Introduction to Artificial Intelligence

What is Artificial Intelligence?

- Artificial intelligence (AI) is wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.
- A branch of Computer Science named *Artificial Intelligence* pursues creating the computers or machines as intelligent as human beings.
- An intelligent entity created by humans.
- Capable of performing tasks intelligently without being explicitly instructed.
- Capable of thinking and acting rationally and humanely.

- What is Artificial Intelligence?
- According to the father of Artificial Intelligence, John McCarthy, it is *“The science and engineering of making intelligent machines, especially intelligent computer programs”.*

- Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.

- AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

Some Definitions (I)

The exciting new effort to make
computers think …
machines with minds,
in the full literal sense.

Haugeland, 1985

What is Artificial Intelligence?

- **Systems that think like humans**
  - Cognitive Modeling Approach
  - "The automation of activities that we associate with human thinking..."
  - Bellman 1978

- **Systems that think rationally**
  - "Laws of Thought" approach
  - "The study of mental faculties through the use of computational models"
  - Charniak and McDermott

- **Systems that act like humans**
  - Turing Test Approach
  - "The art of creating machines that perform functions that require intelligence when performed by people"
  - Kurzweil 1990

- **Systems that act rationally**
  - Rational Agent Approach
  - "The branch of CS that is concerned with the automation of intelligent behavior"
  - Lugar and Stubblefield

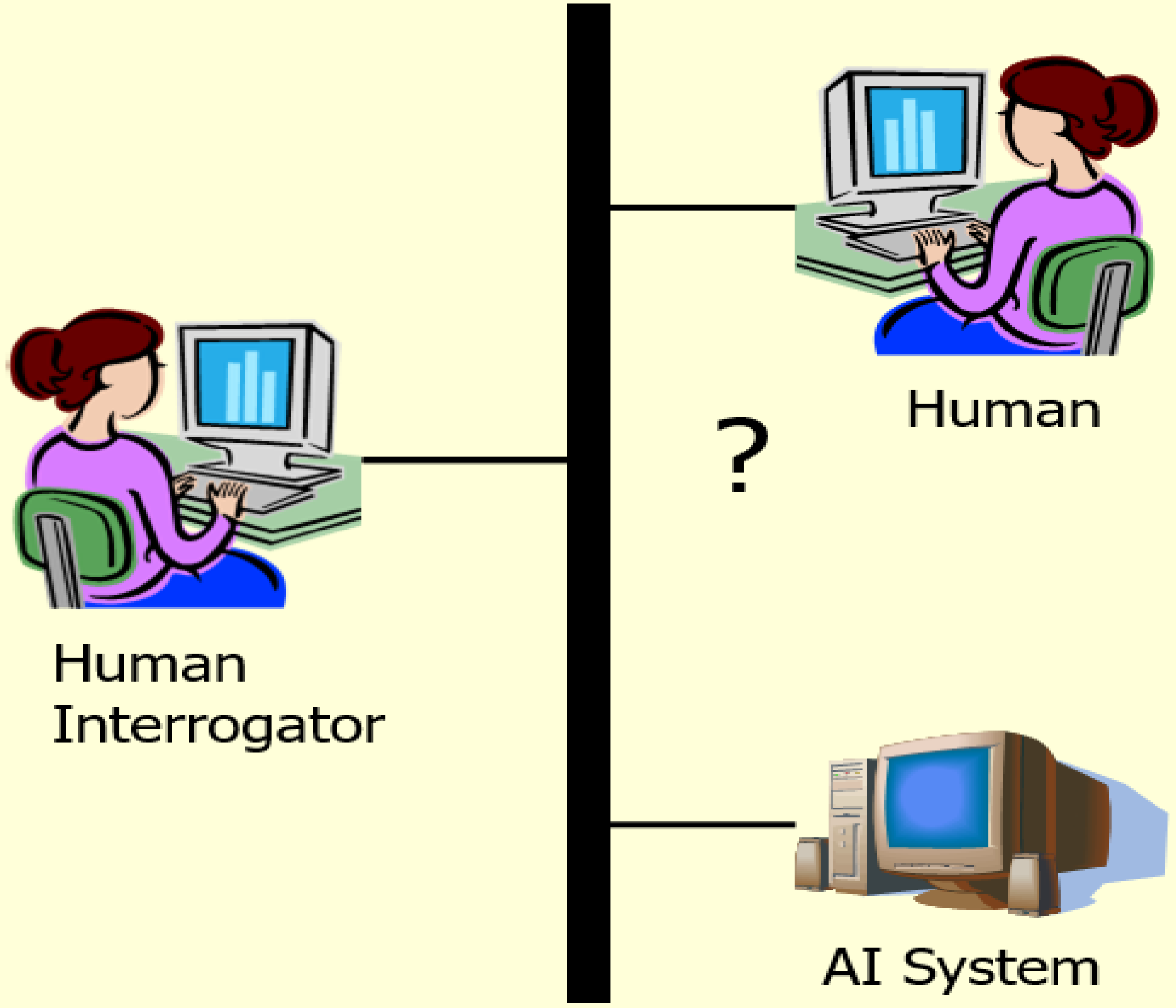
•Goals of AI

- **To Create Expert Systems** − The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** − Creating systems that understand, think, learn, and behave like humans.

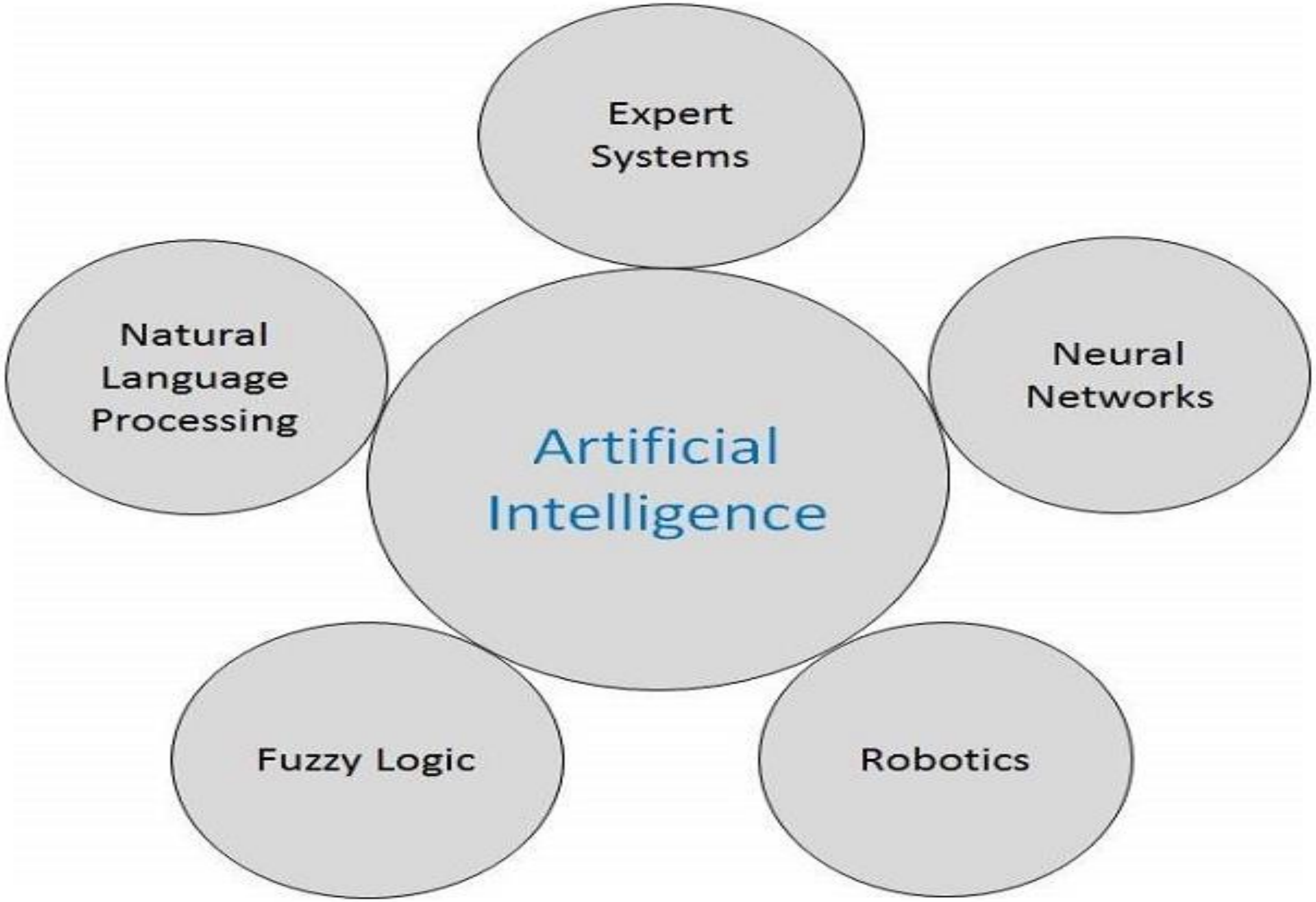
Acting Humanly

- The Turing Test (1950)
  - Can machines think?
  - Can machines behave intelligently?

- Operational test for intelligent behavior
  - The Imitation Game



The domain of artificial intelligence is huge in breadth and width. While proceeding, we consider the broadly common and prospering research areas in the domain of AI –



Sr.No.	Research Areas	Real Life Application
1	**Expert Systems** Examples − Flight-tracking systems, Clinical systems.	
2	**Natural Language Processing** Examples: Google Now feature, speech recognition, Automatic voice output.	
3	**Neural Networks** Examples − Pattern recognition systems such as face recognition, character recognition, handwriting recognition.	
4	**Robotics** Examples − Industrial robots for moving, spraying, painting, precision checking, drilling, cleaning, coating, carving, etc.	
5	**Fuzzy Logic Systems** Examples − Consumer electronics, automobiles, etc.	

Task Domains of Artificial Intelligence		
Mundane (Ordinary) Tasks	**Formal Tasks**	**Expert Tasks**
Perception • Computer Vision • Speech, Voice	• Mathematics • Geometry • Logic • Integration and Differentiation	• Engineering • Fault Finding • Manufacturing • Monitoring
Natural Language Processing • Understanding • Language Generation • Language Translation	Games • Go • Chess (Deep Blue) • Ckeckers	Scientific Analysis
Common Sense	Verification	Financial Analysis
Reasoning	Theorem Proving	Medical Diagnosis
Planing		Creativity
Robotics • Locomotive		

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.
- **Depth** – Length of the shortest path from initial state to goal state.

Problems, Problem Spaces and Search

To build a system to solve a problem

1. Define the problem precisely
2. Analyse the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best problem-solving techniques and apply it to the particular problem.

Search Space Definitions

- Problem formulation
  - Describe a general problem as a search problem
- Solution
  - Sequence of actions that transitions the world from the initial state to a goal state
- Solution cost (additive)
  - Sum of the cost of operators
  - Alternative: sum of distances, number of steps, etc.
- Search
  - Process of looking for a solution
  - Search algorithm takes problem as input and returns solution
  - We are searching through a space of possible states
- Execution
  - Process of executing sequence of actions (solution)

- A state space represents a problem in terms of states and operators that change states.
- A state space consists of:
- A representation of the states the system can be in.
- For example, in a board game, the board represents the current state of the game.
- A set of operators that can change one state into another state. In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.
- An initial state.
- A set of final states; some of these may be desirable, others undesirable. This set is often represented implicitly by a program that detects terminal states.

- 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. 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.

- 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.
- 1 Search proceeds with different types of ‘search control strategies’.
- 2 The depth-first search and breadth-first search are the two common search strategies.

Example: Eight puzzle (8-Puzzle)

The 8-puzzle is a 3 × 3 array containing eight square pieces, numbered 1 through 8, and one empty space. A piece can be moved horizontally or vertically into the empty space, in effect exchanging the positions of the piece and the empty space. There are four possible moves, UP (move the blank space up), DOWN, LEFT and RIGHT. The aim of the game is to make a sequence of moves that will convert the board from the start state into the goal state:

2	3	4
8	6	2
7		5

Initial State

1	2	3
8		4
7	6	5

Goal State

This example can be solved by the operator sequence UP, RIGHT, UP, LEFT, DOWN.

Playing 8-Puzzle

Problem Solving as State Space Search

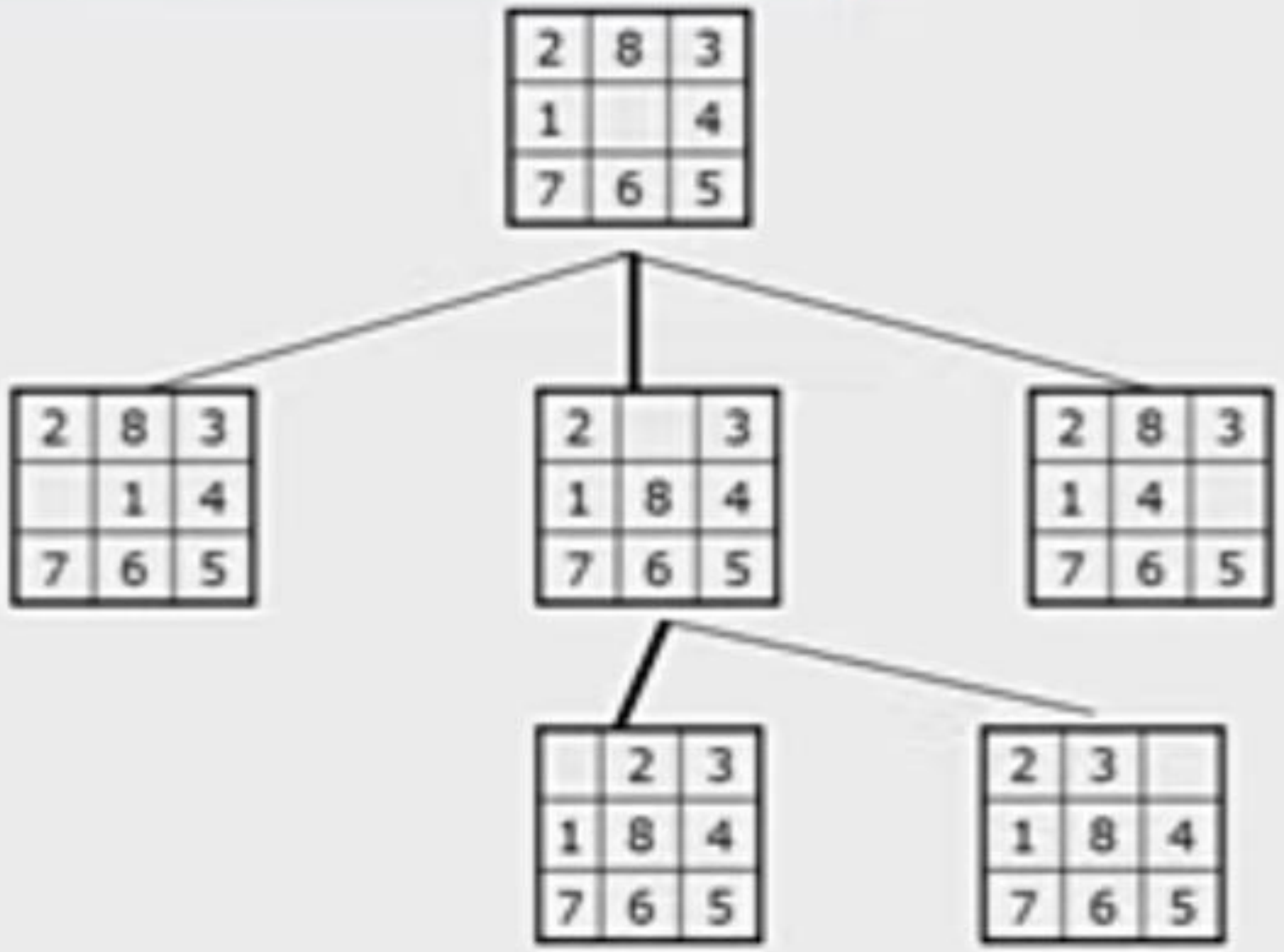
s ; Start Node

2	8	3
1	6	4
7		5

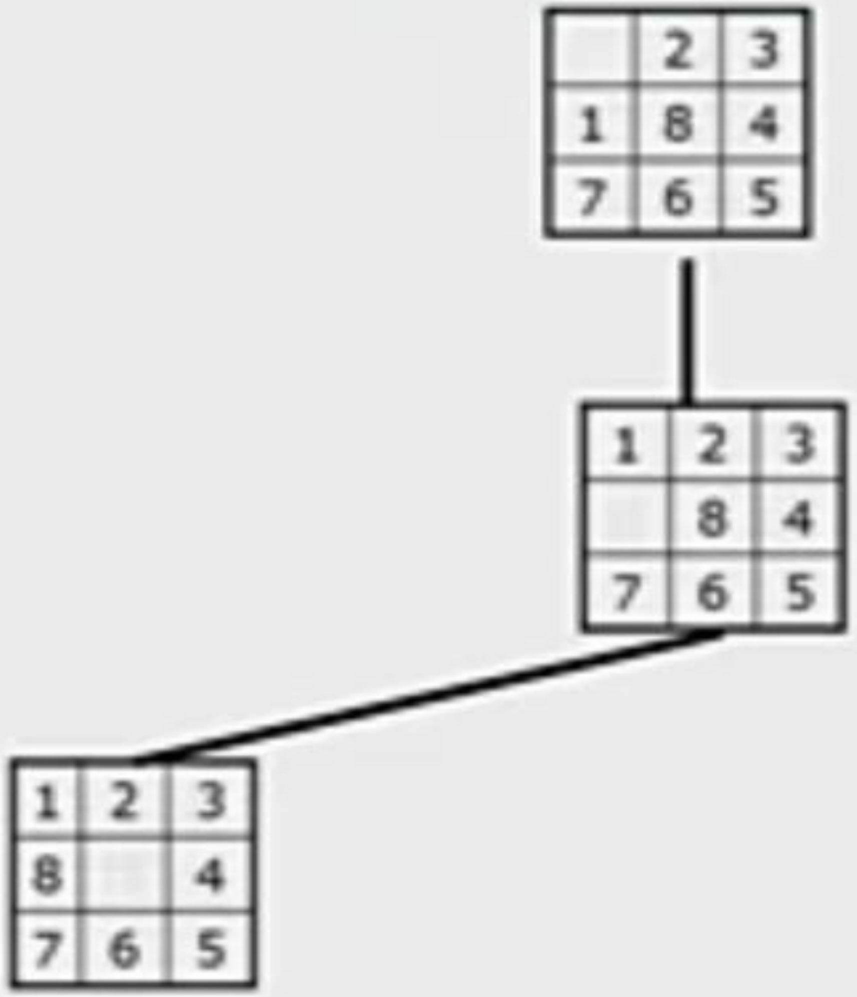
g ; Goal Node

1	2	3
8		4
7	6	5

Playing 8-Puzzle



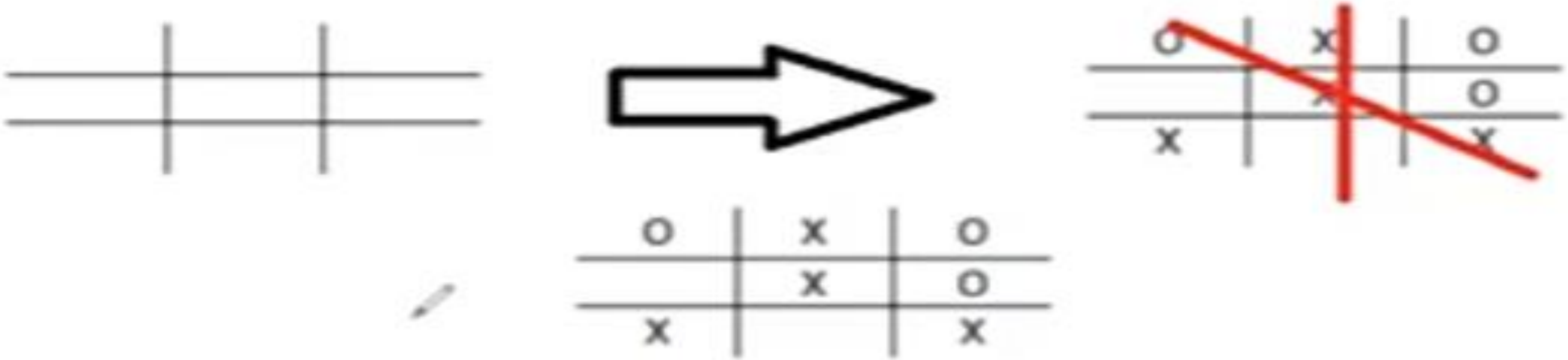
Playing 8-Puzzle



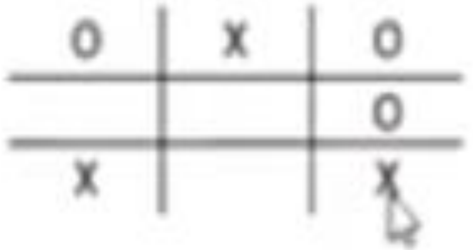
Tic-Tac-Toe Game Playing:

- Algorithm:
- 1. View the vector as a ternary number. Convert it to a decimal number.
- 2. Use the computed number as an index into Move-Table and access the vector stored there.
- 3. Set the new board to that vector

TIC - TAC - TOE



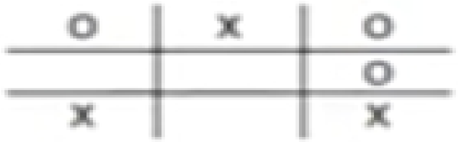
TIC - TAC - TOE



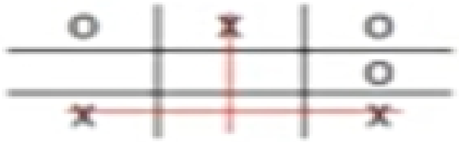
Heuristic Function:

H = x's probability to win – o's probability to win

TIC - TAC - TOE



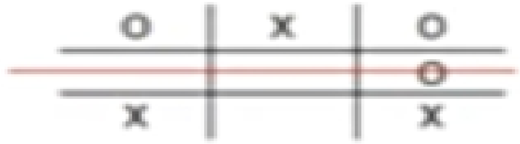
O's turn – 3 places to insert O



Calculate heuristic function

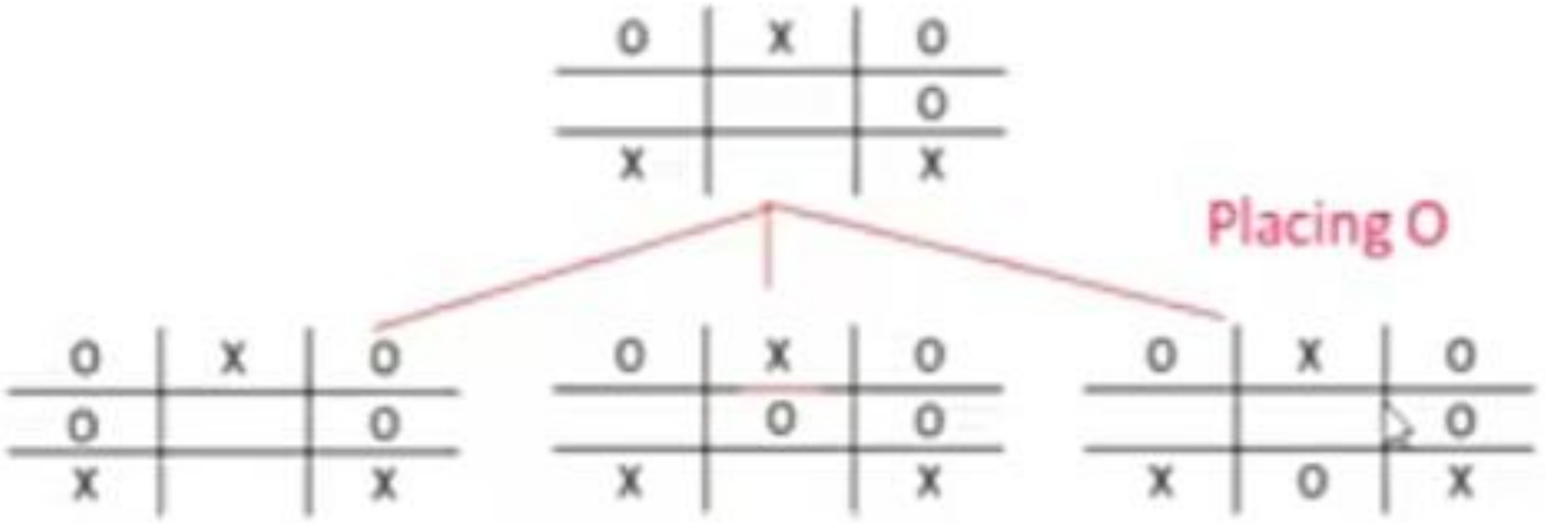
X's possibility to win - 2

O's possibility to win - 1



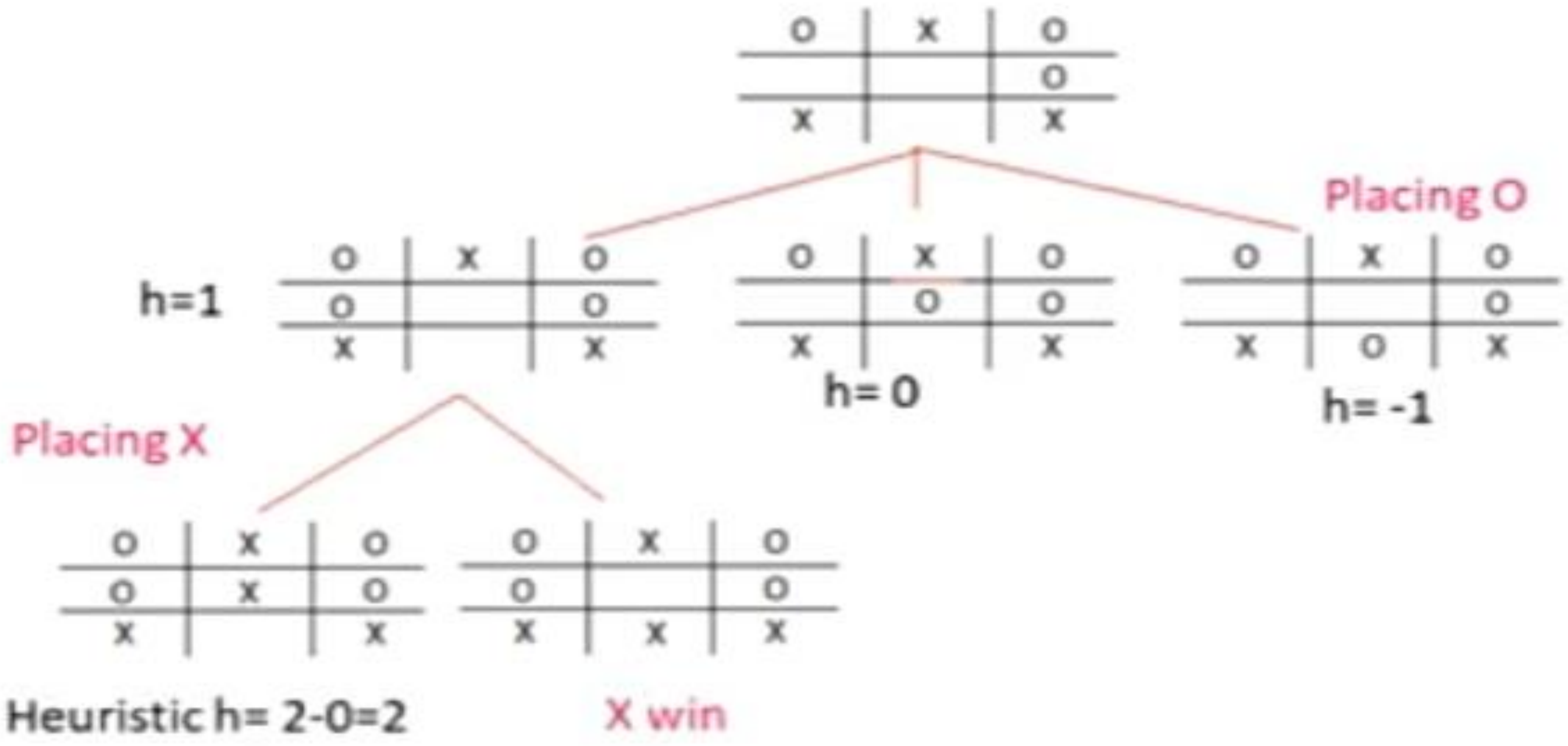
Heuristic Function h = 2-1=1

TIC - TAC - TOE

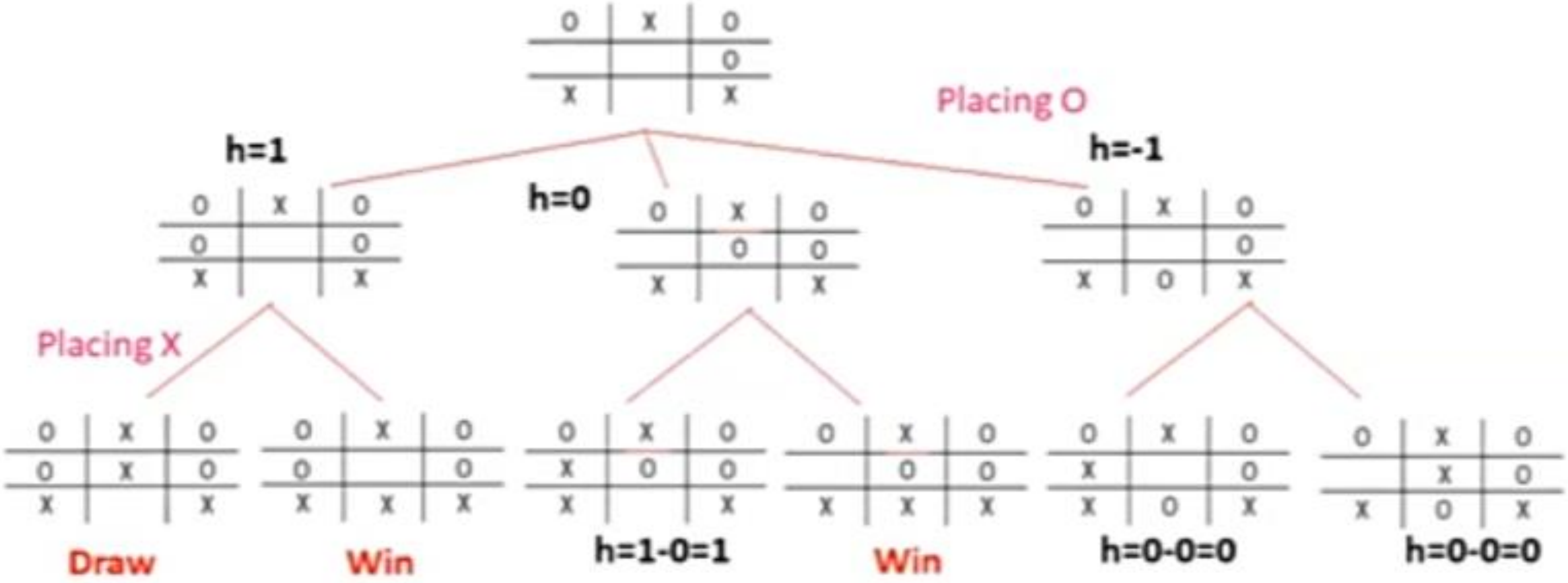


Heuristic h= 2-1=1   Heuristic h= 1-1=0   Heuristic h= 0-1= -1

TIC - TAC - TOE



TIC - TAC - TOE



Procedure:

1) Elements of vector:

    0: Empty

    1: X

    2: O

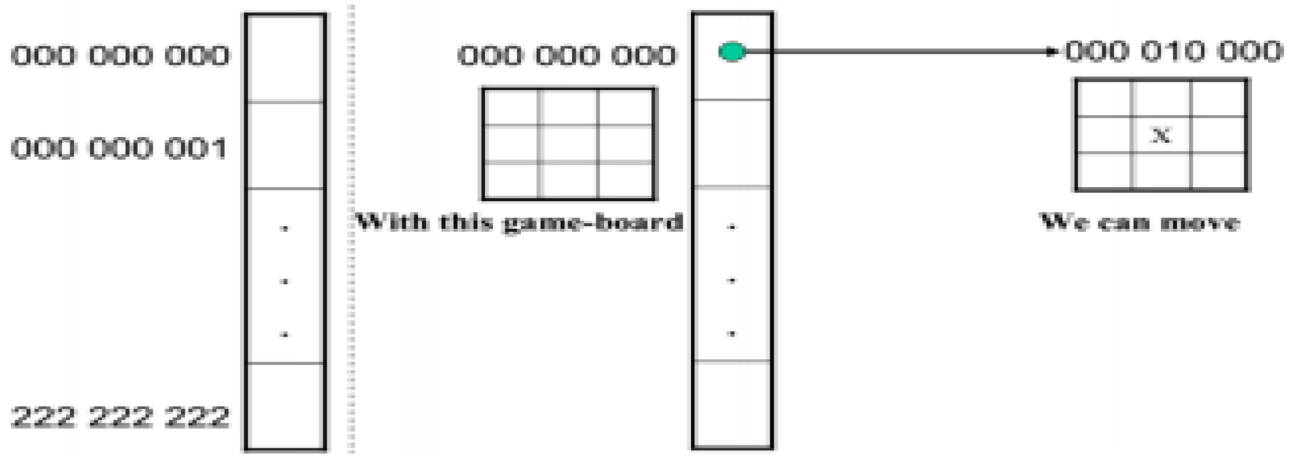
→ the vector is a ternary number

2) Store inside the program a move-table (lookuptable):

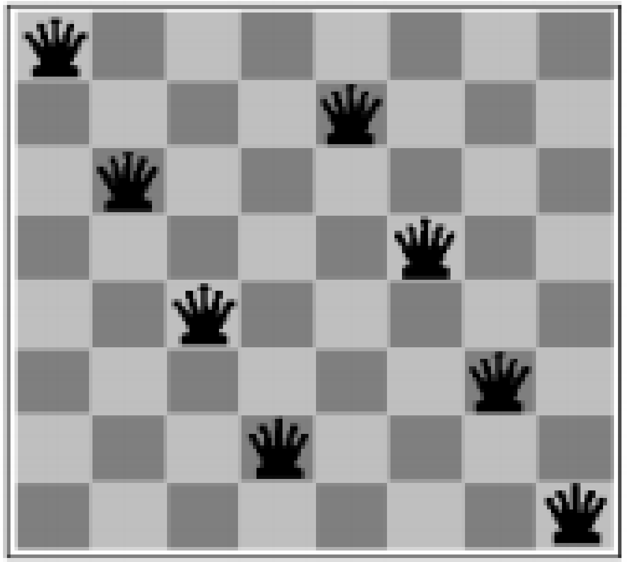
    a) Elements in the table: 19683 (3^9)

    b) Element = A vector which describes the most suitable move from the

❖ Data Structure:



Example: **8-queens problem**



1. **Initial State**: Any arrangement of 0 to 8 queens on board.
2. **Operators**: add a queen to any square.
3. **Goal Test**: 8 queens on board, none attacked.
4. **Path cost**: not applicable or Zero (because only the final state counts, search cost might be of interest).

Water Jug Problem

- The state space for this problem can be described as the set of ordered pairs of integers (x,y) such that x = 0, 1,2, 3 or 4 and y = 0,1,2 or 3; x represents the number of gallons of water in the 4-gallon jug and y represents the quantity of water in 3-gallon jug
- The start state is (0,0)
- The goal state is (2,n)

Production rules for Water Jug Problem

- The operators to be used to solve the problem can be described as follows:

Sl No	Current state	Next State	Descritpion
1	(x,y) if 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 the 3-gallon jug
5	(x,y) if x>0	(0, y)	Empty the 4 gallon jug
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

Production rules

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 gallons from 3-gallon jug into the 4-gallon jug
12	(2,y)	(0,y)	Empty the 2 gallons in the 4-gallon jug on the ground

To solve the water jug problem

- Required a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see if it corresponds to goal state.

- One solution to the water jug problem
- Shortest such sequence will have a impact on the choice of appropriate mechanism to guide the search for solution.

Gallons in the 4-gallon jug	Gallons in the 3-gallon jug	Rule applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5 or 12
0	2	9 0r 11
2	0	

Representation

- State Representation and Initial State
- we will represent a state of the problem as a
- tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug.
- Note $0 \le x \le 4$, and $0 \le y \le 3$. Our initial state: (0,0)

Production rules - Formulation

1. Fill 4-gal jug $(x,y) \rightarrow (4,y)$ where ,$x < 4$

2. Fill 3-gal jug $(x,y) \rightarrow (x,3)$ where, $y < 3$

Production rules - Formulation

3. Empty 4-gal jug on ground $(x,y) \rightarrow (0,y)$ where, $x > 0$

4. Empty 3-gal jug on ground $(x,y) \rightarrow (x,0)$ where, $y > 0$

Production rules - Formulation

5.Empty some water in 3 gallon jug$(x,y) \rightarrow (x,y-d)$ where, $y>0$

6. Empty some water in 4 gallon jug$(x,y) \rightarrow (x-d,y)$ where, $x>0$

Production rules - Formulation

7. Pour water from 3-gal jug $(x,y) \rightarrow (4, y - (4 - x))$ to fill 4-gal jug $x+y \geq 4$ and $y > 0$

8. Pour water from 4-gal jug $(x,y) \rightarrow (x - (3-y), 3)$ to fill 3-gal-jug $x+y \geq 3$ and $x > 0$

Production rules - Formulation

9. Pour all of water from 3-gal jug $(x,y) \rightarrow (x+y, 0)$ into 4-gal jug $0 < x+y \leq 4$ and $y \geq 0$

10. Pour all of water from 4-gal jug $(x,y) \rightarrow (0, x+y)$ into 3-gal jug $0 < x+y \leq 3$ and $x \geq 0$

Breadth First Search:

Let us discuss these strategies using water jug problem. These may be applied to any search problem.

Construct a tree with the initial state as its root.

Generate all the offspring of the root by applying each of the applicable rules to the initial state.

$$\begin{tikzcd}
& (0, 0) \arrow[dl] \arrow[dr] & \\
(4, 0) & & (0, 3)
\end{tikzcd}$$

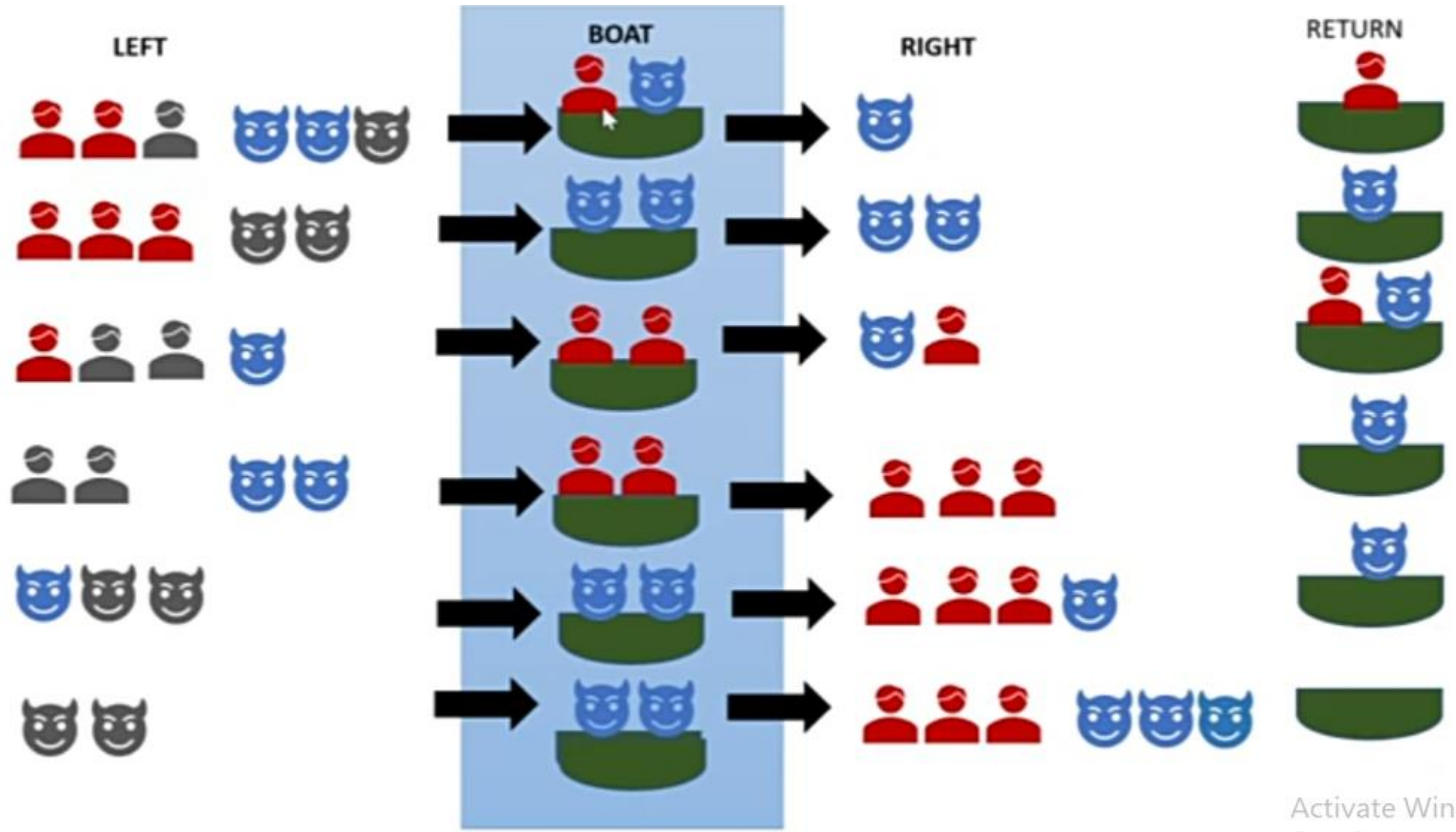
Now for each leaf node, generate all its successors by applying all the rules that are appropriate.

Missionaries and Cannibals Problem

- Three missionaries and three cannibals wish to cross the river.
- They have a small boat that will carry up to two people.
- Everyone can navigate the boat.
- If at any time the Cannibals outnumber the Missionaries on either bank of the river, they will eat the Missionaries.
- **How can everyone get across the river without the missionaries risking being eaten?**

State Representation

- Initial State: 3 missionaries, 3 cannibals and the boat are on Left side.
  L <MMMCCC>
- Goal: Move all the missionaries and cannibals across the river.
  R<MMMCCC>
- Constraint: Missionaries can never be outnumbered by cannibals on either side of river, or else the missionaries are killed.
- Operators: Move boat from left to right
  Move boat from right to left



Activate Windo

Production Rules

Rule	State	Description
1	(0,M)	One missionary sailing the boat from Left to Right
2	(M,0)	One missionary sailing the boat from Right to Left
3	(M,M) (M,M)	Two missionaries sailing from Left to Right Two missionaries sailing from Right to Left
4	(C,C) (C,C)	Two cannibals sailing from Left to Right Two cannibals sailing from Right to Left
5	(0,C)	One cannibal sailing the boat from Left to Right
6	(C,0)	One cannibal sailing the boat from Right to Left
7	(M,C)	One missionary & one cannibal sailing the boat from Left to Right
8	(C,M)	One missionary & one cannibal sailing the boat from Right to Left

Solution

ACTION	LEFT	BOAT	RIGHT
A missionary & cannibal	MMMCCC	MC	C
Two cannibals cross over	MMMCC	CC	CC
Two missionaries cross over	MMMC	MM	CM
Two missionaries cross over	MMCC	MM	MMM
Two cannibals cross over	CCC	CC	MMMC
Two cannibals cross over	CC	CC	MMMCCC

- Uninformed search: Also called blind, exhaustive or brute-force search, uses no information about the problem to guide the search and therefore may not be very efficient.

- Informed Search: Also called heuristic or intelligent search, uses information about the problem to guide the search, usually guesses the distance to a goal state and therefore efficient, but the search may not be always possible.

State Space Search

Definition: A state space is the set of all configurations that a given problem and its environment could achieve.

- All the states the system can be in are represented as *nodes* of a graph.
- An action that can change the system from one state to another is represented by a *link* from one node to another.
- Links may be *unidirectional* or *bi-directional* .
- Search for a solution.
- A solution might be:
  - *Any* path from start state to goal state.
  - The *best* (e.g. lowest cost) path from start state to goal state.
- It may be possible to reach the same state through many *different* paths.
- There may be *loops* in the graph (can go round in circle).

Search Algorithm Terminology

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

> **Search Space:** Search space represents a set of possible solutions, which a system may have.
>
> **Start State:** It is a state from where agent begins **the search**.
>
> **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.

Properties of Search Algorithms

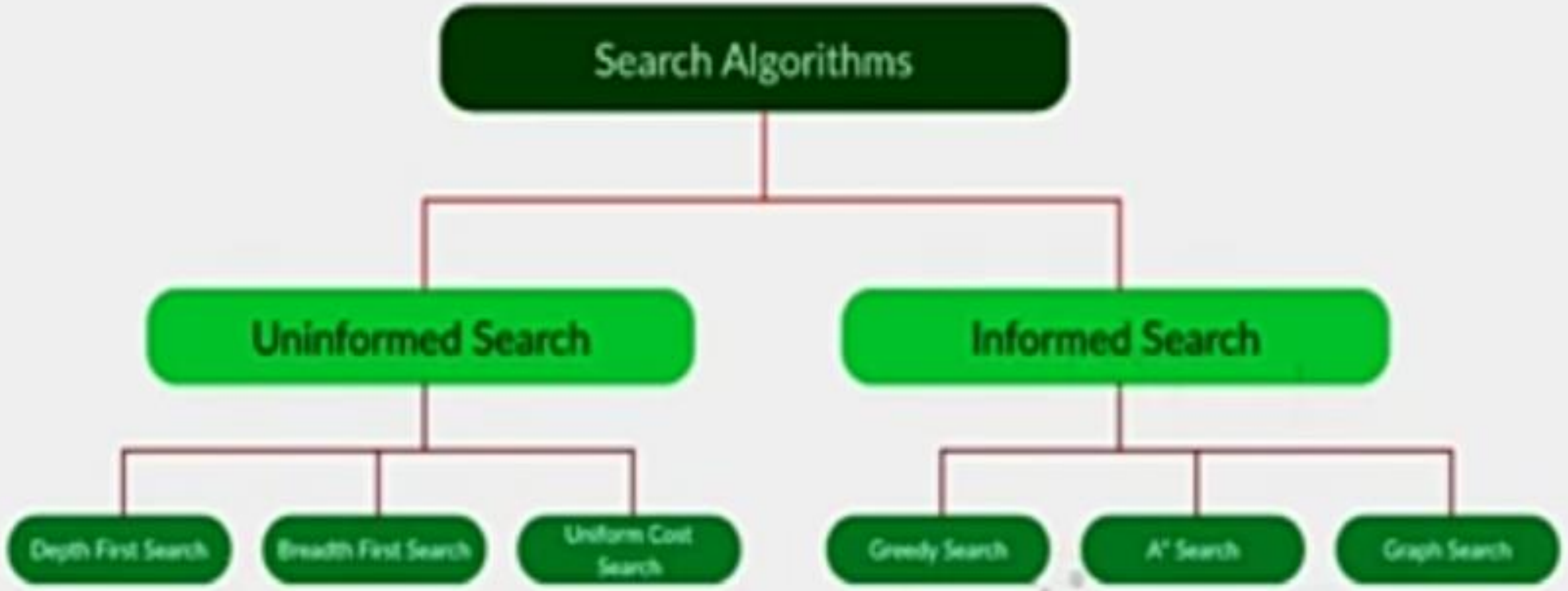
Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

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.

Types of Search Algorithms



Informed vs. Uninformed Search

INFORMED SEARCH	UNINFORMED SEARCH
It uses knowledge for the searching process.	It doesn't use knowledge for searching process.
It finds solution more quickly.	It finds solution slow as compared to informed search.
It is highly efficient.	It is mandatory efficient.
Cost is low.	Cost is high.
It consumes less time.	It consumes moderate time.
It provides the direction regarding the solution.	No suggestion is given regarding the solution in it.
It is less lengthy while implementation.	It is more lengthy while implementation.
Greedy Search, A* Search, Graph Search	Depth First Search, Breadth First Search

Activate Win

Uninformed Search

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

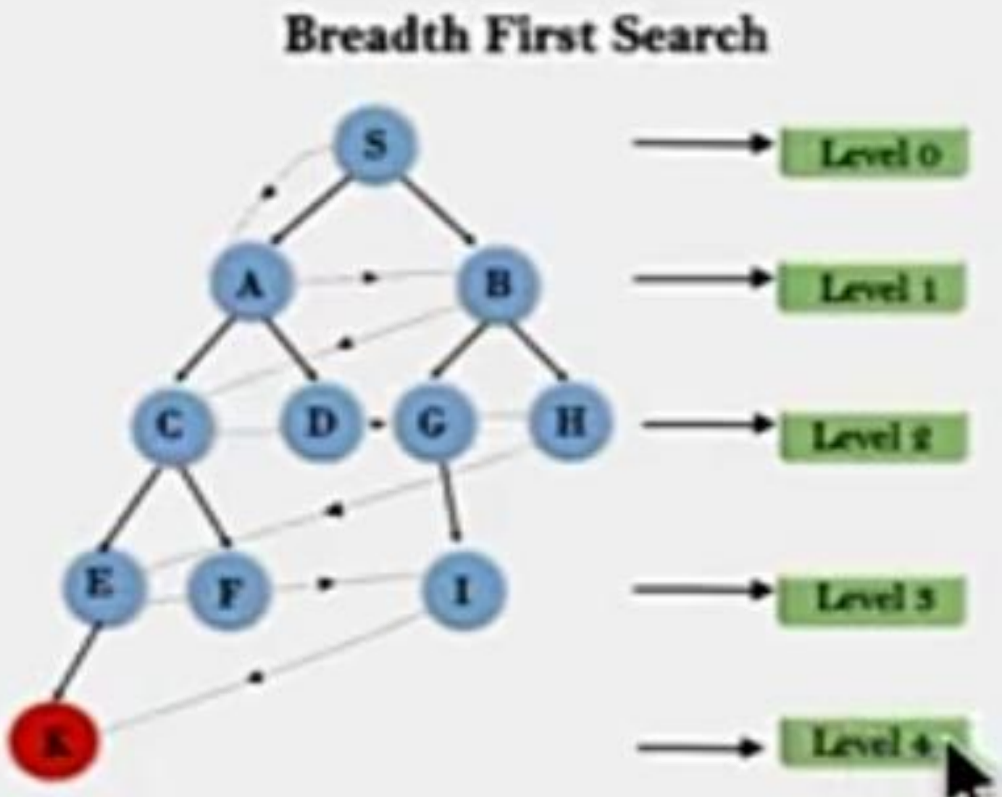
- Breadth-first Search
- Depth-first Search
- Uniform cost search

Breadth First Search

- This algorithm searches breadth wise in a tree or graph, so it is called breadth- first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- Breadth-first search implemented using FIFO queue data structure.

Steps:

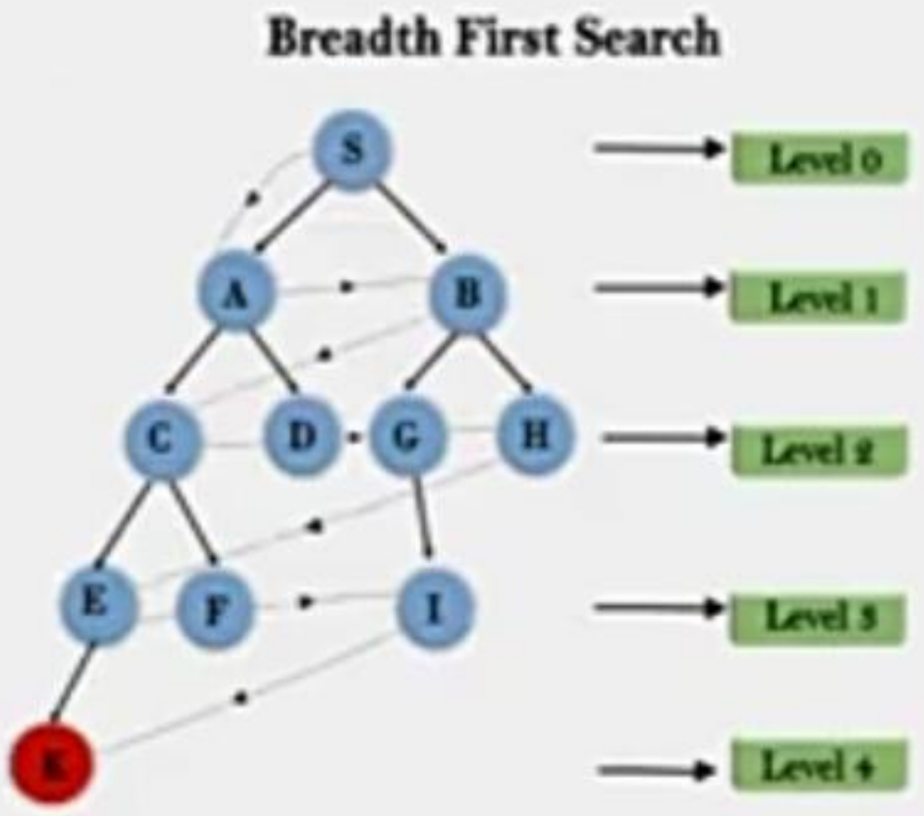
1: Traverse the root node.
2: Traverse all neighbours of root node.
3: Traverse all neighbours of neighbours of the root node.
4: This process will continue until we are getting the goal node.



Breadth First Search

Algorithm

- Enqueue the root node
- Dequeue a node and examine it
  - If the element sought is found in this node, quit the search and return a result.
  - Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
- If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
- If the queue is not empty, repeat from Step 2.



Breadth First Search

Properties of breadth-first search

b: branching factor (avg. no of nodes per child)
d: depth of shallowest goal node

Total number of nodes generated is:
$1+b+b^2+b^3+\ldots \ldots + b^d = O(b^d)$

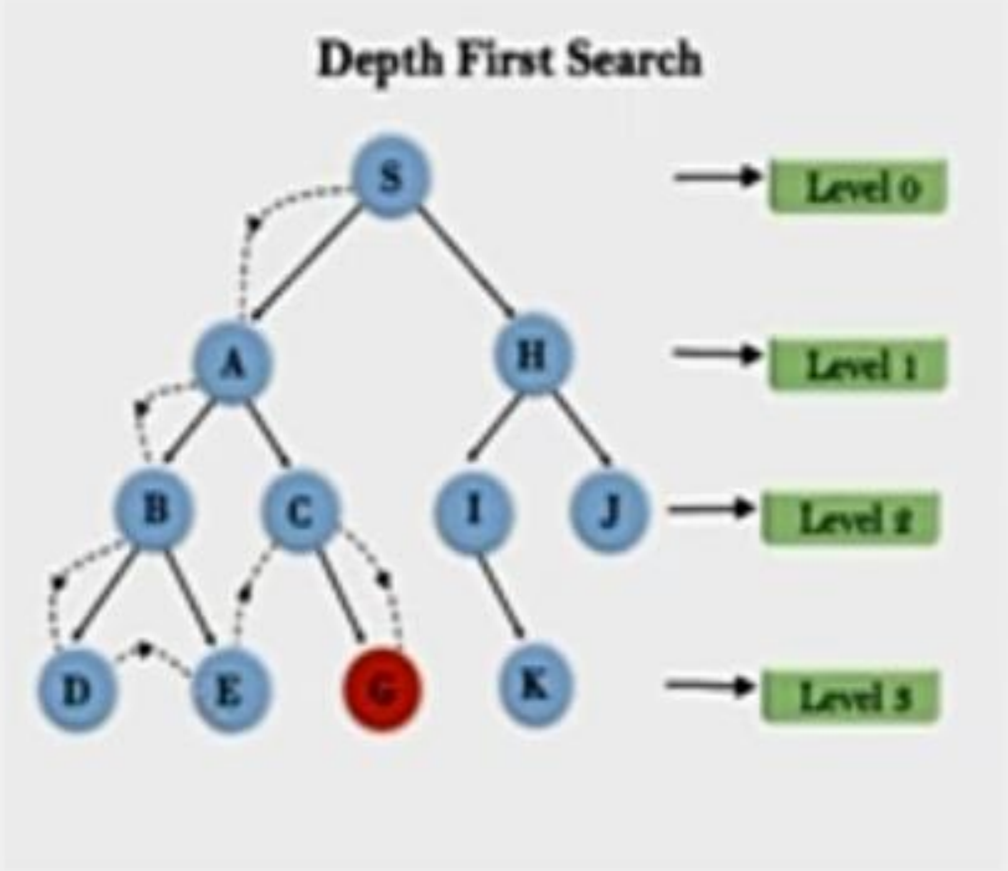
- Complete Yes (if b is finite)
- Time $1+b+b^2+b^3+\ldots \ldots + b^d = O(b^d)$
- Space $O(b^d)$ (keeps every node in memory)
- Optimal if all operators have the same cost.

Depth First Search

- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path.

- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors . As those nodes are expanded , they are dropped from the Frontier [stack] ,so then the search "backs up" to the deepest next node that still has unexplored successors .

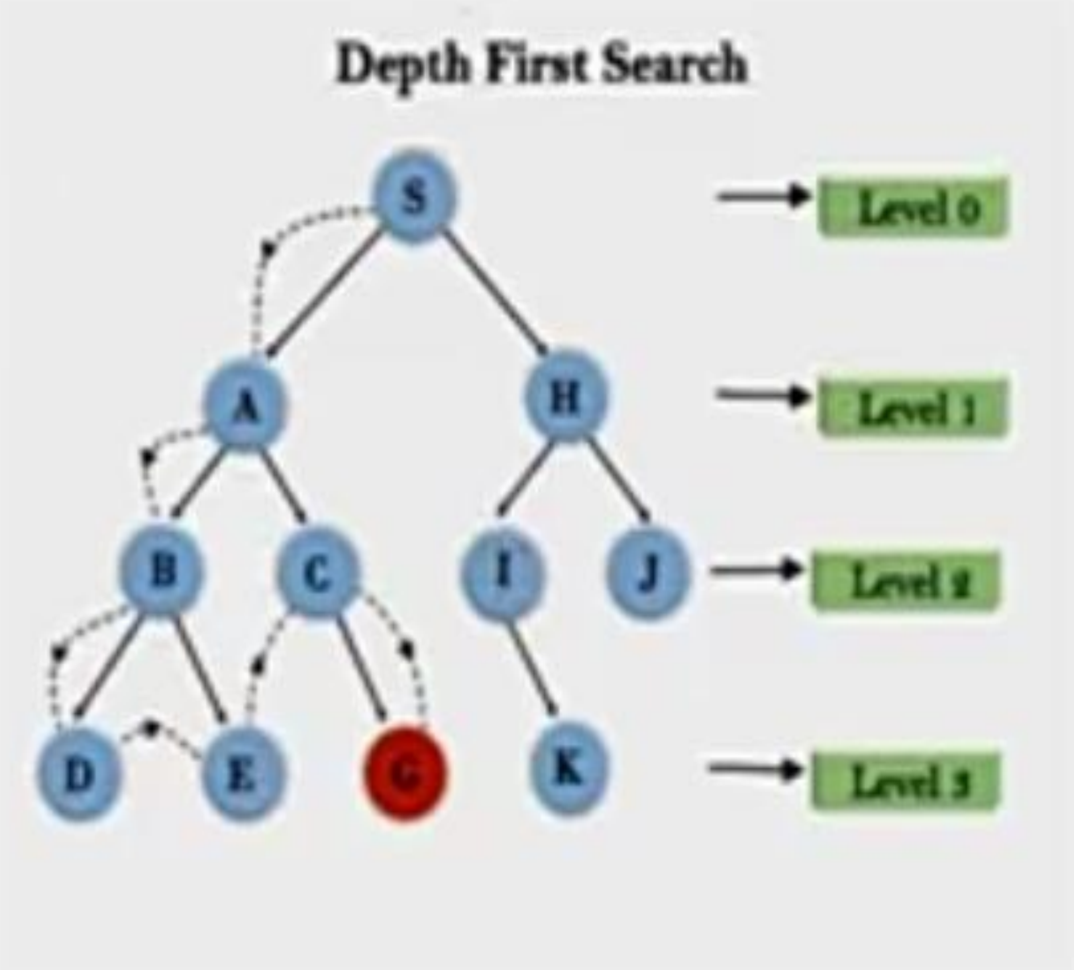
- Use Stack data structure



Depth First Search

Algorithm

```
Put the root node on a stack;
while (stack is not empty) {
    remove a node from the stack;
    if (node is a goal node) return success;
    put all children of node onto the stack;
}
return failure;
```



Properties of Depth First Algorithm

b(branching factor) : Maximum number of successors of any node.
m: maximal depth of a leaf node

- Number of nodes generated (worst case): $1+b+b^2+b^3+\ldots\ \ldots+b^d = O(b^d)$

- Complete: only for finite search tree
- Optimal : Not optimal
- Time complexity: $O(b^d)$
- Space complexity : O(bm) [or O(m)]

- **CHARACTERISTICS OF PRODUCTION SYSTEMS**
- Production systems provide us with good ways of describing the operations that can be
- performed in a search for a solution to a problem.
- At this time, two questions may arise:
- 1. Can production systems be described by a set of characteristics? And how can they be easily implemented?
- 2. What relationships are there between the problem types and the types of production systems well suited for solving the problems?

- To answer these questions, first consider the following definitions of classes of production systems:

- 1. A monotonic production system is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
- 2. A non-monotonic production system is one in which this is not true.
- 3. A partially communicative production system is a production system with the property that if the application of a particular sequence of rules transforms state P into state Q, then any combination of those rules that is allowable also transforms state P into state Q.
- 4. A commutative production system is a production system that is both monotonic and partially commutative.

Table 2.1 Four Categories of Production Systems

Production System	Monotonic	Non-monotonic
Partially Commutative	Theorem Proving	Robot Navigation
Non-partially Commutative	Chemical Synthesis	Bridge

Is there any relationship between classes of production systems and classes of problems? For any solvable problems, there exist an infinite number of production systems that show how to find solutions. Any problem that can be solved by any production system can be solved by a commutative one, but the commutative one is practically useless. It may use individual states to represent entire sequences of applications of rules of a simpler, non-commutative system. In the formal sense, there is no relationship between kinds of problems and kinds of production systems Since all problems can be solved by all kinds of systems. But in the practical sense, there is definitely such a relationship between the kinds of problems and the kinds of systems that lend themselves to describing those problems.

Is there any relationship between classes of production systems and classes of problems? For any solvable problems, there exist an infinite number of production systems that show how to find solutions. Any problem that can be solved by any production system can be solved by a commutative one, but the commutative one is practically useless. It may use individual states to represent entire sequences of applications of rules of a simpler, non-commutative system. In the formal sense, there is no relationship between kinds of problems and kinds of production systems Since all problems can be solved by all kinds of systems. But in the practical sense, there is definitely such a relationship between the kinds of problems and the kinds of systems that lend themselves to describing those problems.

Partially commutative, monotonic productions systems are useful for solving ignorable problems. These are important from an implementation point of view without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed. Both types of partially commutative production systems are significant from an implementation point; they tend to lead to many duplications of individual states during the search process. Production systems that are not partially commutative are useful for many problems in which permanent changes occur.

Production system

Inorder to solve a problem:

- We must first reduce it to one for which a precise statement can be given. This can be done by defining the problem's state space (start and goal states) and a set of operators for moving that space.
- The problem can then be solved by searching for a path through the space from an initial state to a goal state.
- The process of solving the problem can usefully be modelled as a production system.

Control Strategies

- How to decide which rule to apply next during the process of searching for a solution to a problem?
- The two requirements of good control strategy are that
  - it should cause motion.
  - It should be systematic

Uninformed Search Methods:

- Breadth- First -Search:

Consider the state space of a problem that takes the form of a tree. Now, if we search the goal along each breadth of the tree, starting from the root and continuing up to the largest depth, we call it breadth first search.

Breadth First Search

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

BFS illustrated:

Step 1: Initially fringe contains only one node corresponding to the source state A.

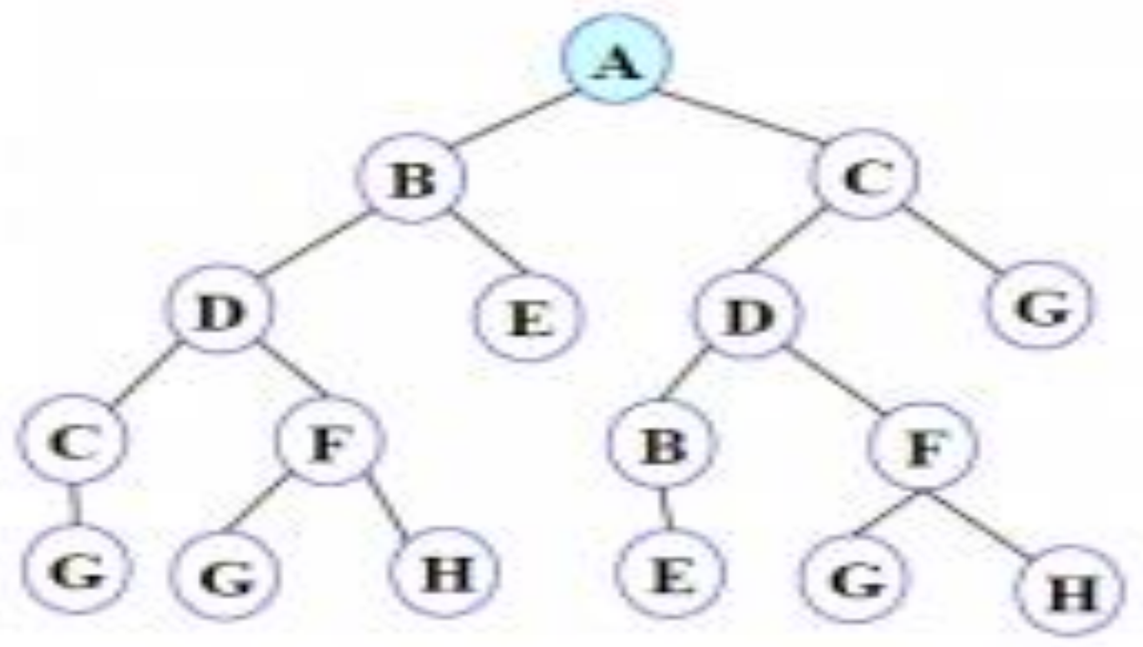


Figure 1

FRINGE: A

Step 2: A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.

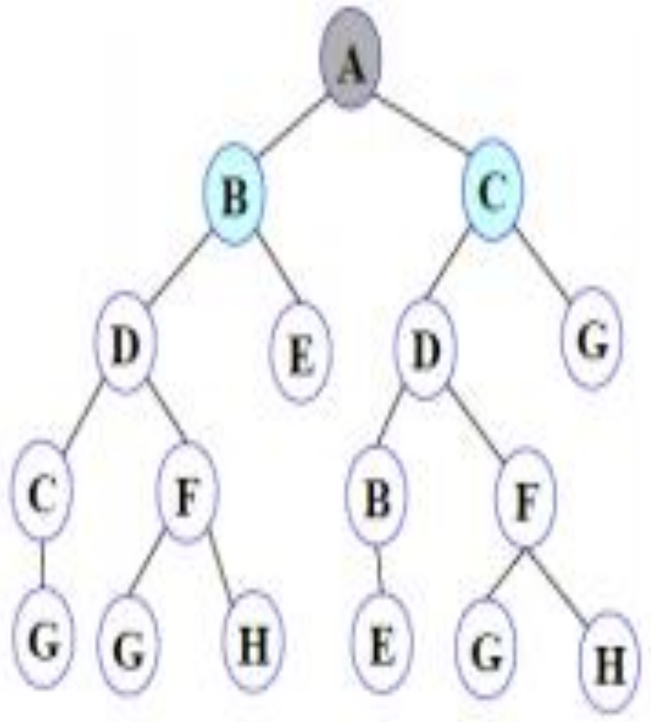


Figure 2

FRINGE: B C

Step 3: Node B is removed from fringe and is expanded. Its children D, E are generated and put at the back of fringe.

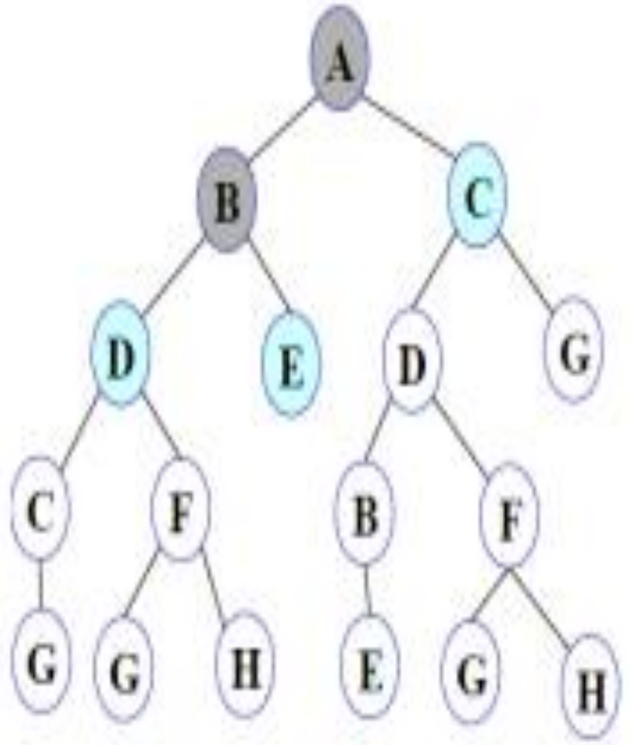


Figure 3

FRINGE: C D E

Step 4: Node C is removed from fringe and is expanded. Its children D and G are added to the back of fringe.

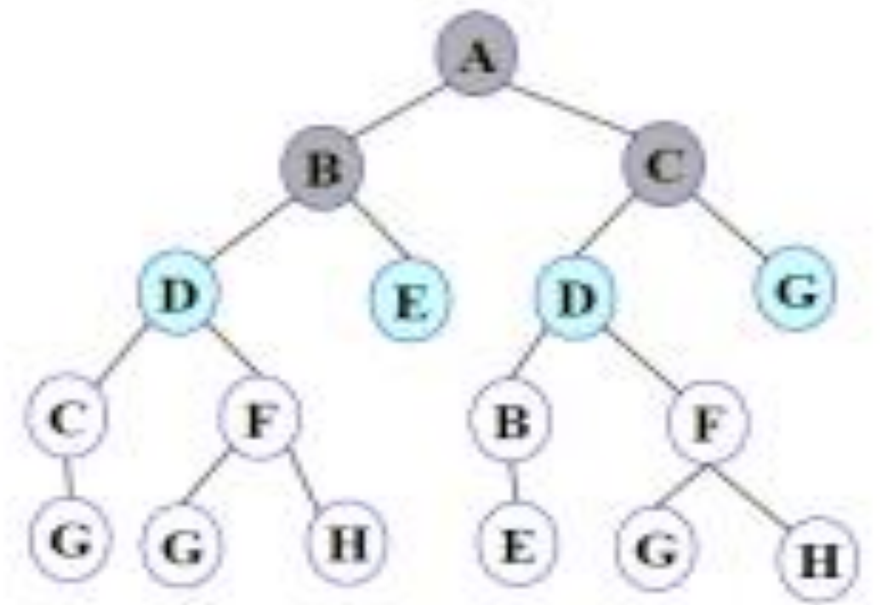


Figure 4

FRINGE: D E D G

Step 5: Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.

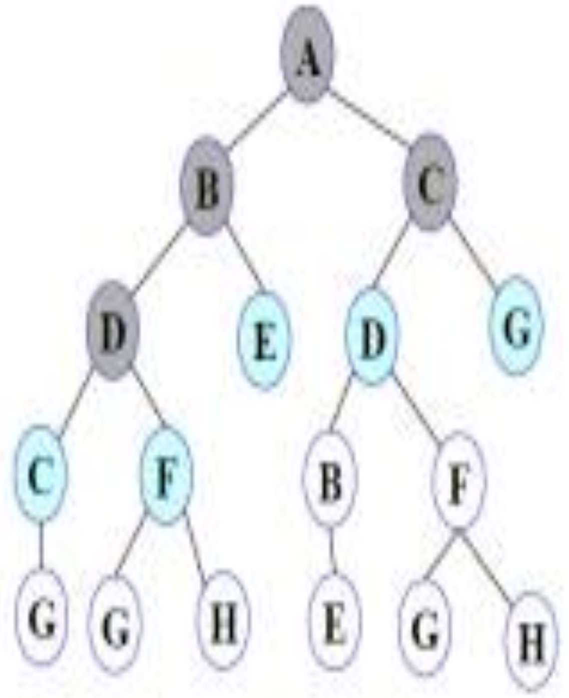


Figure 5

FRINGE: E D G C F

Step 6: Node E is removed from fringe. It has no children.

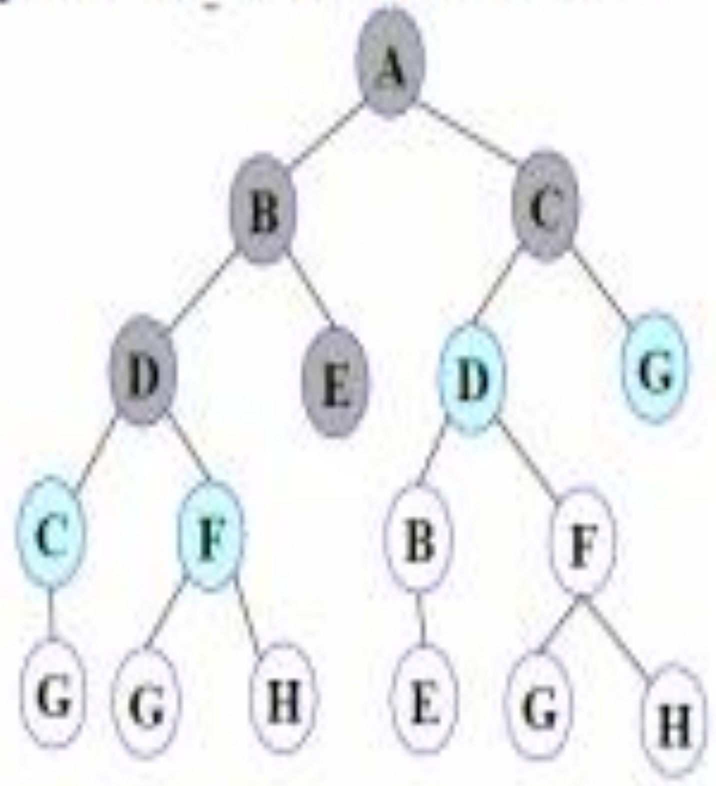


Figure 6

FRINGE: D G C F

Step 7: D is expanded; B and F are put in OPEN.

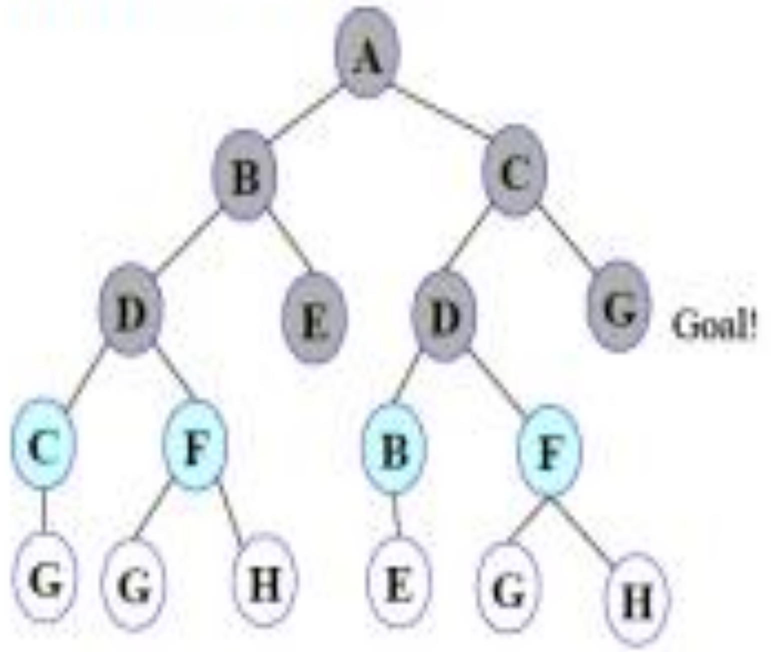


Figure 7

FRINGE: G C F B F

Step 8: G is selected for expansion. It is found to be a goal node. So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.

Breadth first search is:

- One of the simplest search strategies
- Complete. If there is a solution, BFS is guaranteed to find it.
- If there are multiple solutions, then a minimal solution will be found
- The algorithm is optimal (i.e., admissible) if all operators have the same cost. Otherwise, breadth first search finds a solution with the shortest path length.
- **Time complexity** : $O(b^d)$
- **Space complexity** : $O(b^d)$
- **Optimality** :Yes

b - branching factor(maximum no of successors of any node),

d – Depth of the shallowest goal node

Maximum length of any path (m) in search space

Advantages of BFS

- BFS will not get trapped exploring a blind alley.
- If there is a solution, BFS is guarnateed to find it.
- If there are multiple solutions, then a minimal solution will be found.

BFS Tree for Water Jug problem

$$\begin{tikzcd}
 & & (0,0) \arrow[dl] \arrow[drr] & & & \\
 & (4,0) \arrow[dl] \arrow[d] \arrow[dr] & & & (0,3) \arrow[dl] \arrow[d] \arrow[dr] & \\
(4,3) & (0,0) & (1,3) & (4,3) & (0,0) & (3,0)
\end{tikzcd}$$

Depth- First- Search.

We may sometimes search the goal along the largest depth of the tree, and move up only when further traversal along the depth is not possible. We then attempt to find alternative offspring of the parent of the node (state) last visited. If we visit the nodes of a tree using the above principles to search the goal, the traversal made is called depth first traversal and consequently the search strategy is called *depth first search*.

Backtracking

- In this search, we pursue a singal branch of the tree until it yields a solution or until a decision to terminate the path is made.
- It makes sense to terminate a path if it reaches dead-end, produces a previous state. In such a state backtracking occurs
- Chronological Backtracking: Order in which steps are undone depends only on the temporal sequence in which steps were initially made.
- Specifically most recent step is always the first to be undone.
- This is also simple backtracking.

Algorithm: Depth First Search

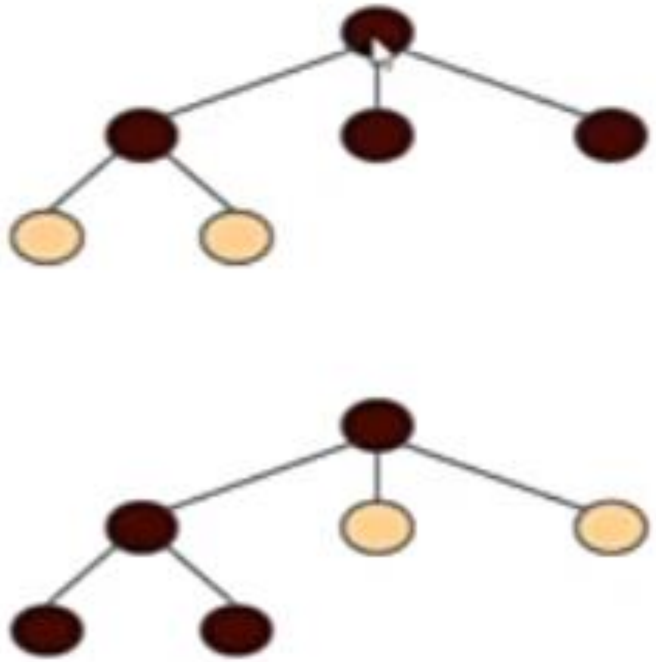
1. If the initial state is a goal state, quit and return success
2. Otherwise, do the following until success or failure is signaled:
   a. Generate a successor, E, of initial state. If there are no more successors, signal failure.
   b. Call Depth-First Search, with E as the initial state
   c. If success is returned, signal success. Otherwise continue in this loop.

Properties of DFS algorithm

- **Is Complete**: DFS is non complete, DFS search algorithm is complete within finite state space
- **Is 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.
- **Time complexity**: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by: $T(n)= 1+ n^2+ n^3 +.........+ n^m= O(n^m)$
  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(bm)**.

Search Strategies: Blind Search

Criterion	Breadth-First	Depth-First
Time	b^d	b^m
Space	b^d	bm
Optimal?	Yes	No
Complete?	Yes	No



- b: branching factor d: solution depth m: maximum depth

Advantages:

- DFS consumes very less memory space.
- It will reach at the goal node in a less time period than BFS if it traverses in a right path.
- It may find a solution without examining much of search because we may get the desired solution

Disadvantages:

- It is possible that may states keep reoccurring.
- There is no guarantee of finding the goal node.
- Sometimes the states may also enter into infinite loops.