**UNIT-2**

## Problem-solving agents

In artificial intelligence, a problem-solving agent refers to a type of intelligent agent designed to address and solve complex problems or tasks in its environment. These agents are a fundamental concept in AI and are used in various applications, from game-playing algorithms to robotics and decision-making systems. Here are some key characteristics and components of a problem-solving agent:

1. **Perception**: Problem-solving agents typically have the ability to perceive or sense their environment. They can gather information about the current state of the world, often through sensors, cameras, or other data sources.
2. **Knowledge Base**: These agents often possess some form of knowledge or representation of the problem domain. This knowledge can be encoded in various ways, such as rules, facts, or models, depending on the specific problem.
3. **Reasoning**: Problem-solving agents employ reasoning mechanisms to make decisions and select actions based on their perception and knowledge. This involves processing information, making inferences, and selecting the best course of action.
4. **Planning**: For many complex problems, problem-solving agents engage in planning. They consider different sequences of actions to achieve their goals and decide on the most suitable action plan.
5. **Actuation**: After determining the best course of action, problem-solving agents take actions to interact with their environment. This can involve physical actions in the case of robotics or making decisions in more abstract problem-solving domains.
6. **Feedback**: Problem-solving agents often receive feedback from their environment, which they use to adjust their actions and refine their problem-solving strategies. This feedback loop helps them adapt to changing conditions and improve their performance.
7. **Learning**: Some problem-solving agents incorporate machine learning techniques to improve their performance over time. They can learn from experience, adapt their strategies, and become more efficient at solving similar problems in the future.

## Well-defined problems and solutions

On the basis of the problem and their working domain, different types of problem-solving agent defined and use at an atomic level without any internal state visible with a problem-solving algorithm. The problem-solving agent performs precisely by defining problems and several solutions. So we can say that problem solving is a part of artificial intelligence that encompasses a number of techniques such as a tree, B-tree, heuristic algorithms to solve a problem.

We can also say that a problem-solving agent is a result-driven agent and always focuses on satisfying the goals.

**There are basically three types of problem in artificial intelligence:**
**1. Ignorable:** In which solution steps can be ignored.
**2. Recoverable:** In which solution steps can be undone.

**3. Irrecoverable:** Solution steps cannot be undo.
**Steps problem-solving in AI:** The problem of AI is directly associated with the nature of humans and their activities. So we need a number of finite steps to solve a problem which makes human easy works.
These are the following steps which require to solve a problem :

- **Problem definition:** Detailed specification of inputs and acceptable system solutions.
- **Problem analysis:** Analyse the problem thoroughly.
- **Knowledge Representation:** collect detailed information about the problem and define all possible techniques.
- **Problem-solving:** Selection of best techniques.

Components to formulate the associated problem:

- **Initial State:** This state requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.
- **Action:** This stage of problem formulation works with function with a specific class taken from the initial state and all possible actions done in this stage.
- **Transition:** This stage of problem formulation integrates the actual action done by the previous action stage and collects the final stage to forward it to their next stage.
- **Goal test:** This stage determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determines the cost to achieve the goal.
- **Path costing:** This component of problem-solving numerical assigned what will be the cost to achieve the goal. It requires all hardware software and human working cost.

## Searching for solutions

- **Search:** Searchingis a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
    a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
    b. **Start State:** It is a state from where agent begins **the search**.
    c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.

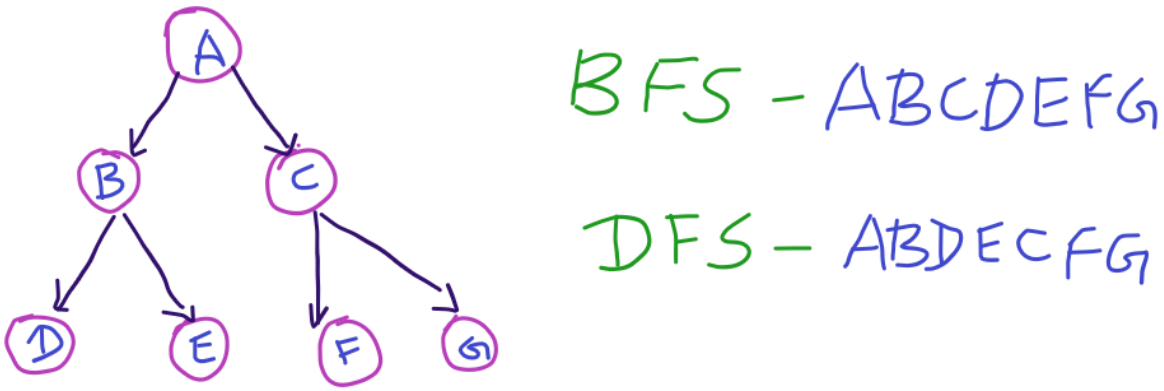
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

## Uniformed Searched strategy

Uninformed searches rely solely on the given problem definition and operate systematically to find a solution. Examples of uninformed search algorithms include breadth-first search (BFS), depth-first search (DFS),uniform-cost search (UCS), depth-limited search , and iterative deepening depth-first search. Although all these examples work in a brute force way, they differ in the way they traverse the nodes.

### Breadth-first search

The **breadth-first search** (BFS) algorithm is commonly used for traversing trees or graphs, where it explores the neighboring nodes in a breadth-first manner, visiting the nodes at the same level before moving to the next level. The breadth-first search approach uses a queue to facilitate its implementation.




### Depth-first search

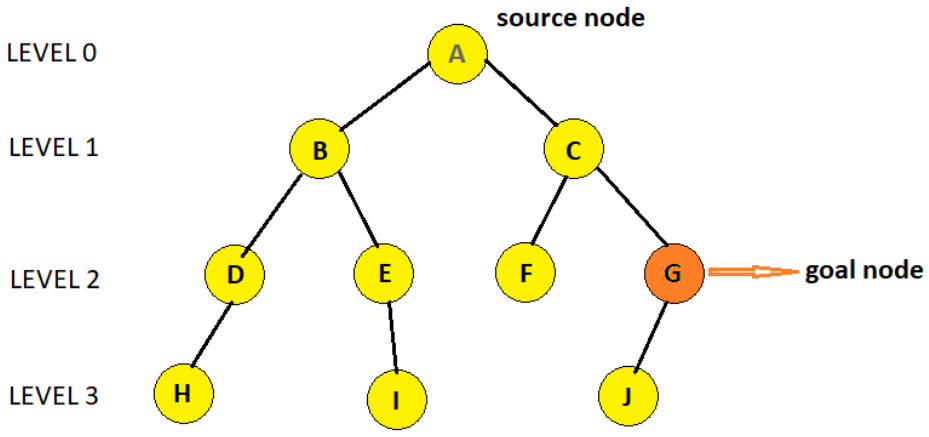
**Depth-first search** (DFS) is an uninformed search algorithm that helps traverse a tree or graph using the concept of backtracking. This algorithm traverses all nodes by advancing along a specific path and uses backtracking when it reaches the end of that path, allowing exploration of alternative paths. The DFS approach uses a stack to facilitate its implementation.

## Iterative deepening depth first search

Iterative deepening depth-first search (IDDFS) is an algorithm that is an important part of an Uninformed search strategy just like BFS and DFS. We can define IDDFS as an algorithm of an amalgam of BFS and DFS searching techniques. In IDDFS, We have found certain limitations in BFS and DFS so we have done hybridization of both the procedures for eliminating the demerits lying in them individually. We do a limited depth-first search up to a fixed "limited depth". Then we keep on incrementing the depth limit by iterating the procedure unless we have found the goal node or have traversed the whole tree whichever is earlier.

### Pseudo-code for IDDFS

```
1 IDDFS(T):

2   for d = 0 to infinity:

3   if (DLS(T, d)):

4   return 1

5   else

6   return 0
```



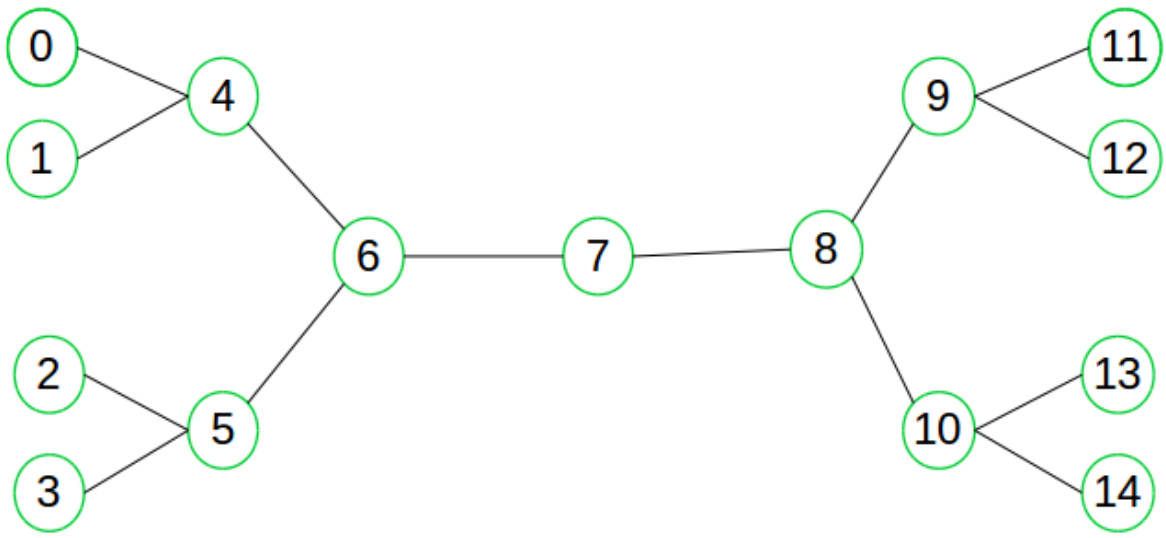

IDDFS with max depth-limit = 3
Note that iteration terminates at depth-limit=2
**Iteration 0:** A
**Iteration 1**: A->B->C
**Iteration 2:** A->B->D->E->C->F->G

## Bidirectional search

Searching a graph is quite famous problem and have a lot of practical use. We have already discussed here how to search for a goal vertex starting from a source vertex using BFS. In normal graph search using BFS/DFS we begin our search in one direction usually from source vertex toward the goal vertex, **but what if we start search from both direction simultaneously.**
Bidirectional search is a graph search algorithm which find smallest path from source to goal vertex. It runs two simultaneous search –

1. Forward search from source/initial vertex toward goal vertex
2. Backward search from goal/target vertex toward source vertex

Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs – one starting from initial vertex and other starting from goal vertex. **The search terminates when two graphs intersect.**
Just like A* algorithm, bidirectional search can be guided by a heuristic estimate of remaining distance from source to goal and vice versa for finding shortest path possible.
Consider following simple example-




## Greedy best first search

*Greedy Best-First Search is an AI search algorithm that attempts to find the most promising path from a given starting point to a goal. It prioritizes paths that appear to be the most promising, regardless of whether or not they are actually the shortest path. The algorithm works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.*

**How Greedy Best-First Search Works?**

- Greedy Best-First Search works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.
- The algorithm uses a heuristic function to determine which path is the most promising.
- The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths.
- If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

A* Algorithm Concepts and Implementation

It is a searching algorithm that is used to find the shortest path between an initial and a final point.

It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal.

It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time.

## AO* algorithm – Artificial intelligence

---

Best-first search is what the AO* algorithm does. The AO* method **divides** any given difficult **problem into a smaller group** of problems that are then resolved **using the AND-OR** graph concept. AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems. The AND side of the graph represents a set of tasks that must be completed to achieve the main goal, while the OR side of the graph represents different methods for accomplishing the same main goal.

*AND-OR Graph*

In the above figure, the **buying of a car** may be broken down into smaller problems or tasks that can be accomplished **to achieve the main goal** in the above figure, which is an example of a simple AND-OR graph. The other task is to either steal a car that will help us accomplish the main goal or use your own money to purchase a car that will accomplish the main goal. The AND symbol is used to indicate the AND part of the graphs, which refers to the need that all subproblems containing the AND to be resolved before the preceding node or issue may be finished.

The start state and the target state are already known in the knowledge-based search strategy known as the **AO* algorithm**, and the best path is identified by heuristics. The informed search technique considerably reduces the algorithm's **time complexity**. The AO* algorithm is far more effective in searching AND-OR trees **than** the A* algorithm.

**Working of AO* algorithm:**

The evaluation function in AO* looks like this:

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

**f(n) = Actual cost + Estimated cost**

here,

f(n) = The actual cost of traversal.

g(n) = the cost from the initial node to the current node.

h(n) = estimated cost from the current node to the goal state.