFS.

Key Features

1. **Space Efficiency**:
   - IDDFS uses the space efficiency of DFS, requiring only O(d)O(d)O(d) memory where ddd is the depth of the search tree.
2. **Completeness**:
   - Like BFS, IDDFS is complete and will find a solution if one exists, assuming the search space is finite.
3. **Optimality**:
   - IDDFS is optimal if all operators have the same cost, as it finds the shallowest goal node.

How IDDFS Works

1. **Set Initial Depth Limit**:
   - Start with a depth limit of 0.
2. **Perform Depth-Limited Search**:
   - Perform a DFS up to the current depth limit. If the goal is not found, increase the depth limit by 1 and repeat the process.
3. **Repeat**:
   - Continue increasing the depth limit and performing depth-limited searches until the goal is found or all nodes are explored.

Example: Solving the 8-Puzzle

Consider the 8-puzzle problem where we need to move tiles to reach a goal state from an initial state. Here's an example configuration:

Initial State:

```
1 2 3
4 5 6
```

```
7 8 _
```

Goal State:

```
1 2 3
4 5 6
7 8 _
```

For demonstration, let's use a simplified version of the problem with a smaller depth.

Steps of IDDFS

Depth Limit = 0

- Start at the root node (initial state).
- No children to explore, so increase depth limit.

Depth Limit = 1

- Explore nodes at depth 1:
  - Move _ left, right, up, or down (depending on the possible moves).

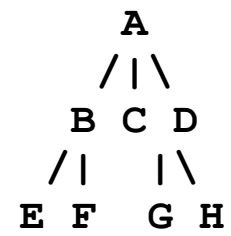
Depth Limit = 2

- Explore nodes at depth 2:
  - For each node at depth 1, generate successors and explore them.

Detailed Example with IDDFS on a Simple Graph

Consider a simple graph for a clearer demonstration:

mathematica



Let's find node "H" starting from "A".

Depth Limit = 0

- Only node "A".

Css

```
A
```

Depth Limit = 1

- Nodes: "A", "B", "C", "D".

css

```
A
|
```

```
B C D
```

Depth Limit = 2

- Nodes: "A", "B", "C", "D", "E", "F", "G", "H".

```
mathematica
A
|
B C D
|
E F G H
```

At depth limit 2, we find the goal node "H".