

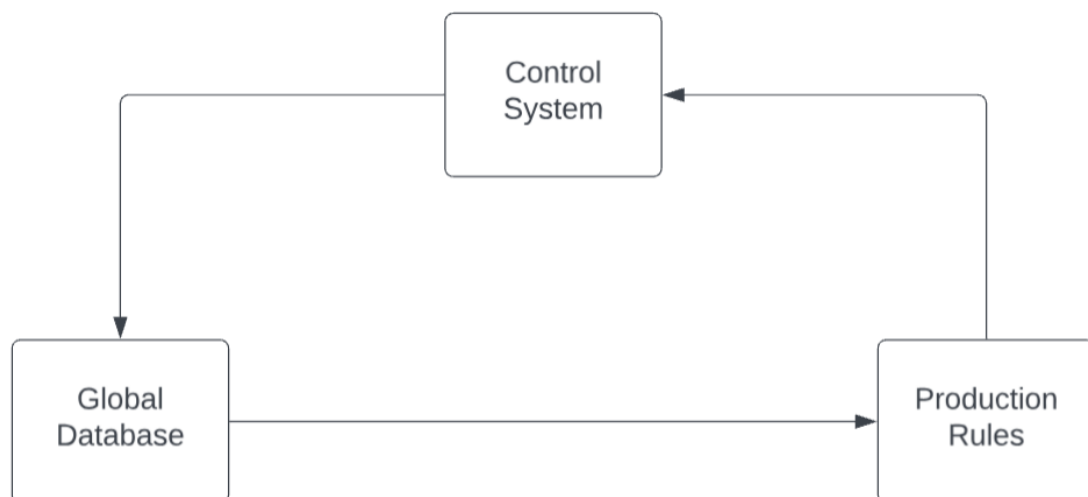
What is a Production System in AI?

A production system is based on a set of rules about behavior. These rules are a basic representation found helpful in expert systems, automated planning, and action selection.

Production system or production rule system is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior but it also includes the mechanism necessary to follow those rules as the system responds to states of the world.

The major components of the Production System in Artificial Intelligence are:

- **Global Database**: The global database is the central data structure used by the production system in Artificial Intelligence.
- **Set of Production Rules**: The production rules operate on the global database. Each rule usually has a precondition that is either satisfied or not by the global database. If the precondition is satisfied, the rule is usually be applied. The application of the rule changes the database.
- **A Control System**: The control system then chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If multiple rules are to fire at the same time, the control system resolves the conflicts.



Production System Rules

Production System rules can be classified as:

You can represent the knowledge in a production system as a set of rules along with a control system and database. It can be written as:

- **Deductive Inference Rules**
- **Abductive Inference Rules**

If(Condition) Else (Condition)

Deductive Inference Rules	Abductive Inference Rules
Deductive inference rules are logic used in AI and knowledge-based systems. They facilitate <u>deductive reasoning</u> , which involves drawing specific conclusions from general premises or facts. In deductive reasoning, the conclusion is guaranteed to be true if the premises are true and the inference rule is valid. Modus Ponens and Modus Tollens are common <u>deductive inference rules</u> that help derive valid conclusions from given facts and rules.	Abductive inference rules are used in AI and reasoning systems to make educated guesses or hypotheses based on observed data or evidence. Abductive reasoning involves generating plausible explanations or hypotheses to explain the available information. Unlike deductive reasoning, abductive conclusions are not guaranteed true but are selected based on their likelihood, given the available evidence. Abductive inference is particularly useful in situations with incomplete or uncertain data, where the system needs to make the best possible guess or explanation.

✓ **The main features of the production system include:**

- ✓ **1. Simplicity:** The structure of each sentence in a production system is unique and uniform as they use the “IF-THEN” structure. This structure provides simplicity in knowledge representation. This feature of the production system improves the readability of production rules.
- └ **2. Modularity:** This means the production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deleterious side effects.
- 3. Modifiability:** This means the facility for modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

4. **Knowledge-intensive:** The knowledge base of the production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Classes of Production System in Artificial Intelligence

There are four major classes of Production System in Artificial Intelligence:

Monotonic Production System

In a monotonic production system, the laws and truths remain constant during execution. Once a fact is deduced, it and the corresponding rule stay fixed throughout the process. Consequently, this stability ensures predictability but may limit adaptability in dynamic environments where changes are frequent.

Partially Commutative Production System:

By contrast, in this type of system, the rules can be applied flexibly, allowing for some degree of adaptability while still maintaining certain constraints. This partial commutativity strikes a balance between stability and flexibility, making it useful in scenarios that require both some level of consistency and the ability to adjust to changes.

Non-Monotonic Production Systems

Conversely, non-monotonic production systems are more dynamic and adaptive. During execution, the system can add, modify, or retract rules. Therefore, this flexibility makes them excellent for situations where the knowledge base needs to change in response to shifting circumstances, offering high adaptability and responsiveness.

Commutative Systems

Finally, commutative systems have rules that can be applied in any sequence without changing the result. In circumstances where the sequence of rule application is not essential, this high degree of flexibility may be beneficial. Thus, commutative systems are ideal when the order of operations does not affect the outcome, providing significant operational flexibility.

Real-World Examples of AI Production Systems in Use

- **Customer Support Chatbots:** AI-powered chatbots in customer support systems use production rules to handle customer inquiries, provide answers, and escalate complex issues to human agents.
- **Fraud Detection Systems:** In financial institutions, AI production systems detect fraudulent activities by analyzing transaction data and applying predefined fraud detection rules.
- **Medical Diagnosis:** AI production systems are used in healthcare for medical diagnosis. They analyze patient symptoms, medical history, and test results to suggest possible diagnoses and treatment options.
- **Traffic Management:** Smart traffic management systems use AI production systems to optimize traffic flow by adjusting signal timings based on real-time traffic conditions and predefined rules.

Production System in Artificial Intelligence: Example

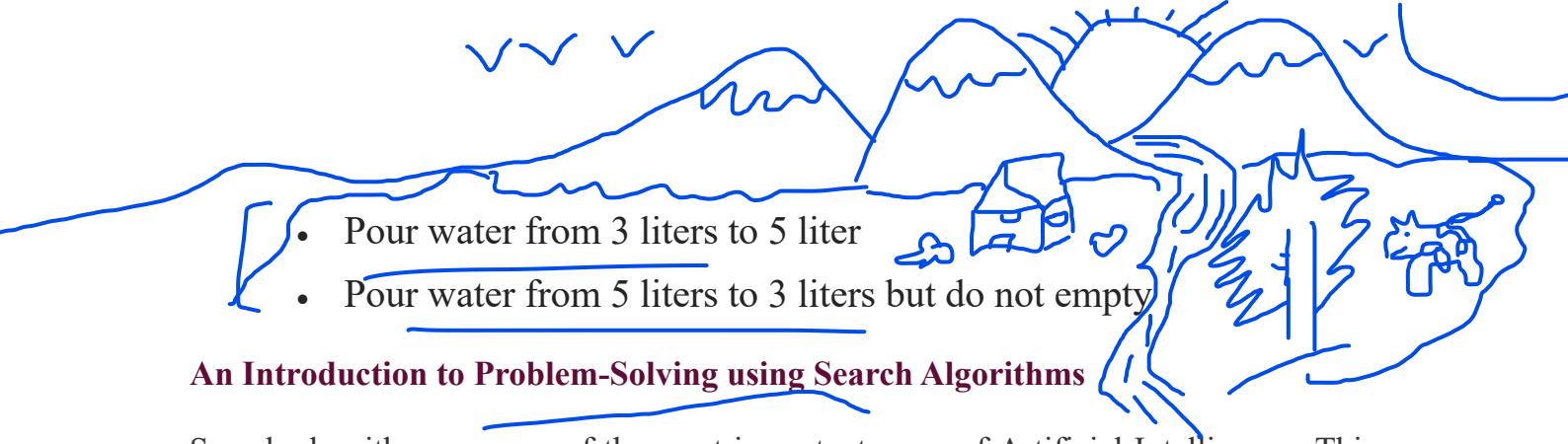
Problem Statement:

We have two jugs of capacity 5l and 3l (liter), and a tap with an endless supply of water. The objective is to obtain 4 liters exactly in the 5-liter jug with the minimum steps possible.

- Fill the 5-liter jug from the tap
- Empty the 5-liter jug
- Fill the 3-liter jug from the tap
- Empty the 3-liter jug
- Then, empty the 3-liter jug to 5 liter
- Empty the 5-liter jug to 3 liter

5L → 3L
0L →

2

- 
- Pour water from 3 liters to 5 liter
 - Pour water from 5 liters to 3 liters but do not empty

An Introduction to Problem-Solving using Search Algorithms

Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI.

Problem-solving agents:

In Artificial Intelligence, Search techniques are universal problem-solving methods. **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

A search problem consists of:

- **A State Space.** Set of all possible states where you can be.
- **A Start State.** The state from where the search begins.
- **A Goal State.** A function that looks at the current state returns whether or not it is the goal state.

The process of problem-solving using searching consists of the following steps.

- Define the problem
- Analyze the problem
- Identification of possible solutions
- Choosing the optimal solution
- Implementation

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:

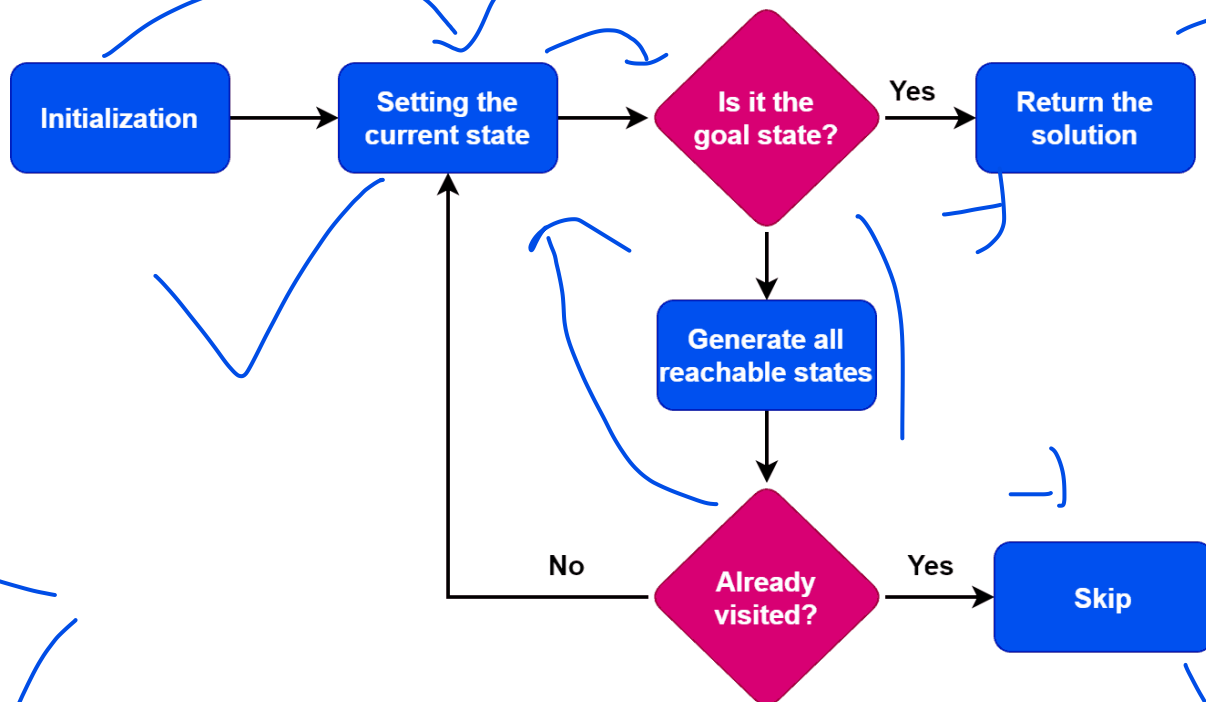
- **Initial State:** This state requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.
- **Action:** This stage of problem formulation works with function with a specific class 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.

State Space Search in Artificial Intelligence

- A state space is a mathematical representation of a problem that defines all possible states that the problem can be in. Furthermore, in search algorithms, we use a state space to represent the current state of the problem, the initial state, and the goal state.
- The state space size can greatly affect a search algorithm's efficiency. Hence, it's important to choose an appropriate representation and search strategy to efficiently search the state space.
- State space search is a strategy used in AI to find a path to a goal when there are many possibilities to consider. It's like having many doors to open, with each door leading to a room full of other doors. Some paths lead to dead ends, some go in circles, and some lead to the goal. The "state space" is the name for all the rooms and doors combined.
- AI uses this method to do things like planning the best route for delivery trucks, figuring out the moves in a game, or even making decisions about investments. It's a fundamental part of how AI systems think and make decisions.

Steps

Now let's discuss the steps of a typical state space search algorithm:



- To begin the search process, we set the current state to the initial state.
- We then check if the current state is the goal state. If it is, we terminate the algorithm and return the result.
- If the current state is not the goal state, we generate the set of possible successor states that can be reached from the current state.
- For each successor state, we check if it has already been visited. If it has, we skip it, else we add it to the queue of states to be visited.
- Next, we set the next state in the queue as the current state and check if it's the goal state. If it is, we return the result. If not, we repeat the previous step until we find the goal state or explore all the states.
- If all possible states have been explored and the goal state still needs to be found, we return with no solution.

Applications of State Space Search

State space search is a fundamental concept in artificial intelligence that has a wide range of applications. Here are some of the key areas where state space search is utilized:

1. Puzzle Solving:

State space search algorithms are often used to solve complex puzzles like the Rubik's Cube, sliding tile puzzles, and the water jug problem we discussed earlier. These puzzles can be represented as a state space, and algorithms can be applied to find solutions.

2. Pathfinding:

In video games and robotics, state space search is used for pathfinding – determining the shortest or most cost-effective path from one point to another. Algorithms like A* and Dijkstra's are used in maps and game levels to find paths considering various terrains and obstacles.

3. Planning and Scheduling:

AI planning involves creating a sequence of actions to achieve a specific goal. State space search helps in finding the best sequence of events in logistics, production schedules, or even in AI for playing strategy games.

4. Natural Language Processing:

State space search can be used in parsing algorithms for natural language processing, where the states represent possible interpretations of the sentences, and the search is for the most likely meaning.

5. Machine Learning:

In machine learning, especially in reinforcement learning, state space search helps in exploring different states and actions to maximize the reward function.

6. Problem Solving in AI:

State space search is a general problem-solving approach in AI. It is used in expert systems, theorem proving, and even in medical diagnosis systems where different states represent the presence or absence of symptoms and diseases.

7. Robotics:

Robots use state space search to navigate and interact with their environment. They need to understand the current state, explore possible next states, and choose the best action to perform tasks.

8. Automated Reasoning:

State space search is used in automated reasoning, where a system needs to infer new knowledge from the known facts. This is crucial in fields like law and computer-aided verification.

9. Optimization Problems:

Many optimization problems can be framed as state space searches, where the goal is to find the state that maximizes or minimizes a particular function.

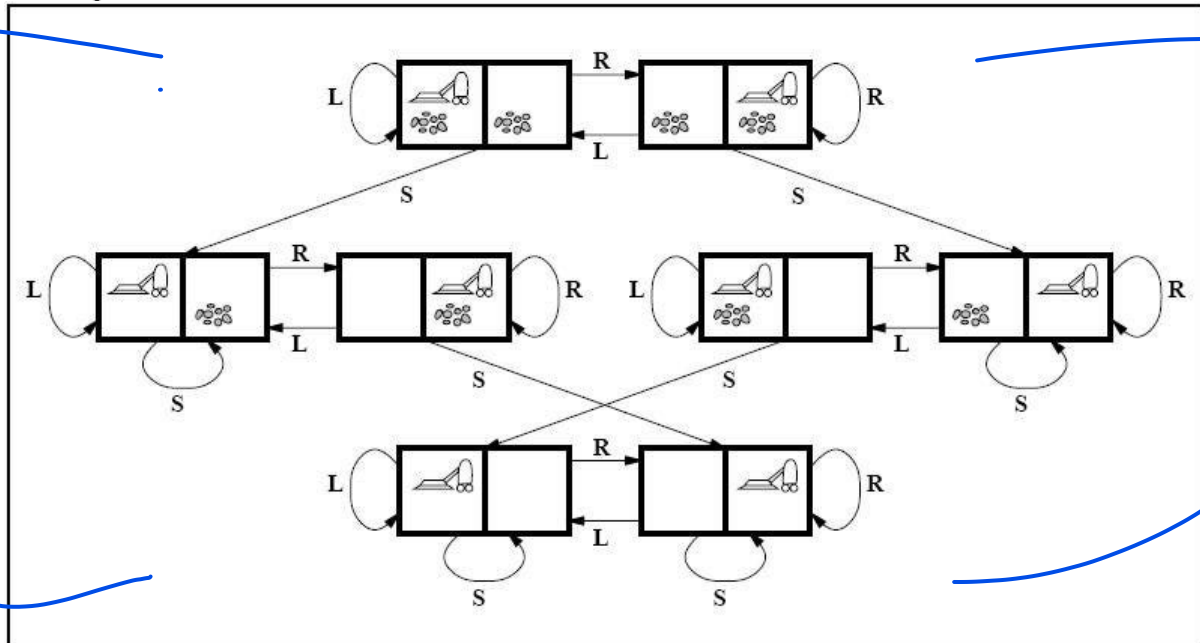
10. Quantum Computing:

In quantum computing, state space search can be used to explore the possibilities of quantum states to solve problems that are intractable on classical computers.

These applications demonstrate the versatility of state space search in solving a variety of problems by systematically exploring possible states and actions to find an optimal solution or to prove that no solution exists.

An Example Problem Formulation

Let us take the example of vacuum world that was introduced in the starting of this series, There is a vacuum cleaner agent and it can move left or right and its jump is to suck up the dirt from the floor.



State space for vacuum world.

The problem for vacuum world can be formulated as follows:

States: The state is determined by both the agent location and the dirt location. The agent is in one of two locations, each of which might or might not contain dirt.

Therefore, there are $2 \times 2^2 = 8$ possible world states.

A larger environment would have $n \times 2$ to the power of n states.

Initial State: Any state can be assigned as the initial state in this case.

Action: In this environment there are three actions, *Move Left*, *Move Right*, *Suck up the dirt*.

Transition Model: All the actions have expected effects, except for when the agent is in leftmost square and the action is *Left*, when the agent is in rightmost square and the action is *Right* and the square is clean when the action is to *Suck*.

Goal Test: Goal test checks whether all the squares are clean.

Path Cost: Each step costs 1, so the path cost is the number of steps in the path.

Properties of search algorithms

Completeness

A search algorithm is said to be complete when it gives a solution or returns any solution for a given random input.

Optimality

If a solution found is best (lowest path cost) among all the solutions identified, then that solution is said to be an optimal one.

Time complexity

The time taken by an algorithm to complete its task is called time complexity. If the algorithm completes a task in a lesser amount of time, then it is an efficient one.

Space complexity

It is the maximum storage or memory taken by the algorithm at any time while searching.

Types of search algorithms

- Uninformed search
- Informed search

Uninformed search algorithms

The uninformed search algorithm does not have any domain knowledge such as closeness, location of the goal state, etc. it behaves in a brute-force way.

It only knows the information about how to traverse the given tree and how to find the goal state. This algorithm is also known as the Blind search algorithm or Brute-Force algorithm.

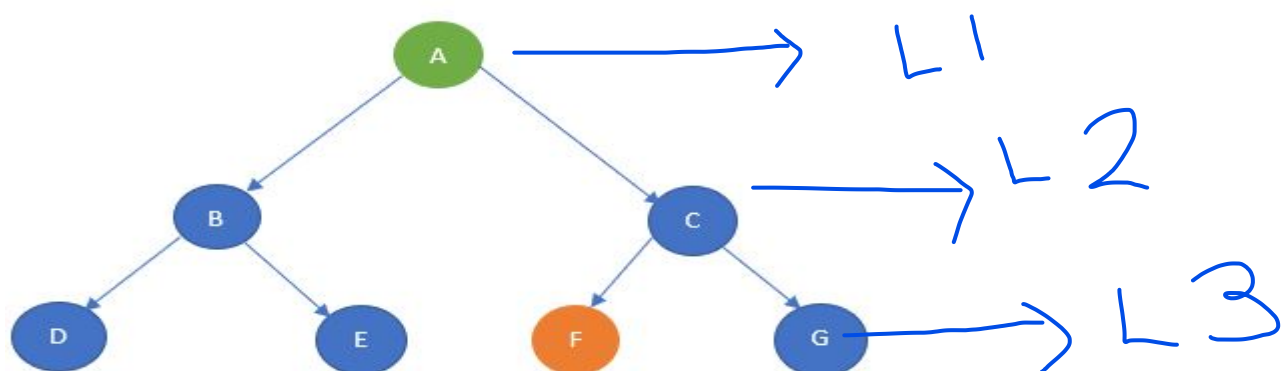
The uninformed search strategies are of six types.

- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening depth-first search
- Bidirectional search
- Uniform cost search

1. Breadth-first search

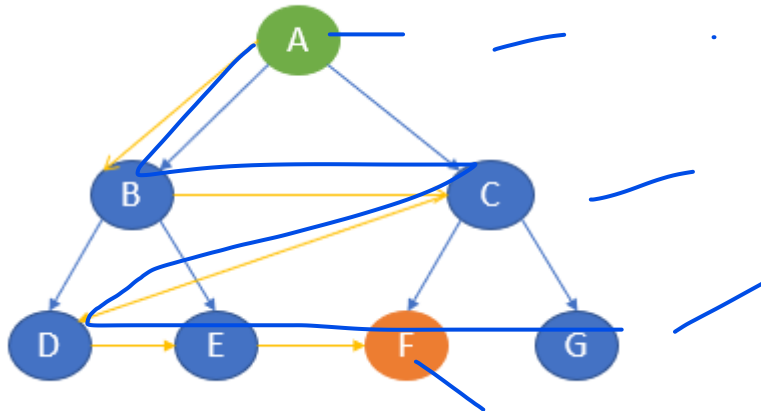
It is of the most common search strategies. It generally starts from the root node and examines the neighbor nodes and then moves to the next level. It uses First-in First-out (FIFO) strategy as it gives the shortest path to achieving the solution. BFS is used where the given problem is very small and space complexity is not considered.

Now, consider the following tree.



The BFS algorithm starts with the start state and then goes to the next level and visits the node until it reaches the goal state.

In this example, it starts from A and then travel to the next level and visits B and C and then travel to the next level and visits D, E, F and G. Here, the goal state is defined as F. So, the traversal will stop at F.



The path of traversal is: A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F

Advantages of BFS

- BFS will never be trapped in any unwanted nodes.
- If the graph has more than one solution, then BFS will return the optimal solution which provides the shortest path.

Disadvantages of BFS

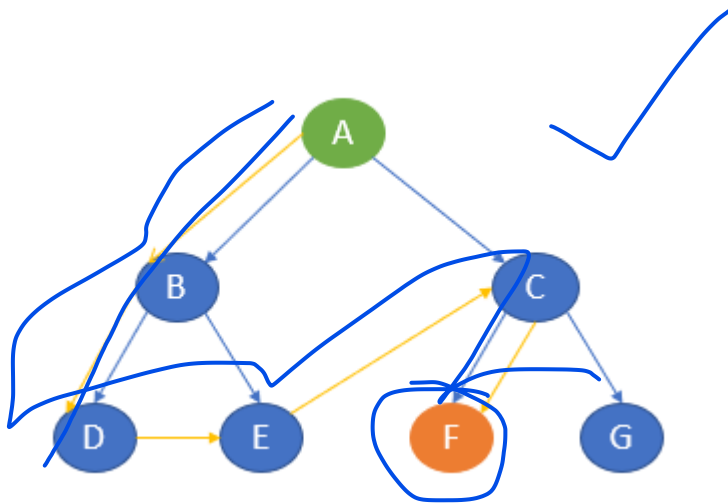
- BFS stores all the nodes in the current level and then go to the next level. It requires a lot of memory to store the nodes.
- BFS takes more time to reach the goal state which is far away.

2. Depth-first search

The depth-first search uses Last-in, First-out (LIFO) strategy and hence it can be implemented by using stack. DFS uses backtracking. That is, it starts from the initial state and explores each path to its greatest depth before it moves to the next path.

DFS will follow: Root node \rightarrow Left node \rightarrow Right node

Here, it starts from the start state A and then travels to B and then it goes to D. After reaching D, it backtracks to B. B is already visited, hence it goes to the next depth E and then backtracks to B. as it is already visited, it goes back to A. A is already visited. So, it goes to C and then to F. F is our goal state and it stops there.



The path of traversal is: A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F

Advantages of DFS

- It takes lesser memory as compared to BFS.
- The time complexity is lesser when compared to BFS.
- DFS does not require much more search.

Disadvantages of DFS

- DFS does not always guarantee to give a solution.
- As DFS goes deep down, it may get trapped in an infinite loop.

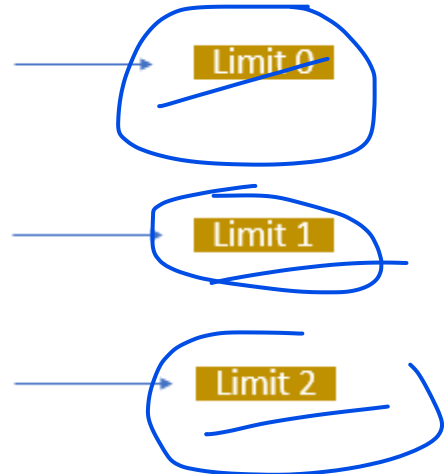
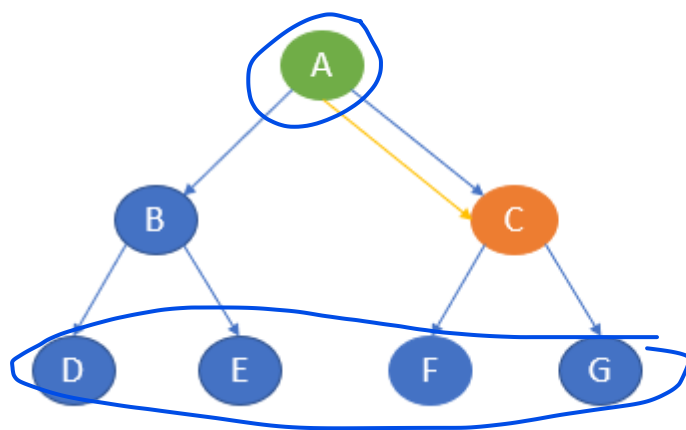
3. Depth-limited search

Depth-limited works similarly to depth-first search. The difference here is that depth-limited search has a pre-defined limit up to which it can traverse the nodes. Depth-limited search solves one of the drawbacks of DFS as it does not go to an infinite path.

Now, consider the same example.

Let's take A as the start node and C as the goal state and limit as 1.

The traversal first starts with node A and then goes to the next level 1 and the goal state C is there. It stops the traversal.

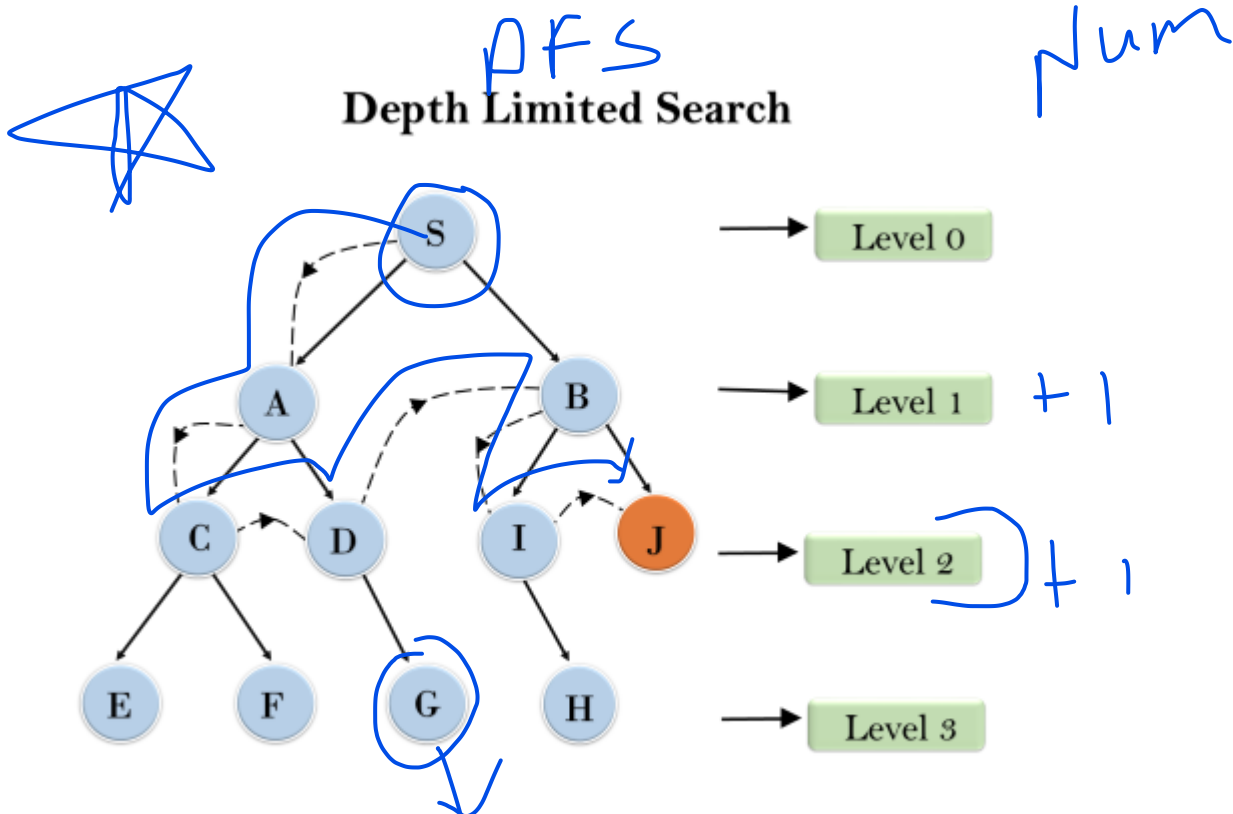


The path of traversal is:

A → C

If we give C as the goal node and the limit as 0, the algorithm will not return any path as the goal node is not available within the given limit.

If we give the goal node as F and limit as 2, the path will be A, C, F.



Advantages of DLS

- It takes lesser memory when compared to other search techniques.

Disadvantages of DLS

- DLS may not offer an optimal solution if the problem has more than one solution.
- DLS also encounters incompleteness.

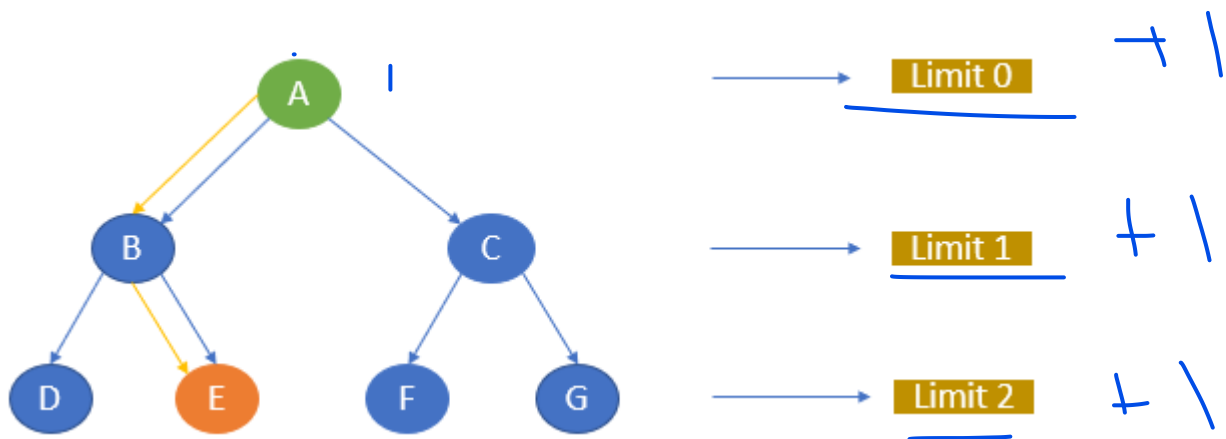
4. Iterative deepening depth-first search

Iterative deepening depth-first search is a combination of depth-first search and breadth-first search. IDDFS find the best depth limit by gradually adding the limit until the defined goal state is reached.

Let me try to explain this with the same example tree.

Consider, A as the start node and E as the goal node. Let the maximum depth be 2.

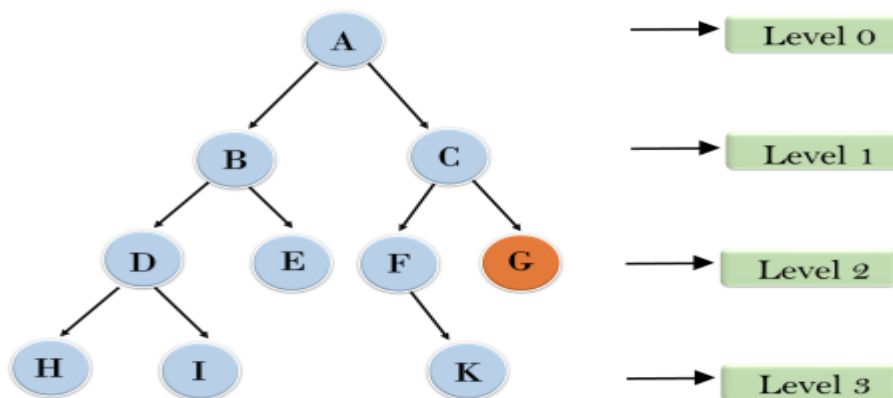
The algorithm starts with A and goes to the next level and searches for E. If not found, it goes to the next level and finds E.



The path of traversal is

A → B → E

Iterative deepening depth first search



1'st Iteration-----> A

2'nd Iteration-----> A, B, C

3'rd Iteration----->A, B, D, E, C, F, G

4'th Iteration----->A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Advantages of IDDFS

- IDDFS has the advantages of both BFS and DFS.
- It offers fast search and uses memory efficiently.

Disadvantages of IDDFS

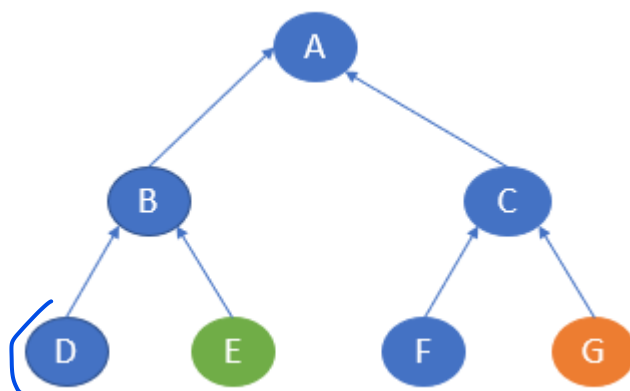
- It does all the works of the previous stage again and again.

5. Bidirectional search

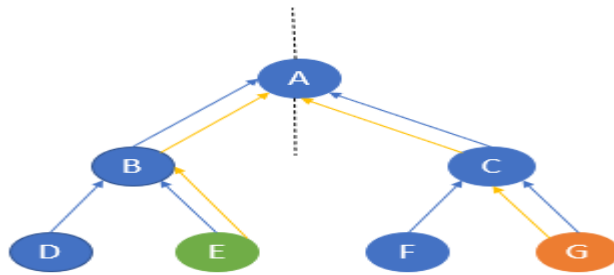
The bidirectional search algorithm is completely different from all other search strategies. It executes two simultaneous searches called forward-search and backwards-search and reaches the goal state. Here, the graph is divided into two smaller sub-graphs. In one graph, the search is started from the initial start state and in the other graph, the search is started from the goal state. When these two nodes intersect each other, the search will be terminated.

Bidirectional search requires both start and goal start to be well defined and the branching factor to be the same in the two directions.

Consider the below graph.



Here, the start state is E and the goal state is G. In one sub-graph, the search starts from E and in the other, the search starts from G. E will go to B and then A. G will go to C and then A. Here, both the traversal meets at A and hence the traversal ends.

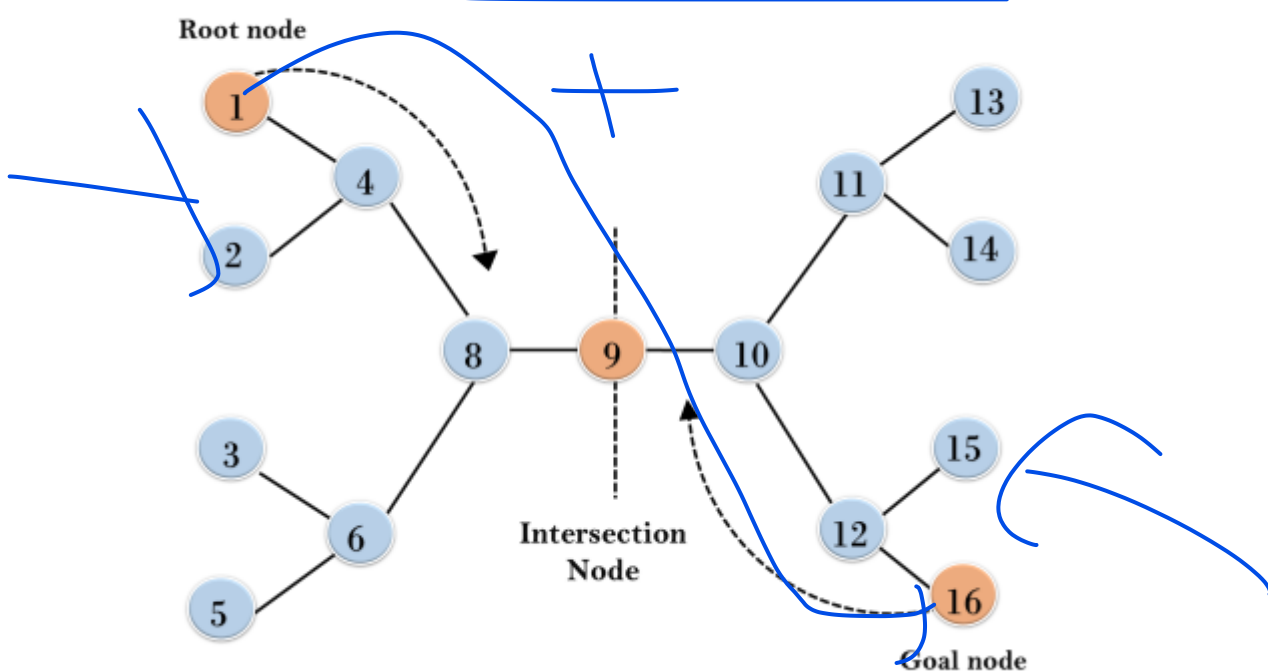


The path of traversal is
 $E \rightarrow B \rightarrow A \rightarrow C \rightarrow G$

Example:

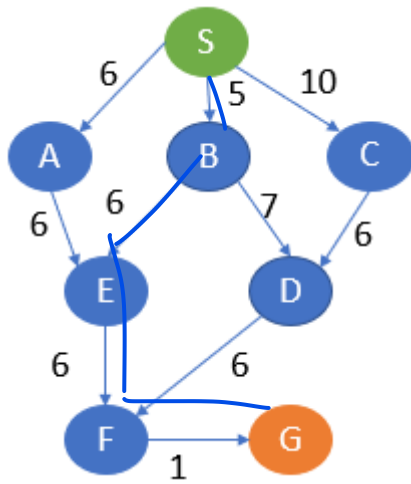
In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction. The algorithm terminates at node 9 where two searches meet.

Bidirectional Search



6. Uniform cost search

Uniform cost search is considered the best search algorithm for a weighted graph or graph with costs. It searches the graph by giving maximum priority to the lowest cumulative cost. Uniform cost search can be implemented using a priority queue. Consider the below graph where each node has a pre-defined cost.



Here, S is the start node and G is the goal node.

From S, G can be reached in the following ways.

S, A, E, F, G -> 19

S, B, E, F, G -> 18

S, B, D, F, G -> 19

S, C, D, F, G -> 23

Here, the path with the least cost is S, B, E, F, G.

