Artificial Intelligence

By: Mohit Goel
Assistant Professor
Email id- mohit.16907@lpu.co.in
Block-36, Room No.-203

Problem and Production System Characteristics

By: Mohit Goel (Mr. Feb)

Problem solving **is a process** of generating solutions from observed data.

- a 'problem' is characterized by a set of *goals*,
- a set of *objects*, and
- a set of *operations*.

These could be ill-defined and may evolve during problem solving.

To solve the problem of building a system you should take the following steps:

1. Define the problem accurately including detailed specifications and what constitutes a suitable solution.

2. Scrutinize the problem carefully, for some features may have a central affect on the chosen method of solution.

3. Segregate and represent the background knowledge needed in the solution of the problem.

4. Choose the best solving techniques for the problem to solve a solution.

PROBLEMS, PROBLEM SPACES AND SEARCH

• A '***problem space***' is an abstract space.

A problem space encompasses all *valid states* that can be generated by the application of any combination of *operators* on any combination of *objects*.

✓The problem space may contain one or more *solutions*. A solution is a combination of *operations* and *objects* that achieve the *goals*.

• A '***search***' refers to the search for a solution in a problem space.

Search proceeds with different types of '*search control strategies*'.

The *depth-first search and breadth-first search* are the two common *search strategies*.

Representing states

- At any moment, the relevant world is represented as a **state**
  - Initial (start) state: **S**
  - An action (or an **operation**) changes the current state to another state (if it is applied): state transition
  - An action can be taken (applicable) only if the its precondition is met by the current state
  - For a given state, there might be more than one applicable actions
  - **Goal state**: a state satisfies the goal description or passes the goal test
  - **Dead-end state**: a non-goal state to which no action is applicable

Representing states

- **State space**:
  - Includes the initial state S and all other states that are reachable from S by a sequence of actions
  - A state space can be organized as a graph:

    nodes: states in the space

    arcs: actions/operations
- The **size of a problem** is usually described in terms of the **number of states** (or the size of the state space) that are possible.
  - Tic-Tac-Toe has about 3^9 states.
  - Checkers has about 10^40 states.
  - Rubik's Cube has about 10^19 states.
  - Chess has about 10^120 states in a typical game.

What are the actions?

- Quantify all of the primitive actions or events that are sufficient to describe all necessary changes in solving a task/goal.
- No uncertainty associated with what an action does to the world. That is, given an action (**operator or move**) and a description of the current state of the world, the action completely specifies
  - **Precondition:** if that action CAN be applied to the current world (i.e., is it applicable and legal), and
  - **Effect:** what the exact state of the world will be after the action is performed in the current world (i.e., no need for "history" information to be able to compute what the new world looks like).

Actions

- That is, the world is in one situation, then an action occurs and the world is now in a new situation.
- For example, if "Mary is in class" and then performs the action "go home," then in the next situation she is "at home." There is no representation of a point in time where she is neither in class nor at home (i.e., in the state of "going home").

- The number of operators needed depends on the **representation** used in describing a state.

Formalizing Search in a State Space

•A state space is a **graph**, (V, E) where V is a set of **nodes** and E is a set of **arcs**, where each arc is directed from a node to another node

•**node:** corresponds to a **state**

–state description

•**arc:** corresponds to an applicable action/operation.

–the source and destination nodes are called as **parent (immediate predecessor)** and **child** (**immediate successor**) nodes with respect to each other

–each arc has a fixed, non-negative **cost** associated with it, corresponding to the cost of the action

Formalizing Search in a State Space

- **node generation:** making explicit a node by applying an action to another node which has been made explicit
- **node expansion:** generate **all** children of an explicit node by applying **all** applicable operations to that node
- One or more nodes are designated as **start nodes**
- A **goal test** predicate is applied to a node to determine if its associated state is a goal state
- A **solution** is a sequence of operations that is associated with a path in a state space from a start node to a goal node
- The **cost of a solution** is the sum of the arc costs on the solution path

For Example:

It one wants to make a cup of coffee then what he will do?

State space representation of this problem can be done as follow:

First of all analyze the problem i.e. verify whether the necessary ingredients like instant cases power milk power, sugar, kettle stove etc are available or not.

If are available, them steps to solve the problems are.

Boil necessary water in the kettle.

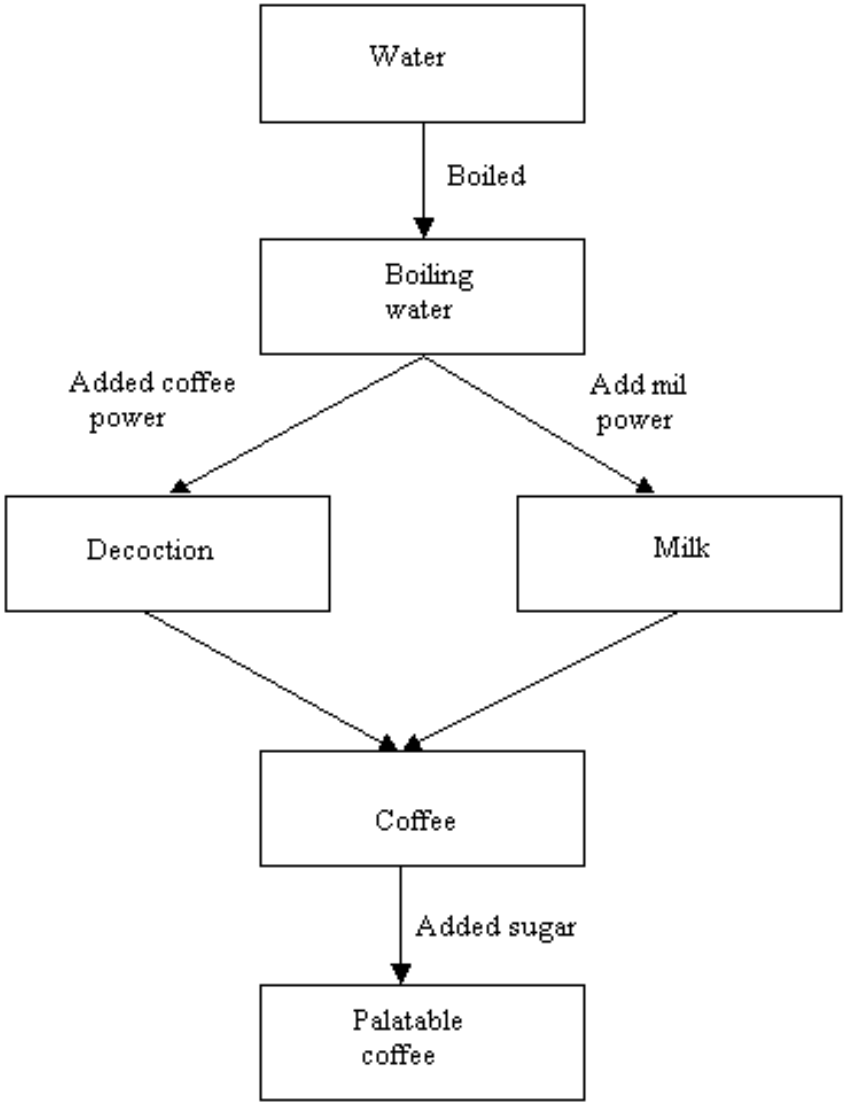
Take same of the boiled water in a cup add necessary amount of instant coffee pour to decoction.

Add milk pour to the remaining boiling water to make milk.

Milk decoction and milk.

Add sufficient quantity of sugar to your taste & coffee is ready.

SIMPLE EXAMPLE FOR THE STATE SPACE



Problem Characteristics

1. Is problem decomposable into set of(nearly) independent smaller or easier sub problems?
2. Can solution steps be ignored or at least undone if they prove unwise?
3. Is the problem's universe predictable?
4. Is a good solution to the problem obvious without comparison to all other possible solutions?
5. Is a desire solution a state of the world or a path to a state?
6. Is a large amount of knowledge absolute required to solve the problem, or is knowledge important only to certain the search?
7. Can a computer that is simply given the problem return the solution, or will the solution of problem require interaction between the computer and a person?

1. Is the problem Decomposable?

By this method we can solve large problem easily.

Ex: Decomposable problem

Symbolic Integration

$$\int (x^2 + 3x + \sin^2 x.\cos^2 x)\,dx$$

Can be divided to

Integral of x^2

Integral of $3x$

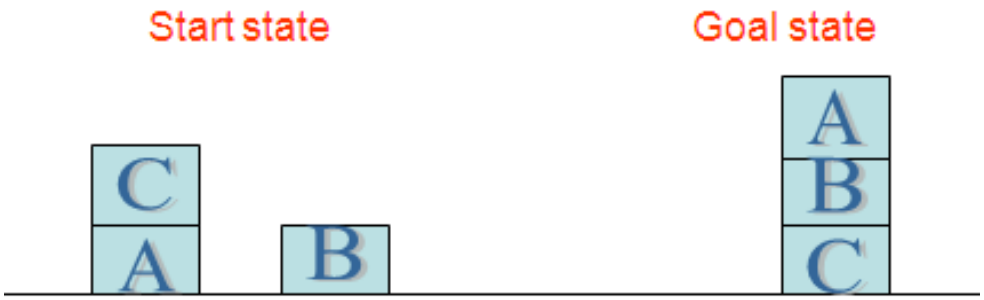
Integral of $\sin^2 x.\cos^2 x$, which can be further divided to $(1-\cos^2 x).\cos^2 x$ ….

Ex: Non- decomposable problems

Block World Problem

Assume that only two operations are available:

1. CLEAR(x)[Block x has nothing on it]->ON(x,Table)[Pick up x and put on the table]
2. Clear(x) and Clear(y)->ON(x,y)[Put x on y]



ON(B,C) and ON(A,B)

2. Can Solution steps be ignored or undone?

- **Ignorable problem:** in which solution steps can be ignored.

Ex:- Theorem Proving

Suppose we are trying to prove a mathematical theorem. We proceed by first proving a lemma that we think will be useful. Eventually, we realize that the lemma is not help at all.

Every thing we need to know to prove theorem is still true and in memory, if it ever was. Any rule that could have been applied at the outset can still be applied. *All we have lost is the effort that was spent exploring the blind alley.*

2. Can Solution steps be ignored or undone?

- Recoverable problem: in which solution steps can be undone.

Ex:- The 8-Puzzle

The 8-puzzle is a square tray in which are placed, eight square tiles and remaining 9th square is uncovered. Each tile has number on it. A tile that is adjacent to blank space can be slide in to that space. A game consist of a starting position and a specific goal position.

We might make stupid move.

We can backtrack and undo the first move. Mistakes can still be recovered from but not quite as easy as in theorem proving.

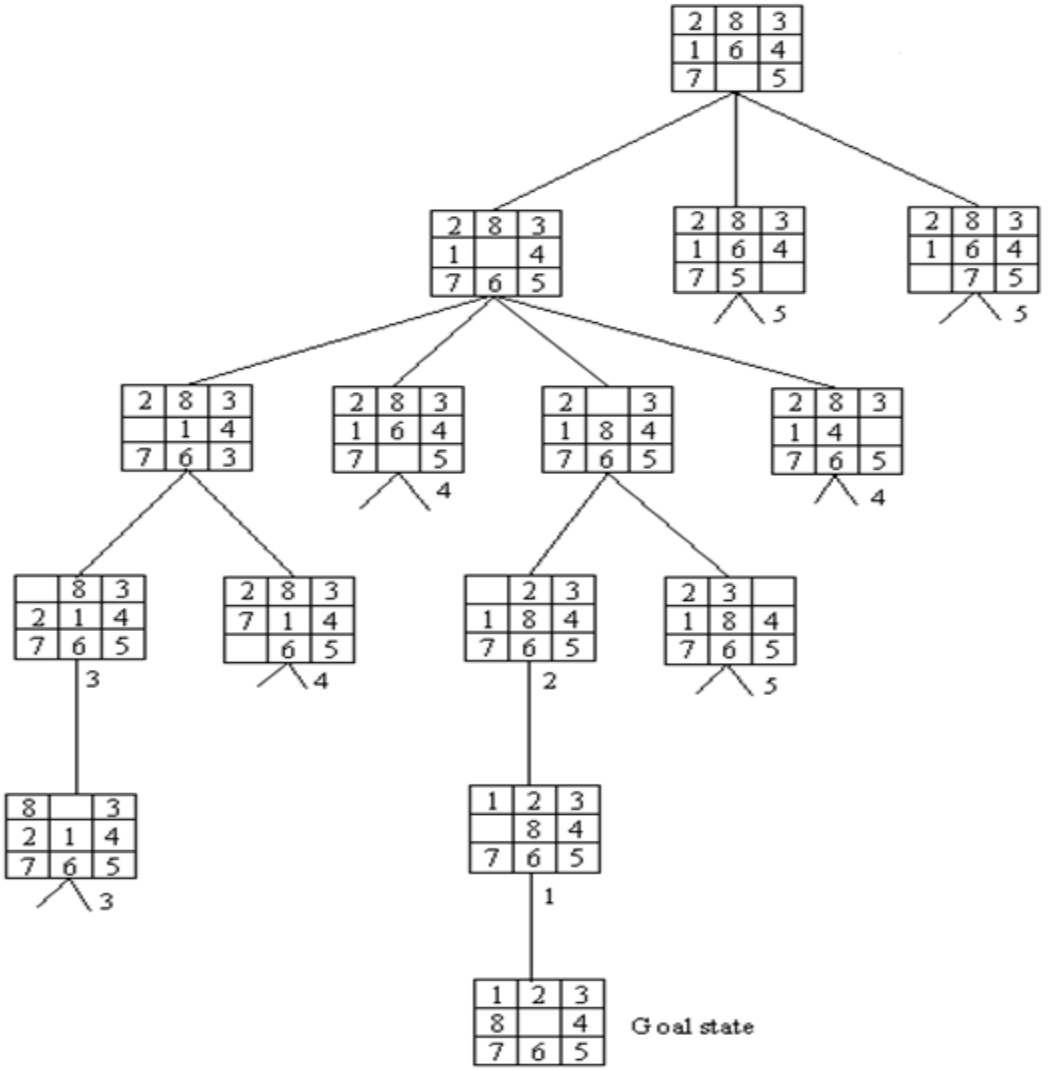
2	8	3
1	6	4
7		5

Initial State

1	2	3
8		4
7	6	5

Goal state

8-Puzzle



2. Can Solution steps be ignored or undone?

- Irrecoverable problem: in which solution steps cannot be undone.

Ex:- Chess

Suppose a chess playing program makes a stupid move and realize it a couple of move later. It cannot simply play as though it never made the stupid move. Nor can it simply backup and start the game over from that point. All it can do is to try to make best of the current situation and go on from there.

2. Can Solution steps be ignored or undone?

- **Ignorable problem** can be solved using a simple control structure that never backtracks. Such a control structure is easy to implement.
- **Recoverable problem** can be solved by slightly more complicated control strategy that does something mistakes and **backtracking** will be necessary to recover from such mistakes.
- **Irrecoverable** problems, solved by a system that expends a great deal of effort making each decision since each the decision must be final.
- **Some irrecoverable** problems can be solved by recoverable style methods used in a **planning process** , in which an entire sequence of steps is analyzed in advance to discover where it will lead before first step is actually taken.

3. Is universe predictable?

Certain-outcome problem

Ex: 8-Puzzle

Every time we make a move, we know exactly what will happen. This is possible to **plan** entire sequence of moves and be confident that we know what the resulting state will be.

Uncertain-outcome problem

Ex: play Bridge

One of the decisions we will have to make is which card to play on the first trick. What we would like to do is to plan entire hand before making the 1st hand. But now it is not possible to do such planning with certainty since we cannot know exactly where all the cards are or what the other players will do on their turn.

4. Is a good solution Absolute or Relative ?

Any-path problem

1. Deena is a man.
2. Deena is a worker in a company.
3. Deena was born in 1905.
4. All men are mortal.
5. All workers in a factory died when there was an accident in 1952.
6. No mortal lives longer than 100 years.

"Is Deena alive"

Solution 1:
1. Deena is a man.
2. Deena was born in 1915.
3. All men are mortal.
4. Now it is 2020, so Siva's age is 105 years.
5. No mortal lives longer than 100 years.

Solution 2:
1. Deena is a worker in the company.
2. All workers in the company died in 1952.

4. Is a good solution Absolute or Relative ?

Since all we are interested in is the answer to question, it does not matter **which path we follow**.

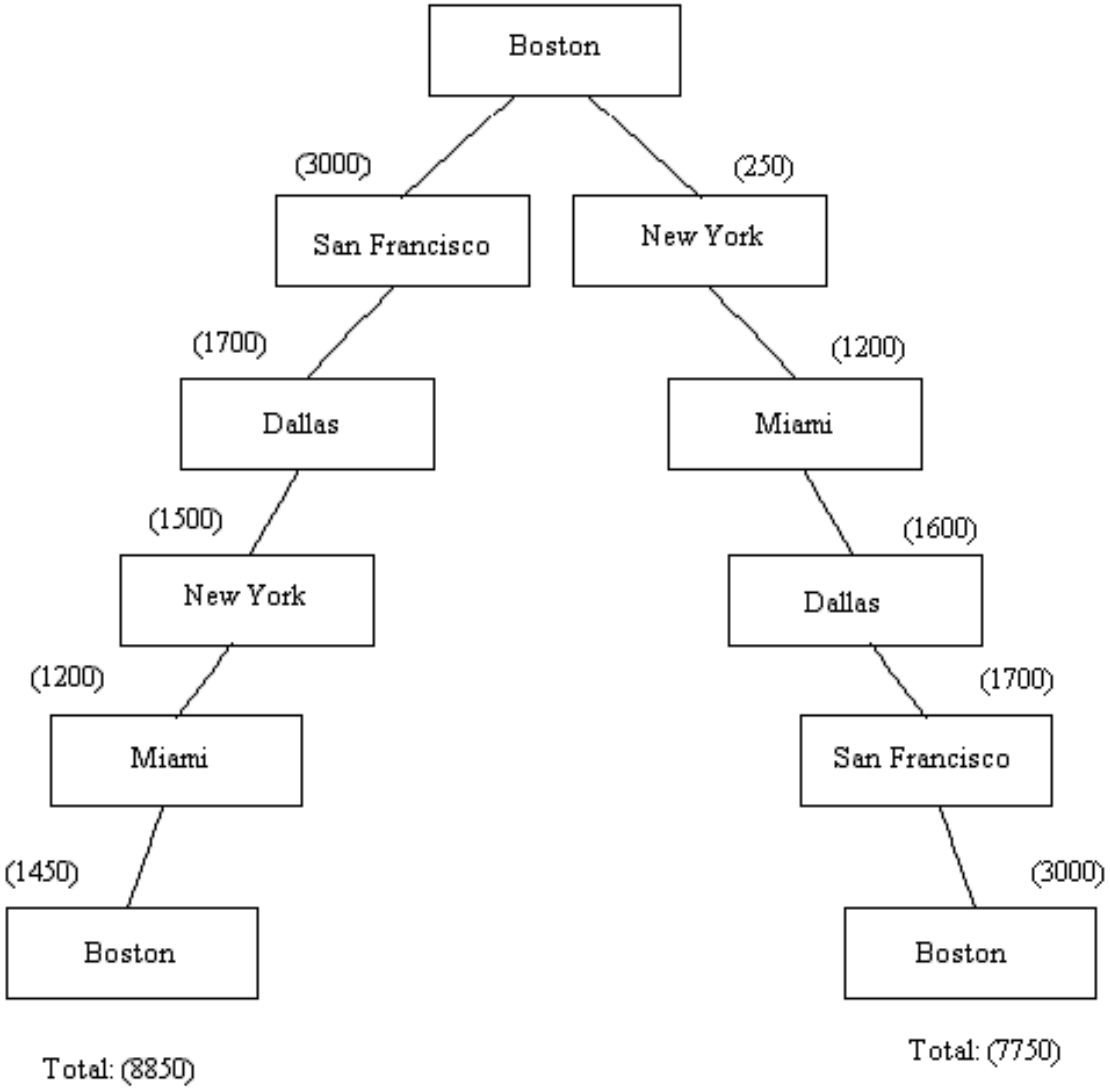
If we do follow one path successfully to the answer, there is no reason to go back and see if some other path might also lead to a solution.

Best-path problem

Ex: Traveling Salesman Problem

- Given a road map of n cities, find the **shortest** tour which visits every city on the map exactly once and then return to the original city (*Hamiltonian circuit*)

	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	


Boston
(3000)
San Francisco
(250)
New York
(1700)
Dallas
(1200)
Miami
(1500)
New York
(1600)
Dallas
(1200)
Miami
(1700)
San Francisco
(1450)
Boston
(3000)
Boston
Total: (8850)
Total: (7750)

4. Is a good solution Absolute or Relative ?

•Best-path problems are, in general, computationally harder than any-path problems.

•Any-path problems can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore. If the heuristics are not perfect, the search for a solution may not be as direct as possible, but that does not matter.

•For true best-path problems, however, no heuristic that could possibly miss the best solution can be used. So a much more exhaustive search will be performed.

5. Is the solution a State or Path ?

Solution is a path to state

Ex: Water jug problem

Here is not sufficient to report that we have solved the problem and the final state is (2,0).

Here we must report is not the final state but the path that we found to that state.

Thus a statement of solution to this problem must be a sequence of operations (some time called *apian*) that produce the final state.

Solution is a state of world

Ex: Natural language understanding

To solve the problem of finding the interpretation we need to produce interpretation itself. No record of processing by which the interpretation was found is necessary.

"The **bank** president ate a **dish** of pasta salad with the fork".

6. What is the role of knowledge?

Knowledge is important only to constrain the search for solution

Ex: playing chess

Suppose you have ultimate computing power available.

How much knowledge **would be required by a perfect program**?

just the rule for determining legal moves and some simple control mechanism that implement an appropriate search procedure.

Knowledge is required even to be able to recognize a solution

Ex: Scanning daily news paper to decide which are supporting the democrates and which are supporting the republicans in some upcoming elections.

you have ultimate computing power available.

How much knowledge **would be required by a perfect program?**

This time answer is great deal. It would have to know:

- The name of candidates in each party.
- For supporting republicans; you want to see done is have taxes lowered.
- For supporting democrats; you want to see done is improved education for minority students.

7. Does the task require interaction with person?

Solitary:

in which the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation of the reasoning process.

Level of interaction b/w computer and user is **problem-in solution-out.**

EX: Theorem Proving

Conversational:

in which there is intermediate communication between a person and the computer, either to prove additional assistance to computer or to prove additional information to user, or both.

Ex: Medical diagnosis

Chess

Problem characteristic	Satisfied	Reason
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	No	In actual game(not in PC) we can't undo previous steps
Is the problem universe predictable?	No	Problem Universe is not predictable as we are not sure about move of other player(second player)

Chess

Is a good solution absolute or relative?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative Solution : once you get one solution you have to find another possible solution to check which solution is best(i.e low cost). By considering this **chess is absolute**
Is the solution a state or a path?	Path	Is the solution a state or a path to a state? – For natural language understanding, some of the words have different interpretations .therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only , the workings are not necessary (i.e path to solution is not necessary) So In chess winning state(goal state) describe path to state

Chess

What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	Conversational In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or ***to provide additional information to the user***, or both. In chess additional assistance is not required

Water jug

Problem characteristic	Satisfied	Reason
Is the problem decomposable?	No	One Single solution
Can solution steps be ignored or undone?	Yes	
Is the problem universe predictable?	Yes	Problem Universe is predictable bcz to slove this problem it require only one person .we can predict what will happen in next step
Is a good solution absolute or relative?	absolute	**Absolute solution** , water jug problem may have number of solution , bt once we found one solution,no need to bother about other solution **Bcz it doesn't effect on its cost**
Is the solution a state or a path?	Path	Path to solution
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	Yes	additional assistance is required. Additional assistance, like to get jugs or pump

8 puzzle

Problem characteristic	Satisfied	Reason
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	Yes	We can undo the previous move
Is the problem universe predictable?	Yes	Problem Universe is predictable bcz to slove this problem it require only one person .we can predict what will be**position of blocks** in next move
Is a good solution absolute or relative?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative Solution : once you get one solution you have to find another possible solution to check which solution is best(i.e low cost). By considering this 8 puzzle **is absolute**

8 puzzle

Is the solution a state or a path?	Path	Is the solution a state or a path to a state? – For natural language understanding, some of the words have different interpretations .therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only , the workings are not necessary (i.e path to solution is not necessary) So In 8 puzzle winning state(goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	Conversational In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or ***to provide additional information to the user***, or both. In 8 puzzle additional assistance is not required

Missionaries and cannibals

Problem characteristic	Satisfied	Reason
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	Yes	
Is the problem universe predictable?	Yes	Problem Universe is not predictable as we are not sure about move of other player(second player)
Is a good solution absolute or relative?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative Solution : once you get one solution you have to find another possible solution to check which solution is best(i.e low cost). By considering this **is absolute**

Missionaries and cannibals

Is the solution a state or a path?	Path	Is the solution a state or a path to a state? – For natural language understanding, some of the words have different interpretations .therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only , the workings are not necessary (i.e path to solution is not necessary) So In winning state(goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	Yes	Conversational In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or ***to provide additional information to the user***, or both. In chess additional assistance is required to move **Missionaries to other side of river of other assistance is required**

Production systems

Production systems

Production systems provide appropriate structures for performing and describing search processes. A production system has four basic components as enumerated below.

- A set of rules
- A database of current facts established during the process of inference.
- A control strategy that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.
- A rule firing module.

1. **A set of production rules (PR):**

Left side determines the applicability of the rule and a right side describes the operation to be performed if the rule is applied.

2. **One or more knowledge/database:**

That contain whatever information is appropriate for the particular task (is called the working memory)

3. **A control structure:**

The control structure is strategy that specifies the order in which the rules will be compared to the database and also a way of resolving the conflicts that arise when several rules match at once.

- *the first requirement of a good control strategy is that it cause motion.*
- *The second requirement of a good control strategy is that it cause systematic.*

4. **Rule applier:**

A conflict may arise when more than one rule that can be fired in a situation of rule interpreter is to decide which is to be served of what is the order. The strategies used to resolve the conflict resolution strategies.

Features of Production System

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.

Production System Characteristics

- Production systems are a good way to describe the operations that can be performed in a search for a solution to a problem.

1. Can production systems, like problems, be described by a set of characteristics that shed some light on how they easily be implemented?
2. If so, what relationships are there b/w problem types and the types of production systems best suited to solve the problem.

Class of production Systems

- A **monotonic production system** In this type of a production system, the rules can be applied simultaneously as the use of one rule does not prevent the involvement of another rule that is selected at the same time.

- **A nonmonotonic production system is one in which this is not true.** The implementation of these systems does not require backtracking to correct the previous incorrect moves.

Class of production Systems

- A **partially commutative production system:** This class helps create a production system that can give the results even by interchanging the states of rules. If using a set of rules transforms State A into State B, then multiple combinations of those rules will be capable to convert State A into State B.

- A **commutative production system** is a production system that is both monotonic and partially commutative.

Relationship b/w problems and production systems

- For any solvable problem, there exist an infinite number of production systems that describe ways to find solution. Some will be more natural or efficient than other.
- Any problem that can be solved by any production system can be solved by a commutative one, but the commutative one may be so unwieldy as to be practically useless.
- So in formal sense, there is no relation ship b/w kind of problems and kind of production system since all problems can be solved by all kinds of system.
- But in practical sense, there definitely is such a relationships b/w kind of problems and kind of systems that lend themselves naturally to describing those problems.

2. Relationship b/w problems and production systems

Ignorable problems; where creating new things rather than changing old once

Change occur but can be reversed and in which order of operation is not critical

	Monotonic	**Nonmonotonic**
Partially Commutative	Theorem Proving	Robot Navigation, 8-puzzle
Not Partially Commutative	Chemical synthesis	Bridge, Chess

where creating new things by changing old once

Reverse not possible and order matter.

Partially commutative, monotonic production systems are useful for solving ignorable problems that involves creating new things rather than changing old ones generally ignorable. Theorem proving is one example of such a creative process partially commutative, monotonic production system are important for a implementation stand point because they can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed.

Non-monotonic, partially commutative production systems are useful for problems in which changes occur but can be reversed and in which order of operations is not critical. This is usually the case in physical manipulation problems such as “Robot navigation on a flat plane”. The 8-puzzle and blocks world problem can be considered partially commutative production systems are significant from an implementation point of view because they tend to read too much duplication of individual states during the search process.

Production systems that are not partially commutative are useful for many problems in which changes occur. For example “Chemical Synthesis”

Advantages of production system:

1.They enhance the heuristic control of search, which promotes adaptability.

2.The production rules can be modified without causing adverse effects in the system.

3.They offer an efficient way of solving problems in real-life situations.

4.They consist of 'IF-THEN' conditions that enhance simplicity in problem-solving.

5.They have reliable troubleshooting methods. It takes little time to locate and resolve conflicts in the system.

Disadvantages Of Production Systems In AI

- It's very difficult to analyse the flow of control within a production system
- There is an absence of learning due to a rule-based production system which does not store the result of the problem for future use.
- The rules in the production system should not have any type of conflict resolution as when a new rule is added to the database it should ensure that it does not have any conflict with any existing rules.


LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
L
P
U

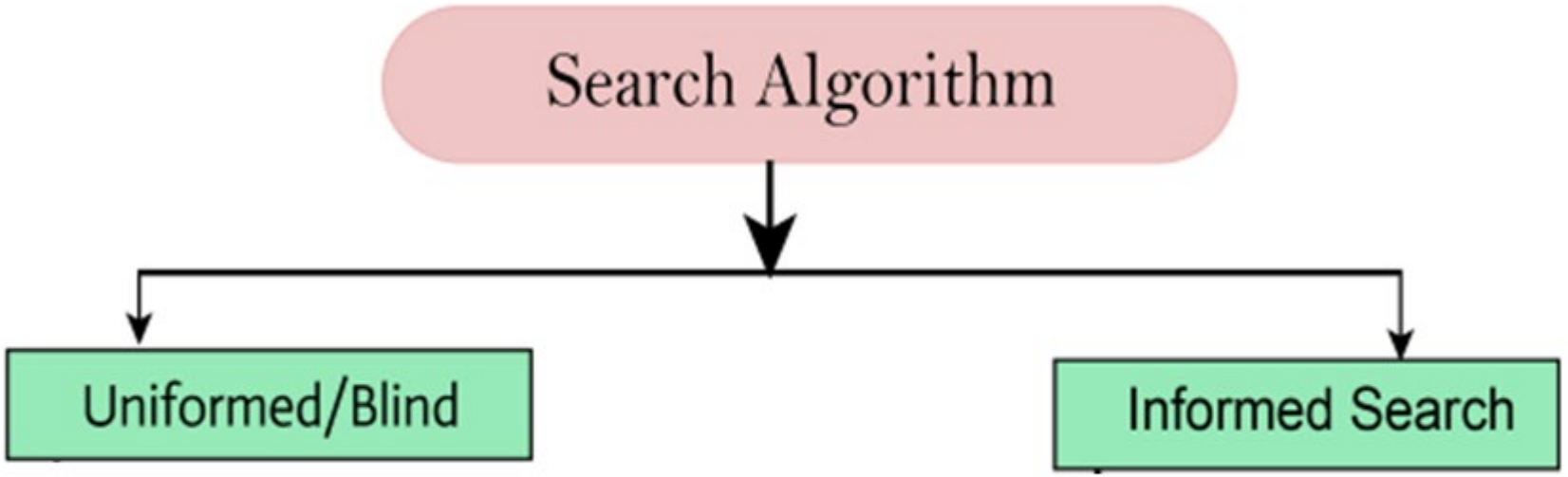
Search Terminology

- **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- **Problem Instance** − It is Initial state + Goal state.
- **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
- **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
- **Space Complexity** − The maximum number of nodes that are stored in memory.
- **Time Complexity** − The maximum number of nodes that are created.
- **Admissibility** − A property of an algorithm to always find an optimal solution.
- **Branching Factor** − The average number of child nodes in the problem space graph.

Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

- **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.
- **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
- **Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.
- **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.


Search Algorithm
Uniformed/Blind
Informed Search

Search Strategies

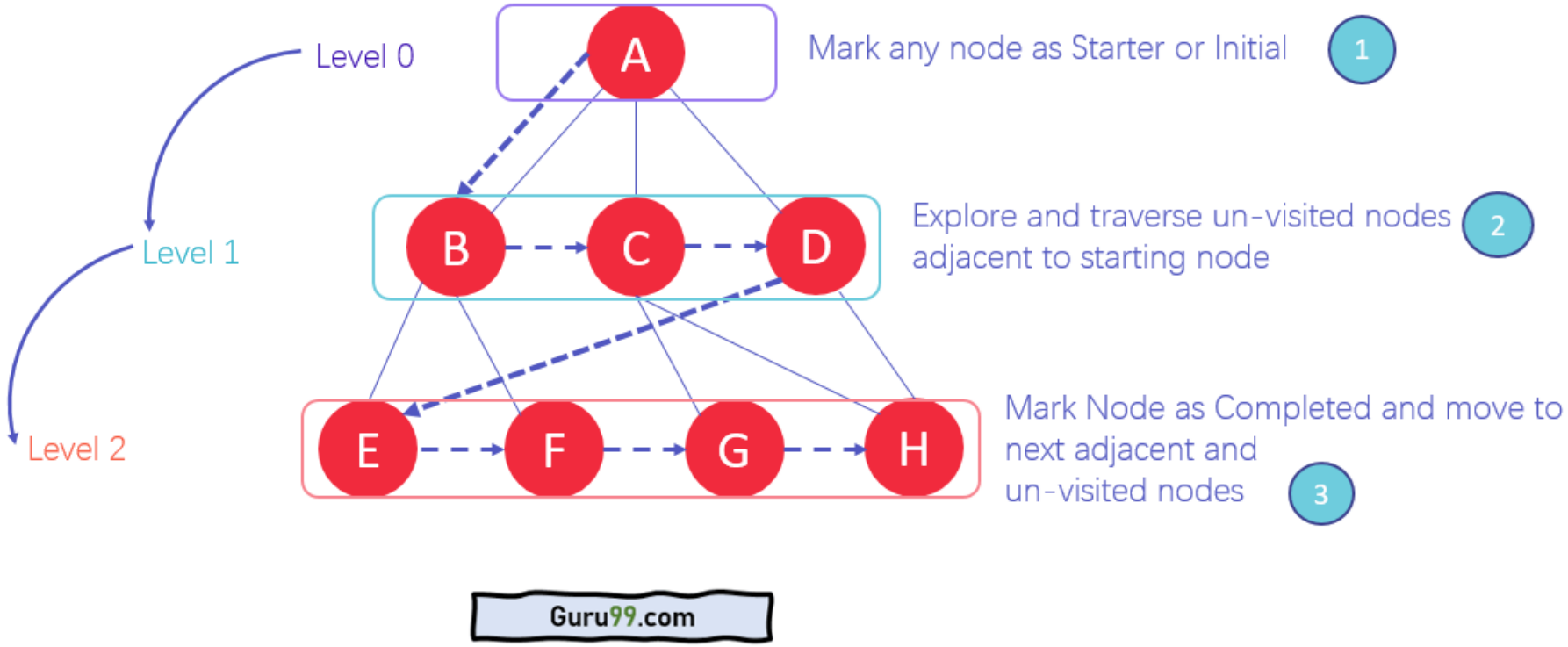
They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements −

- State description
- A set of valid operators
- Initial state
- Goal state description

Algorithm : Breadth-First Search

CONCEPT DIAGRAM

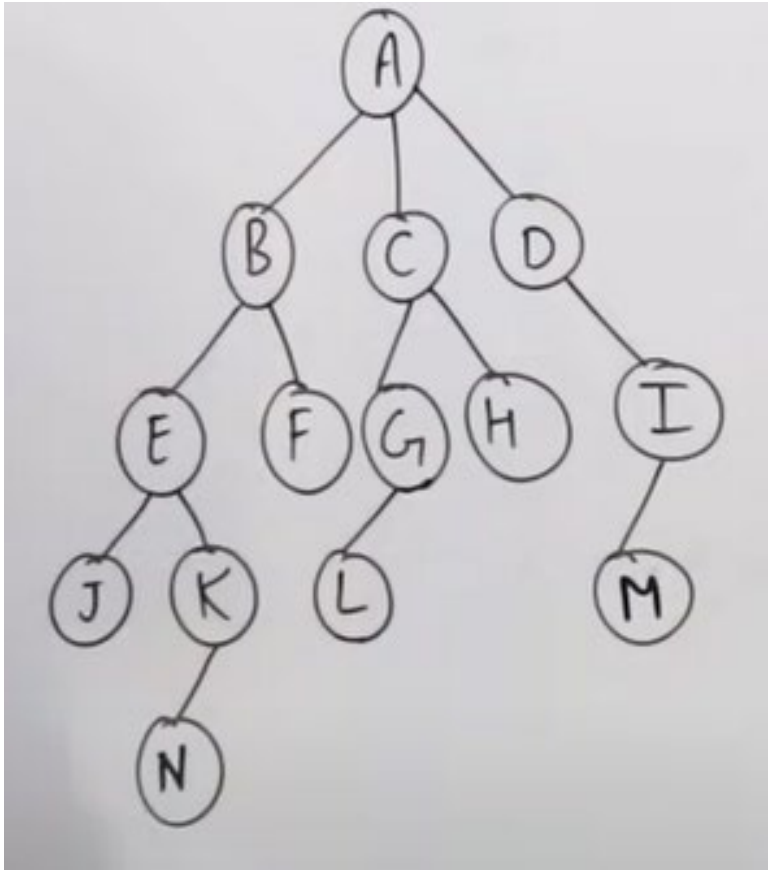


1.In the various levels of the data, you can mark any node as the starting or initial node to begin traversing. The BFS will visit the node and mark it as visited and places it in the queue.

2.Now the BFS will visit the nearest and un-visited nodes and marks them. These values are also added to the queue. The queue works on the FIFO model.

3.In a similar manner, the remaining nearest and un-visited nodes on the graph are analyzed marked and added to the queue. These items are deleted from the queue as receive and printed as the result.

Algorithm : Breadth-First Search




LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
L
P
U

Algorithm : Breadth-First Search

1. Create a variable called *NODE-LIST* and set it to the initial state.
2. Until a goal state is found or *NODE-LIST* is empty:

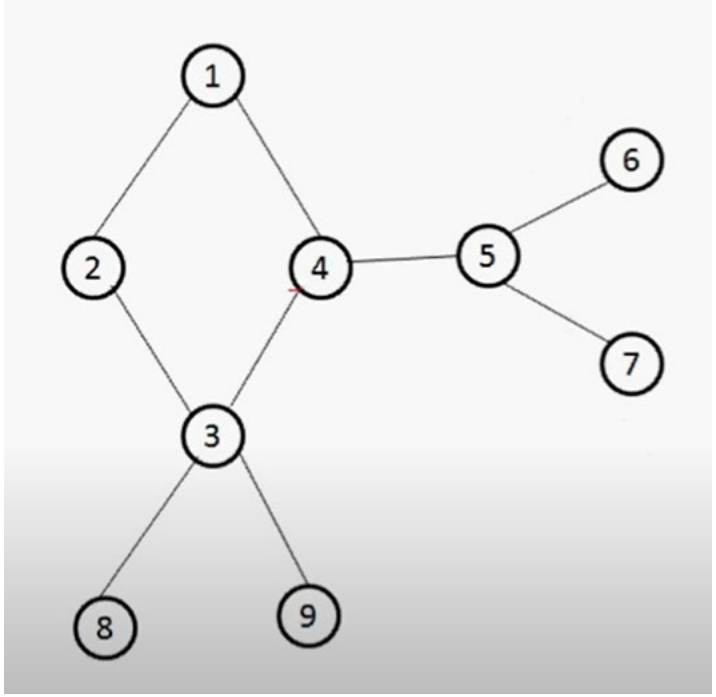
(a) Remove the first element from *NODE-LIST* and call it *E*. If *NODE-LIST* was empty, quit.

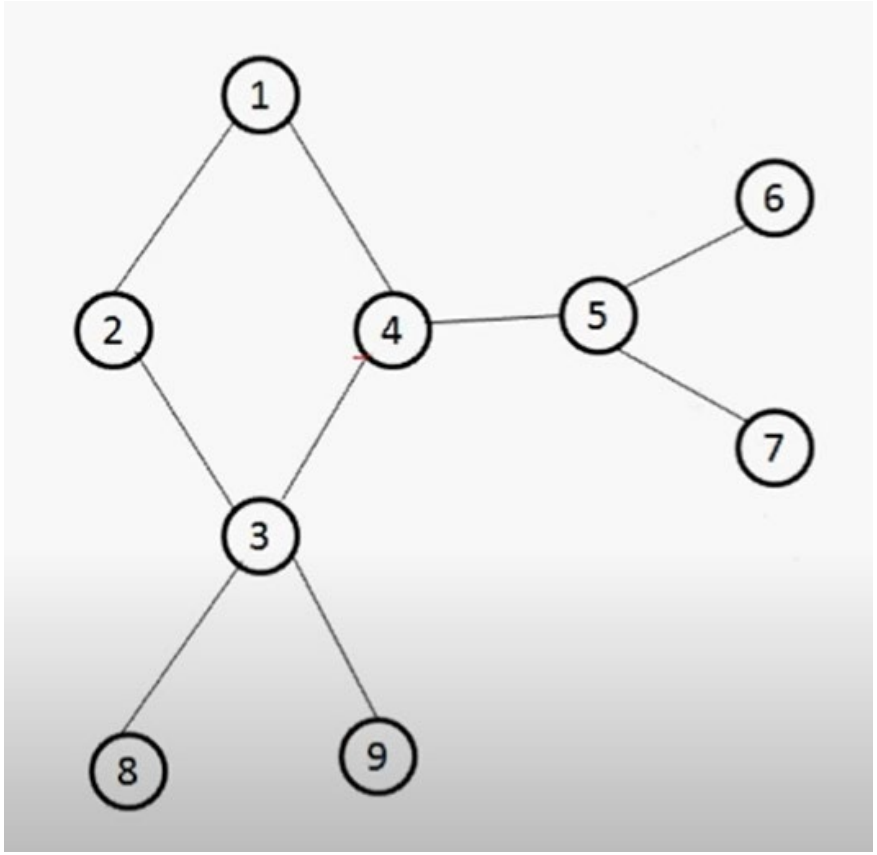
(b) For each way that each rule can match the state described in *E* do:

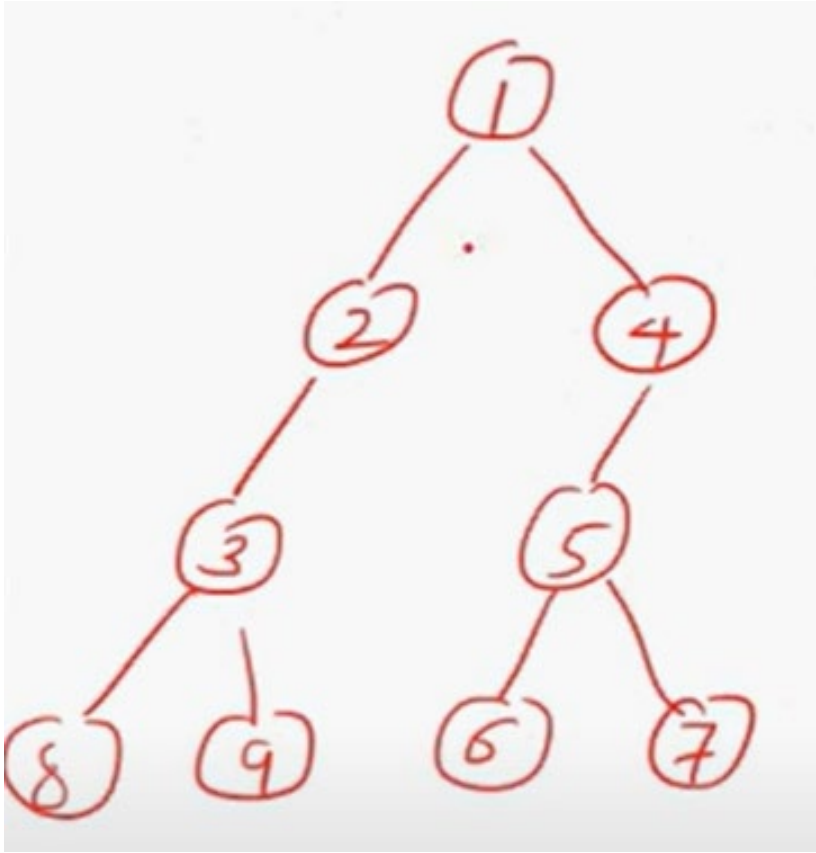
(i) Apply the rule to generate a new state,

(ii) If the new state is a goal state, quit and return this state.

(iii) Otherwise, add the new state to the end of NODE-LIST.


1
2
3
4
5
6
7
8
9


1
2
4
5
6
7
3
8
9

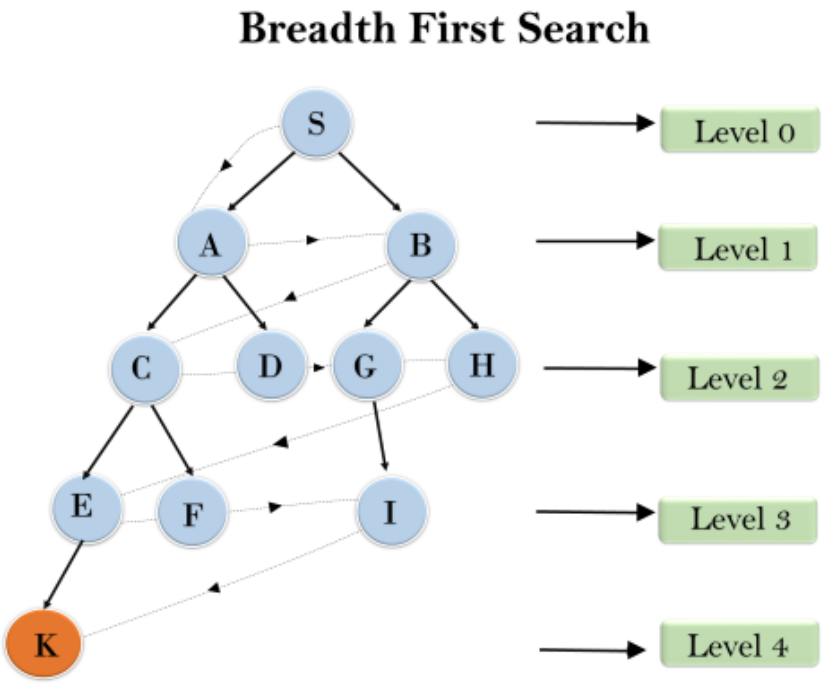

1
2
4
3
5
8
9
6
7


LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
L
P
U

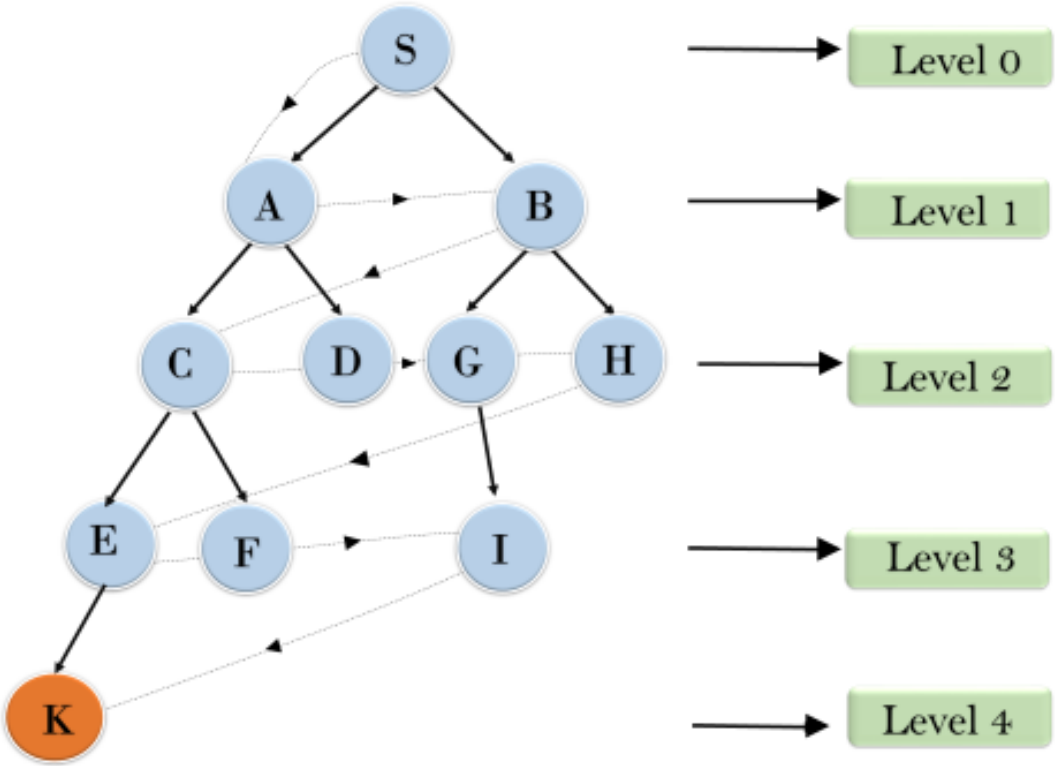
Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

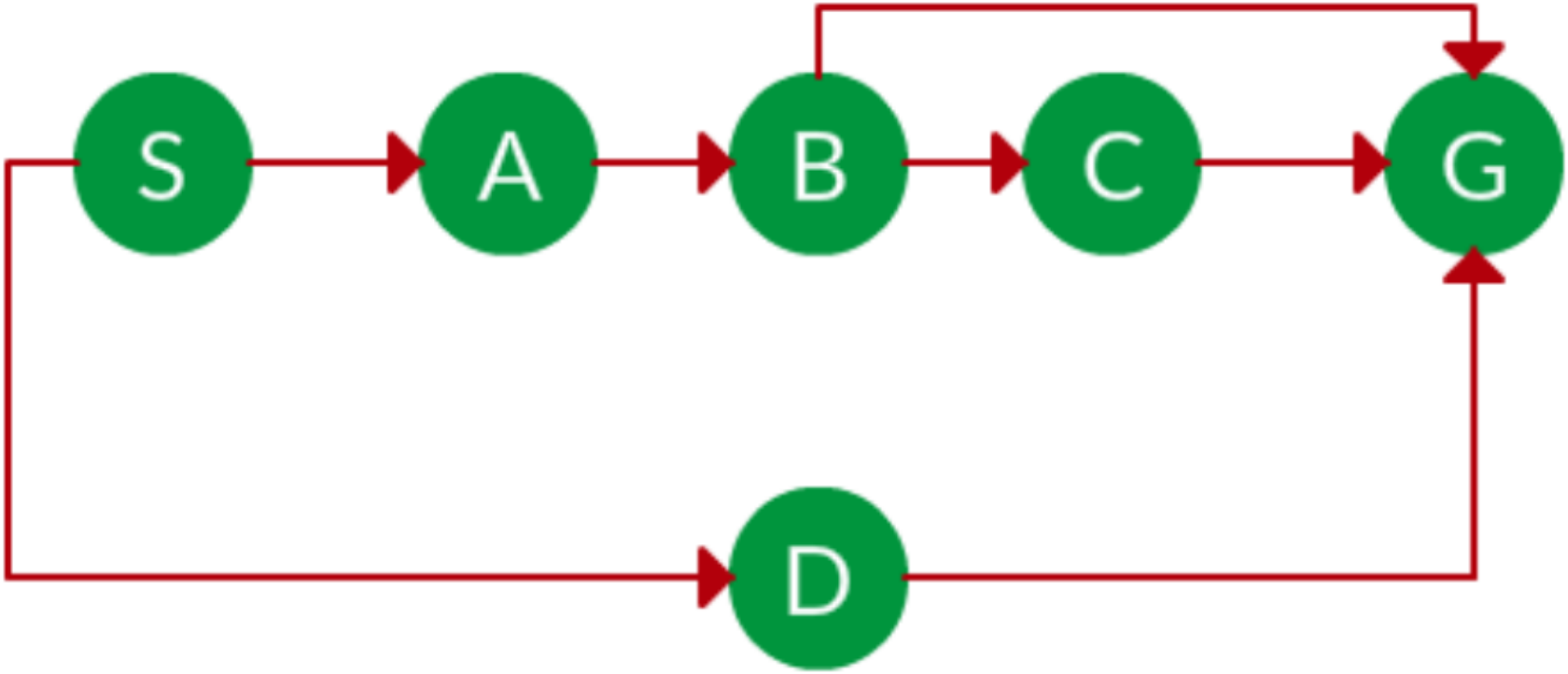


Breadth First Search

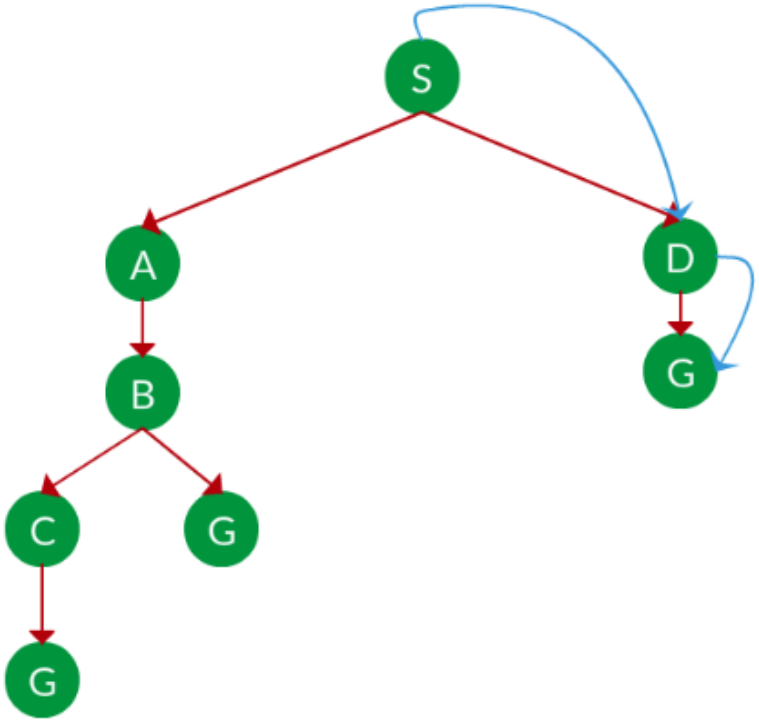


Example:

Question. Which solution would BFS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.....+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

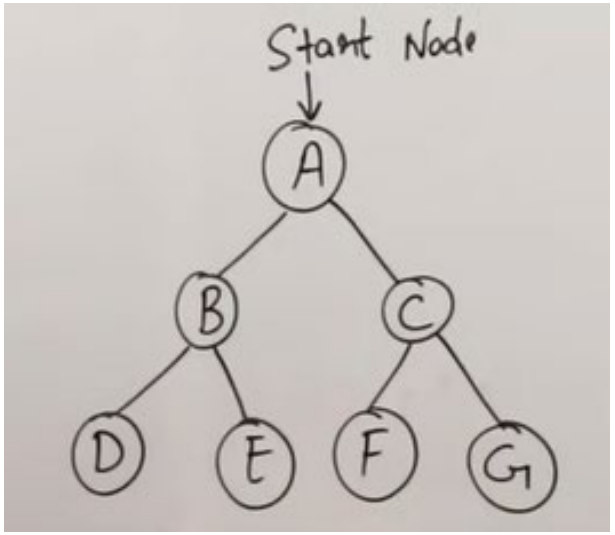
Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Algorithm: Depth-first Search

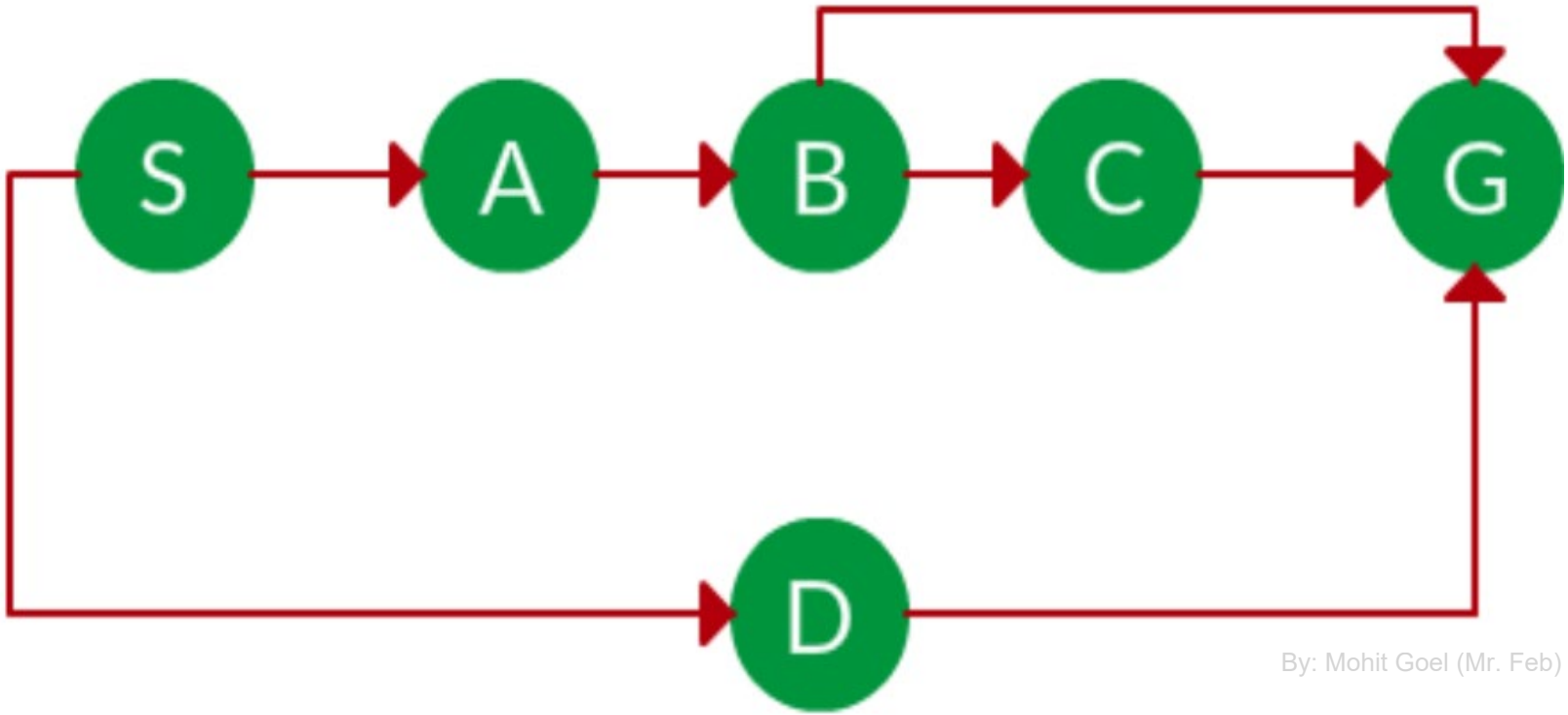
- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.

Algorithm : Depth-First Search

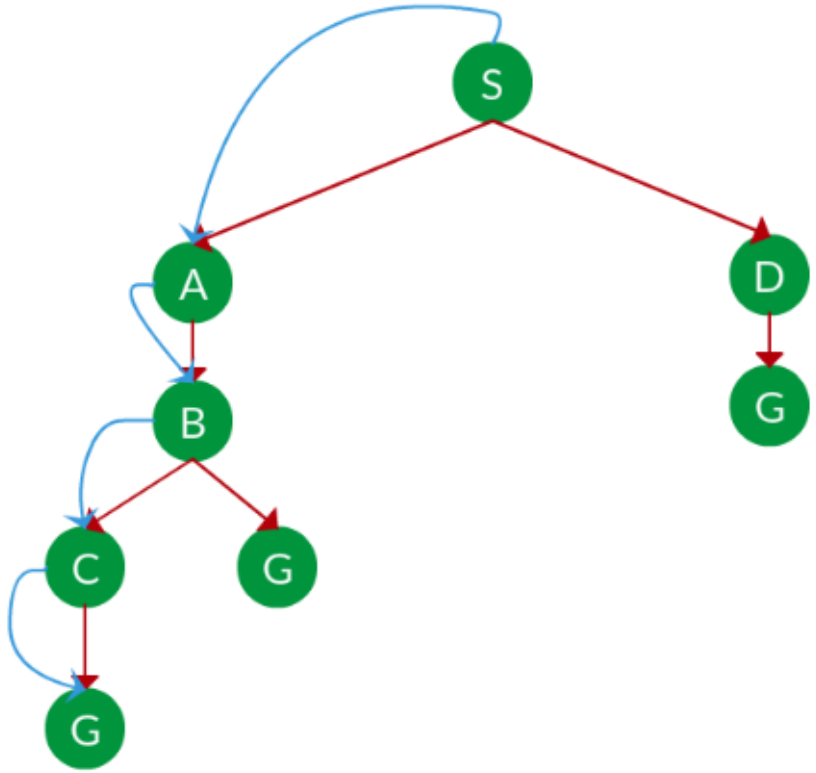


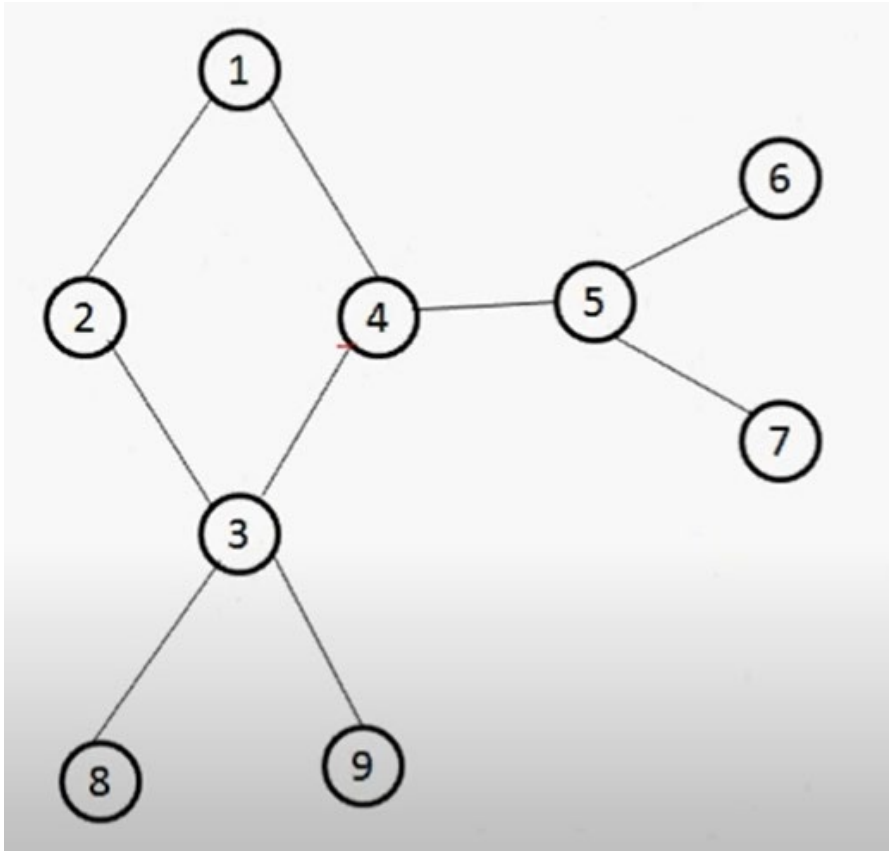
Algorithm : Depth-First Search

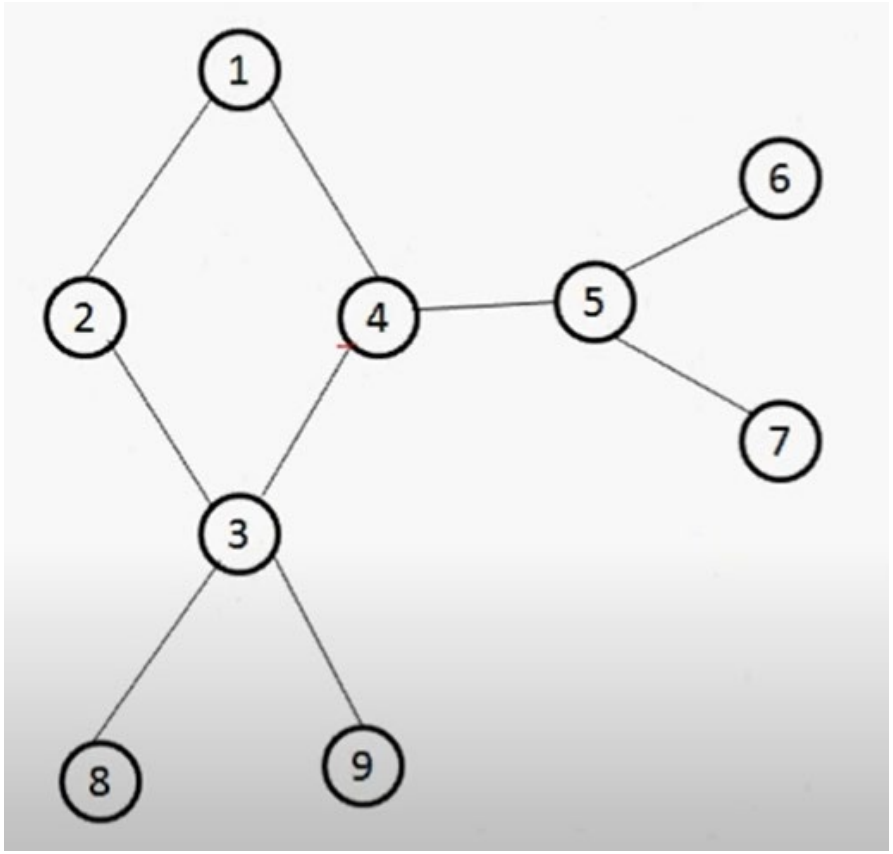
Question. *Which solution would DFS find to move from node S to node G if run on the graph below?*

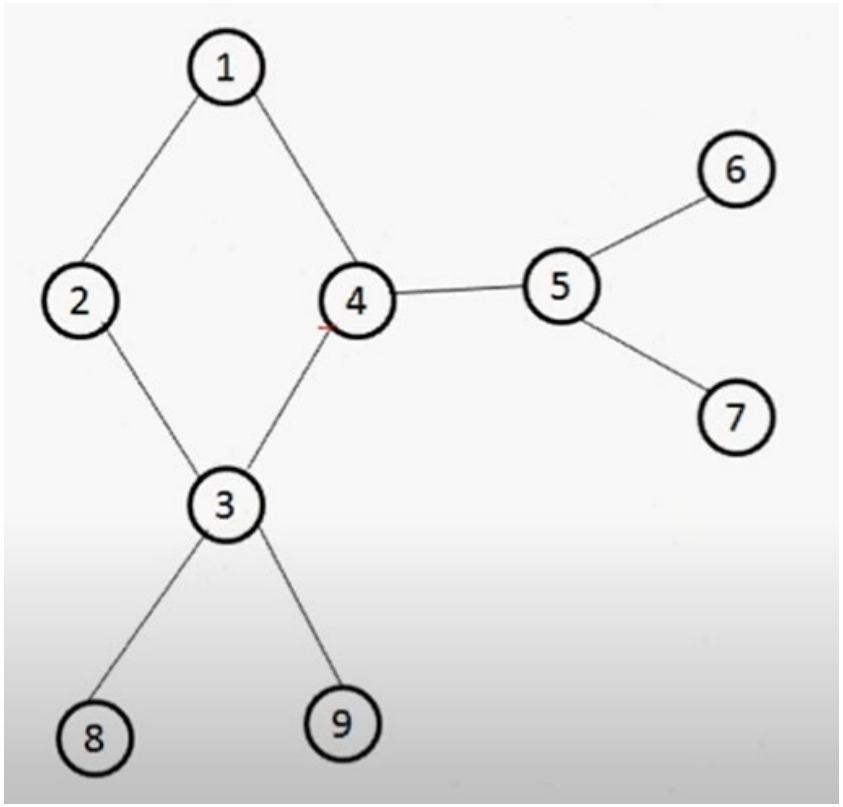


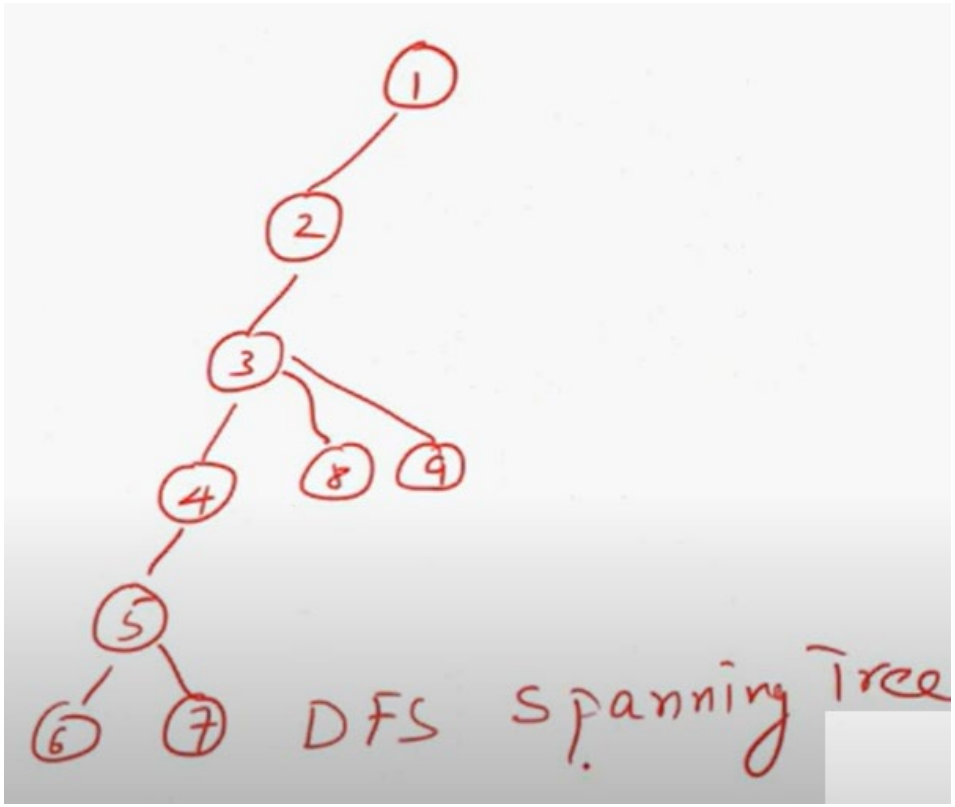
Solution. The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.




1
2
3
4
5
6
7
8
9


1
2
3
4
5
6
7
8
9


1
2
3
4
5
6
7
8
9


1
2
3
4
8
9
5
6
7
DFS spanning Tree

Algorithm : Depth-First Search

1. If the initial state is a goal state, quit and return success.

2. Otherwise, do the following until success or failure is signaled:

   (a)Generate a successor, *E,* of the initial state. If there are no more successors, signal failure.

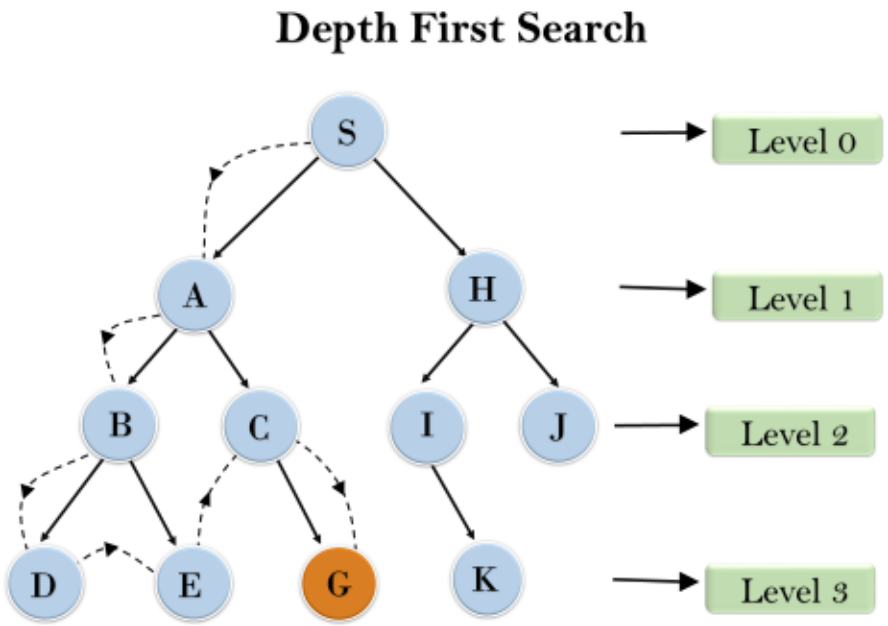
   (b) Call Depth-First Search with *E* as the initial state.

   (c) If success is returned, signal success. Otherwise continue in this loop.

Example:

In the below search tree, we have shown the flow of depth-first search

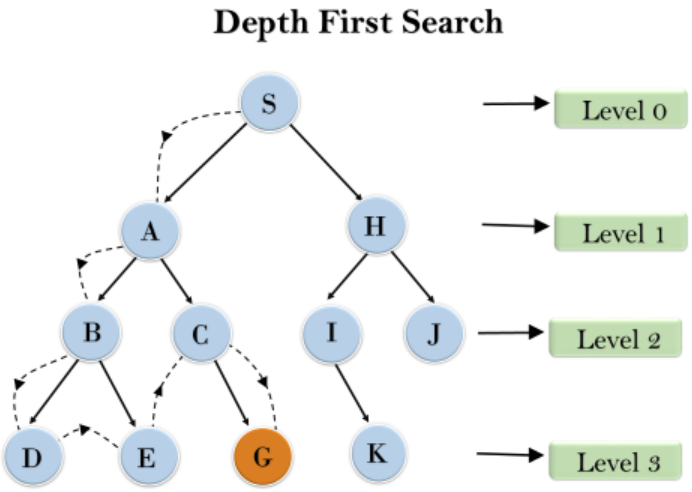
It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



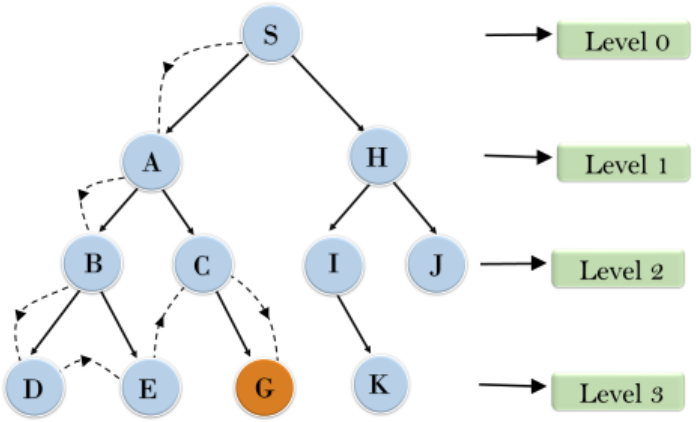
Example:

In the below search tree, we have shown the flow of depth-first search

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Depth First Search



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n)= 1+ b^2+ b^3 +.....+ b^d=O(b^d)$$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bd)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Bidirectional Search Algorithm:

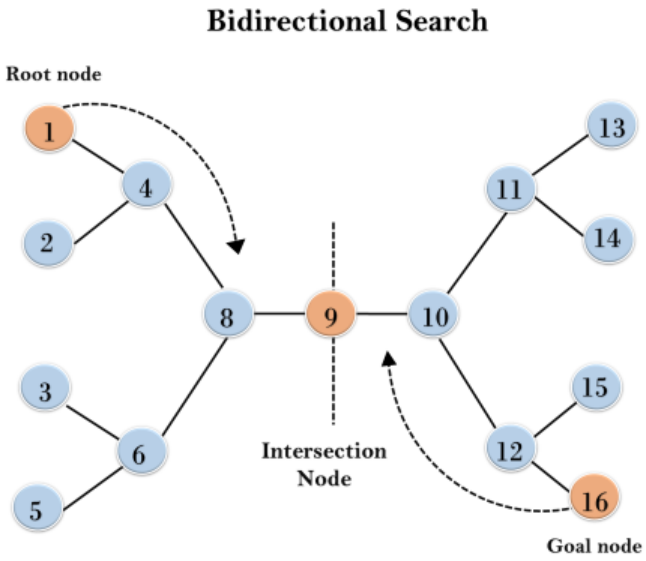
Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, etc.

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.



When to use bidirectional approach?

We can consider bidirectional approach when-

1.Both initial and goal states are unique and completely defined.

2.The branching factor is exactly the same in both directions.

Why bidirectional approach?

Because in many cases it is faster, it dramatically reduce the amount of required exploration.

Suppose if branching factor of tree is b and distance of goal vertex from source is d, then the normal BFS/DFS searching complexity would be $O(b^d)$. On the other hand, if we execute two search operation then the complexity would be $O(b^{d/2})$ for each search and total complexity would be $O(b^{d/2} + b^{d/2})$ which is far less than $O(b^d)$.

Bidirectional Search Algorithm:

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.

Performance measures

•Completeness : Bidirectional search is complete if BFS is used in both searches.
•Optimality : It is optimal if BFS is used for search and paths have uniform cost

Iterative deepening Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Iterative deepening Search:

Advantages:

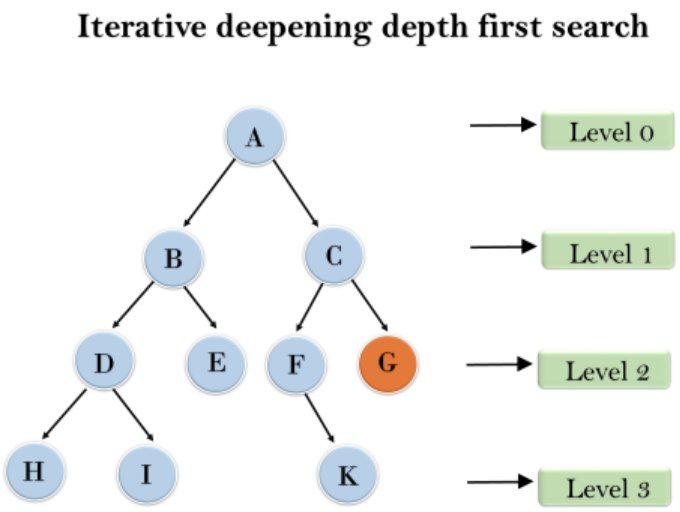
It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

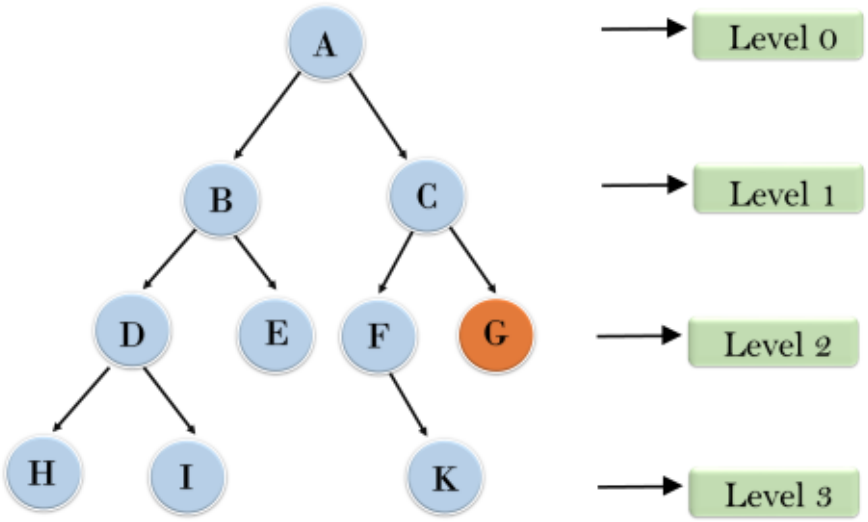
The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



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.

Completeness:

This algorithm is complete is if the branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is $O(b^d)$.

Space Complexity: The space complexity of IDDFS will be **O(bd)**.

Optimal:
IDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

Issues in the design of search programs

1.The direction in which to conduct search (forward versus backward reasoning). If the search proceeds from start state towards a goal state, it is a forward search or we can also search from the goal.

2.How to select applicable rules (Matching). Production systems typically spend most of their time looking for rules to apply. So, it is critical to have efficient procedures for matching rules against states.

3.How to represent each node of the search process (knowledge representation problem).

One other issue we should consider at this point is that of search trees versus search graphs. As mentioned above, we can think of production rules as generating nodes in a search tree. Each node can be expanded in turn, generating a set of successors. This process continues until a node representing a solution is found. Implementing such a procedure requires little bookkeeping. However, this process often results in the same node being generated as part of several paths and so being processed more than once. This happens because the search space may really be an arbitrary directed graph rather than a tree.

Principles of a production system by example: Water Jug Problem

Problem: given two water jugs (4 liter & 3 liter); neither has any measuring markers on it. there is a pump that can use to fill the jugs with water. **How can you get exactly 2 liter water in 4 liter jug?**

By: Mohit Goel(Mr. Feb)

1	(x,y) is X<4 → (4,Y)	Fill the 4-gallon jug
2	(x,y) if Y<3 → (x,3)	Fill the 3-gallon jug
3	(x,y) if x>0 → (x-d,Y)	Pour some water out of the 4-gallon jug.
4	(x,y) if Y>0 → (x,Y-d)	Pour some water out of 3-gallon jug.
5	(x,y) if x>0 → (0,y)	Empty the 4-gallon jug on the ground
6	(x,y) if y>0 → (x,0)	Empty the 3-gallon jug on the ground
7	(x,y) if X+Y >= 4 and y>0 → (4,y-(4-x))	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8	(x,y) if X+Y>=3 and x>0 → (x-(3-y),3))	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9	(x,y) if X+Y <=4 and y>0 → (x+y,0)	Pour all the water from the 3-gallon jug into the 4-gallon jug.
10	(x,y) if X+Y<=3 and x>0 → (0,x+y)	Pour all the water from the 4-gallon jug into the 3-gallon jug.
11	(0,2) → (2,0)	Pour the 2-gallon water from 3-gallon jug into the 4-gallon jug.
12	(2,Y) → (0,y)	Empty the 2-gallon in the 4-gallon jug on the ground.

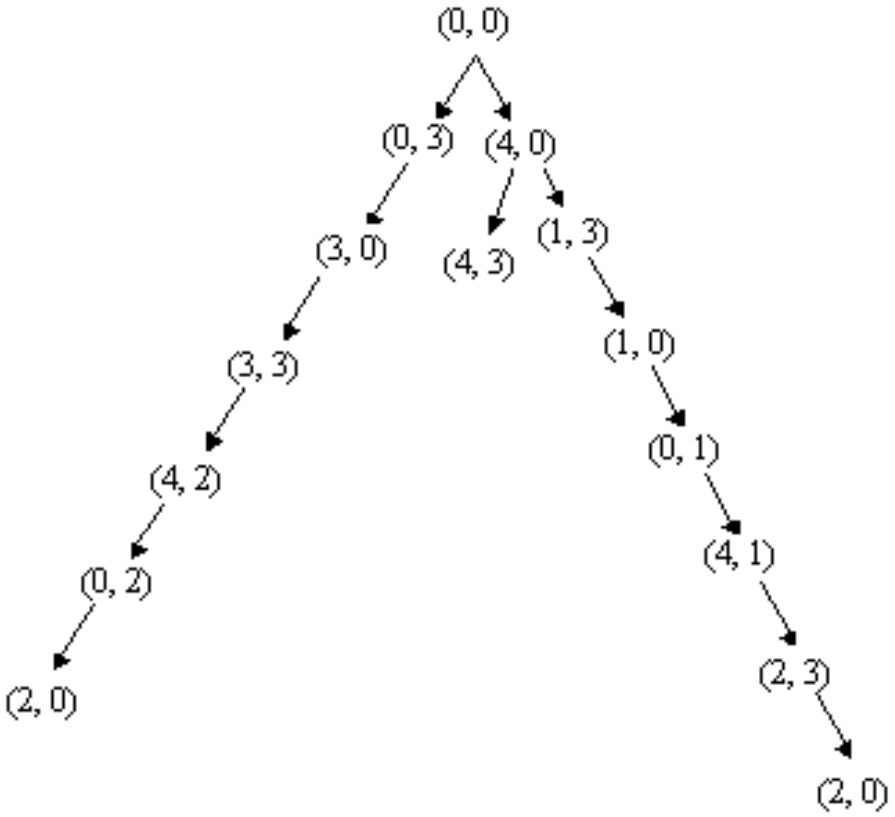
By: Mohit Goel(Mr. Feb)

One solution

4L	3L	rule applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

To keep track of the reasoning process, we draw a state-space for the problem. The leaves generated after firing of the rules should be stored in WM.

By: Mohit Goel(Mr. Feb)



By: Mohit Goel(Mr. Feb)

Example: Missionaries and Cannibals

The Missionaries and Cannibals problem illustrates the use of state space search for planning under constraints:

Three missionaries and three cannibals wish to cross a river using a two person boat. If at any time the cannibals outnumber the missionaries on either side of the river, they will eat the missionaries. How can a sequence of boat trips be performed that will get everyone to the other side of the river without any missionaries being eaten?

By: Mohit Goel(Mr. Feb)

State representation:

1. BOAT position: original (T) or final (NIL) side of the river.
2. Number of Missionaries and Cannibals on the original side of the river.
3. Start is (T 3 3); Goal is (NIL 0 0).

By: Mohit Goel(Mr. Feb)

Operators:

(MM 2 0)	Two Missionaries cross the river.
(MC 1 1)	One Missionary and one Cannibal.
(CC 0 2)	Two Cannibals.
(M 1 0)	One Missionary.
(C 0 1)	One Cannibal.

By: Mohit Goel(Mr. Feb)

By: Mohit Goel(Mr. Feb)

Missionaries/Cannibals Search Graph