ARTIFICIAL INTELLIGENCE
UNIT-II

Dr. R. Seeta Sireesha

Associate Professor

CSE Department

GVPCE(A)

UNIT - 2

PROBLEM SOLVING AND SEARCHING:

Introduction to Problem Solving, Problem Formulation, State Space Representation, Problem Formulation of real-world problems, Production System, Problem Characteristics, Solving problems by searching.

UNINFORMED & INFORMED SEARCH STRATEGIES:

Breadth-First Search, Depth First Search, Uniform Cost Search, Depth-Limited Search, Iterative Deepening Search, Bidirectional Search, Comparing Uniform Search Strategies, Hill Climbing, Best First Search, A* Search, AO* Search. (TextBook-1)

Introduction

- In Artificial Intelligence (AI), the term *problem solving* is given to the analysis of how computers can be made to find solutions in well-circumscribed domains.
- Since puzzles and games have explicit rules, these are often used to explore ideas in AI
- A good example of a problem-solving domain is the “Tower of Hanoi” puzzle
- *Problem solving* is defined as the way in which an agent finds a sequence of actions that achieves its goals, when no single action will do.
- *Problem formulation* usually requires abstracting away real-world details to define a state space that can feasibly be explored. This step of abstraction is performed by an agent called as a *problem-solving agent*.

Types of Problems

There are three types of problem in artificial intelligence:

- Ignorable: In which solution steps can be ignored.
- Recoverable: In which solution steps can be undone.
- Irrecoverable: Solution steps cannot be undo.

Steps in problem-solving in AI

- **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.

Structure of problem solving agent

1. **Goal state**: A state that describes the objective that the agent is trying to achieve is called as the goal state of the agent.
2. **Action**: When there is a transition between the world states an action is said to be performed

Steps in problem solving by problem solving agent

1. **Goal formulation**: On the basis of the current situation and agent's performance measure, it is the first step in problem solving. Agent's task is to find out sequence of actions that will get it to the goal state. Goals help organize behavior by limiting the objectives the agent is trying to achieve.
2. **Problem formulation**: It is a process of deciding what actions and states to consider given a goal.

3. Choosing the best sequence: An agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that lead to the states of known value and then choosing the best sequence.

- The process of finding such sequence is called search, a search algorithm is like a black box which takes problem as input returns a **solution,** and once solution is found the sequence actions are carried out, and this is called the execution phase.

```
Function SIMPLE-PROBLEM-SOLVING-AGENT (Perfect) returns
Action
Static: Seq, an action sequence, initially empty
        State, the description of current world state
        Goal, a goal, initially null
        Problem, a problem formulation
State ← UPDATE-STATE (State, Percept)
If seq is empty then do
Goal ← FORMULATE-GOAL (State)
Problem ← FORMULATE-PROBLEM (State, Goal)
Seq ← SEARCH (Problem)
Action ← FIRST (Seq)
Seq ← REST (Seq)
return action.
```

Problem Formulation

Problem formulation is the process of deciding what actions and states to consider given a goal.

ALGORITHM

Step 1: Analyze the problem to get the starting state and goal state.

Step 2: Find out the data about the starting state, goal state.

Step 3: Find out the production rules from initial database for progressing the problem to goal state.

Step 4: Select some rules from the set of rules that can be applied to data.

Step 5: Apply those rules to the initial state and proceed to get the next state.

Step 6: Determine some new generated states after applying the rules. Accordingly, make them as current state.

Step 7: Finally, achieve some information about the goal state from the recently used current state and get the goal state.

Step 8: Exit.

Components to formulate the problem

- **Initial State**: This state requires an initial state for the problem which starts the AI agent towards a specified goal.
- **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 determine 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 Representation

- A state–space representation allows for the formal definition of a problem, which makes the movement from initial state to the goal state quite easily.
- So, we can say that various problems like planning, learning, theorem proving, etc., are all essentially search problems only.
- *State–space search* is a process used in the field of computer science, including AI, in which successive configurations or states of an instance are considered, with the goal of finding a *goal state* with a desired property.

  SSR: {S, A, Action(S),Result(S, a), cost(S, a)}

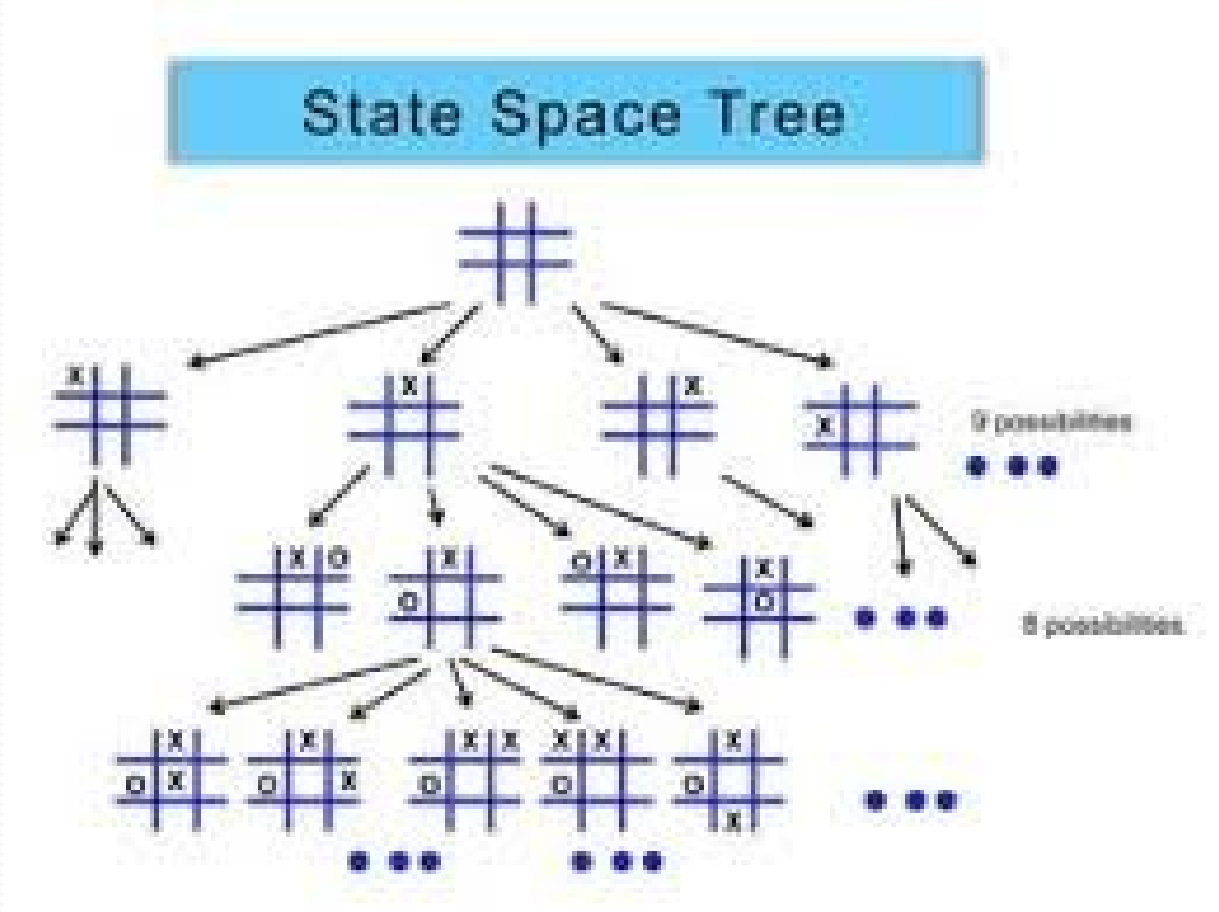
  S-> total number of states {start, intermediate, goal}

  A-> all possible actions

Tic-Tac Toe State Problem Formulation

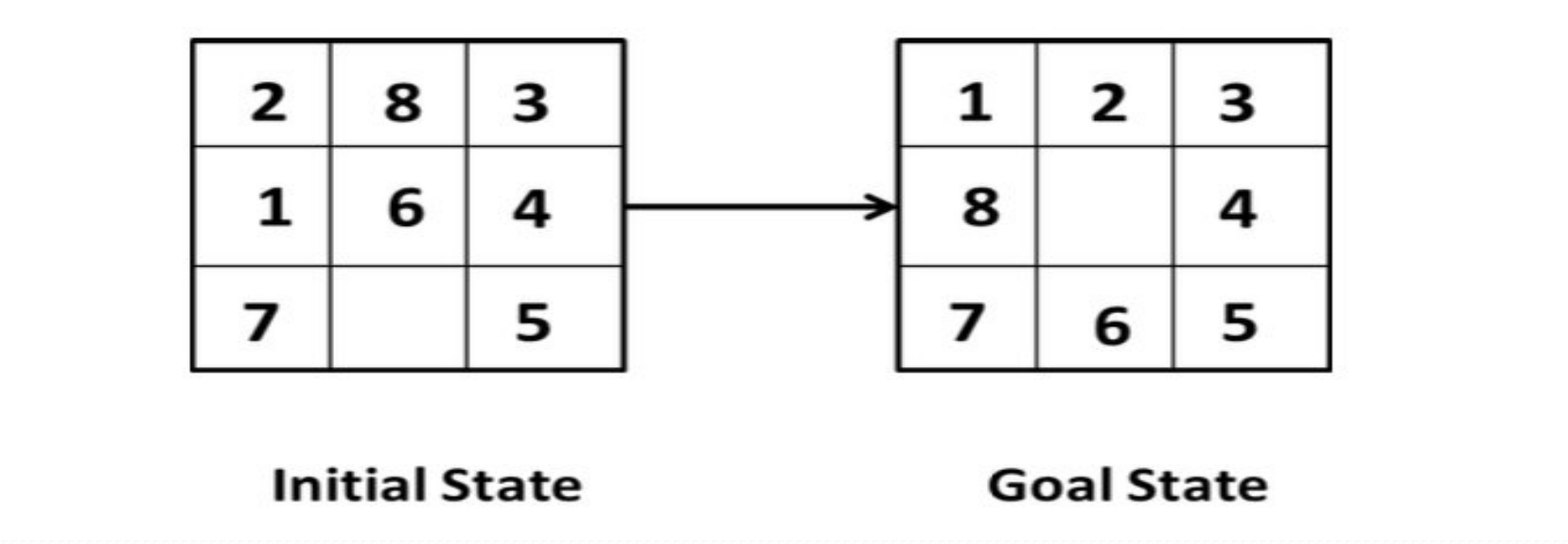
- The start state is an empty board, and the termination or goal description is a board state having three Xs in a row, column, or diagonal (assuming that the goal is a win for X).

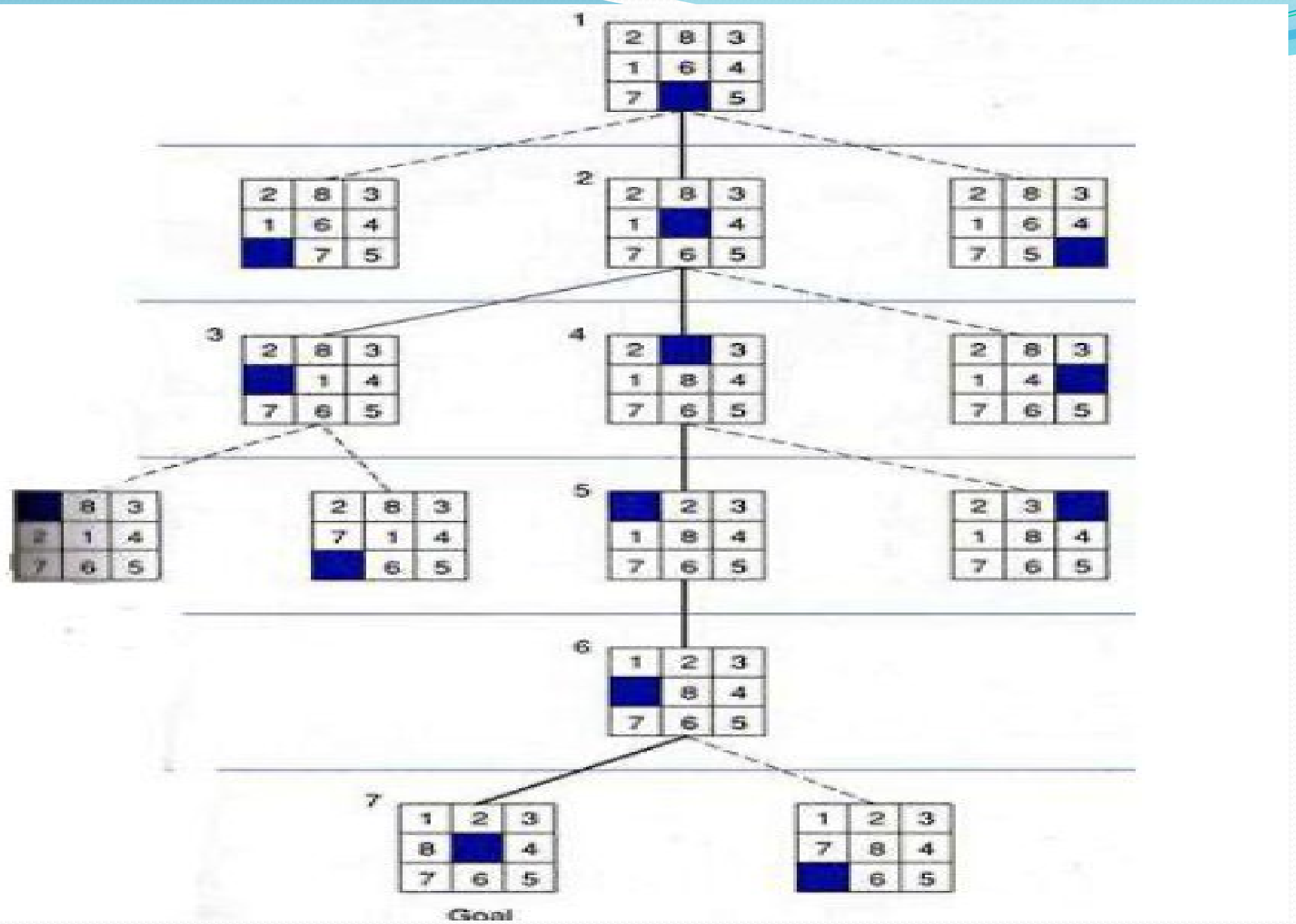
Tic-Tac Toe State space representation



State Space Representation of the Eight Tile Puzzle

The eight tile puzzle consist of a 3 by 3 (3 × 3) square frame board which holds eight movable tiles numbered 1 to 8. One square is empty, allowing the adjacent tiles to be shifted. The objective of the puzzle is to find a sequence of tile movements that leads from a starting configuration to a goal configuration.



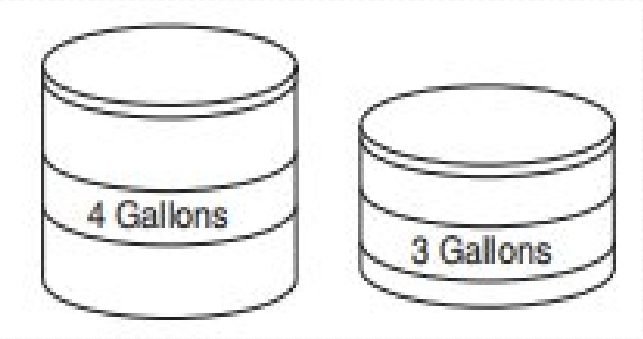

1
2
3
4
5
6
7
Goal

Problem formulation of the Eight Tile Puzzle

- **States:** It specifies the location of each of the eight tiles and the blank in one of the nice squares.
- **Initial state:** Any state can be designated as the initial state.
- **Goal:** Many goal configurations are possible.
- **Legal moves (or state):** They generate legal states that result from trying the four actions:
  - Blank moves left
  - Blank moves right
  - Blank moves up
  - Blank moves down
- **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

Problem formulation of the Water Jug Problem

Problem definition: You have given two jugs one with capacity of 4 gallons and other with capacity of 3 gallons none is having any measuring marks on it. There is the pump that can be used to fill water in jug.



Problem formulation Steps:

- **1. Describe the states**
- **2. Identify the initial state**
- **3. Identify the goal state**
- **4. Identify set of rules (all possible actions)**
- **5. Find the solution path in the state space**

Problem formulation of the Water Jug Problem

- **Solution: Now, we need to solve the problem such that there is two gallons of water in four gallons of jug.** We will have to keep track of the amount of water in each of the jugs after we perform a particular action. Hence, we should represent the state such that it reflects the amount of water in each of the individual jugs.
- **Representation of state: For the aforementioned problem, the state can be described as ordered pair of** integer (*x, y) such that 'x' is the amount of water in four gallon of jug, hence 'x' can take values 0, 1, 2, 3, 4* gallons. (**Note: Since *'x' is the amount of water in four gallon of jug, the jug can be empty, i.e. has 0 gallons*** of water or can have 1/2/3/4 gallons of water, respectively). *'y' is the amount of water in three gallons of jug.* Similarly *y can take values as 0, 1, 2, 3 ...*

Problem formulation of the Water Jug Problem

- **Initial state: Initially, we assume that both the jugs are empty. That is both the jugs contain zero gallons of** water. This initial situation (also called as initial state) can be represented using state representation notation as follows:
- SI = (0, 0) Note that initial state is represented by SI.
- **Goal state: We need two gallons of water in four gallons of jug as the goal but there can be any amount of** water in three gallons of jug. Hence goal state is the set of states.
- SG = {(2, 0), (2, 1), (2, 2), (2, 3)}
- This notation indicates that there must be two gallons of water in four gallons of jug but there can be 0/1/2/3 gallons of water in three gallons of jug which we are not concerned. Hence, goal state represented by SG is set of states as shown. Hence, as the goal the agent can achieve any one of the states that belongs to this set.

Problem formulation of the Water Jug Problem

- **Set of rules: Now, we write set of rules that will be applied to current state so as to change the current state** and go to the next state. To write these set of rules one has to simply think of **all possible combinations** that can occur,

Rule 1: Fill in 4G Jug completely

$(x, y) \rightarrow (4, y)$ if $(x < 4)$

Rule 2: Fill in 3G Jug completely

$(x, y) \rightarrow (x, e)$ if $(y < 3)$

Rule 3: Pour some water from 4G Jug to ground

$(x, y) \rightarrow (x-d, y)$ if $(x > 0)$

Rule 4: Pour some water from 3G jug to ground

$(x, y) \rightarrow (x, y-d)$ if $(y > 0)$

Rule 5: Empty 4G jug

$(x, y) \rightarrow (0, y)$ if $(x > 0)$

Problem formulation of the Water Jug Problem

Rule 6: Empty 3G jug

$(x, y) \rightarrow (x, 0)$ if $(y > 0)$

Rule 7: Pour some water from 3G jug to 4G jug until 4G jug is FULL

$(x, y) \rightarrow (4, y - (4 - x))$ if $(y > 0)$ and $(x + y) \geq 4$

Rule 8: Pour some water from 4G jug to 3G jug until 3G jug is FULL

$(x, y) \rightarrow (x - (3 - y), 3)$ if $(x > 0)$ and $(x + y) \geq 3$

Rule 9: Pour all water from 3G jug to 4G jug

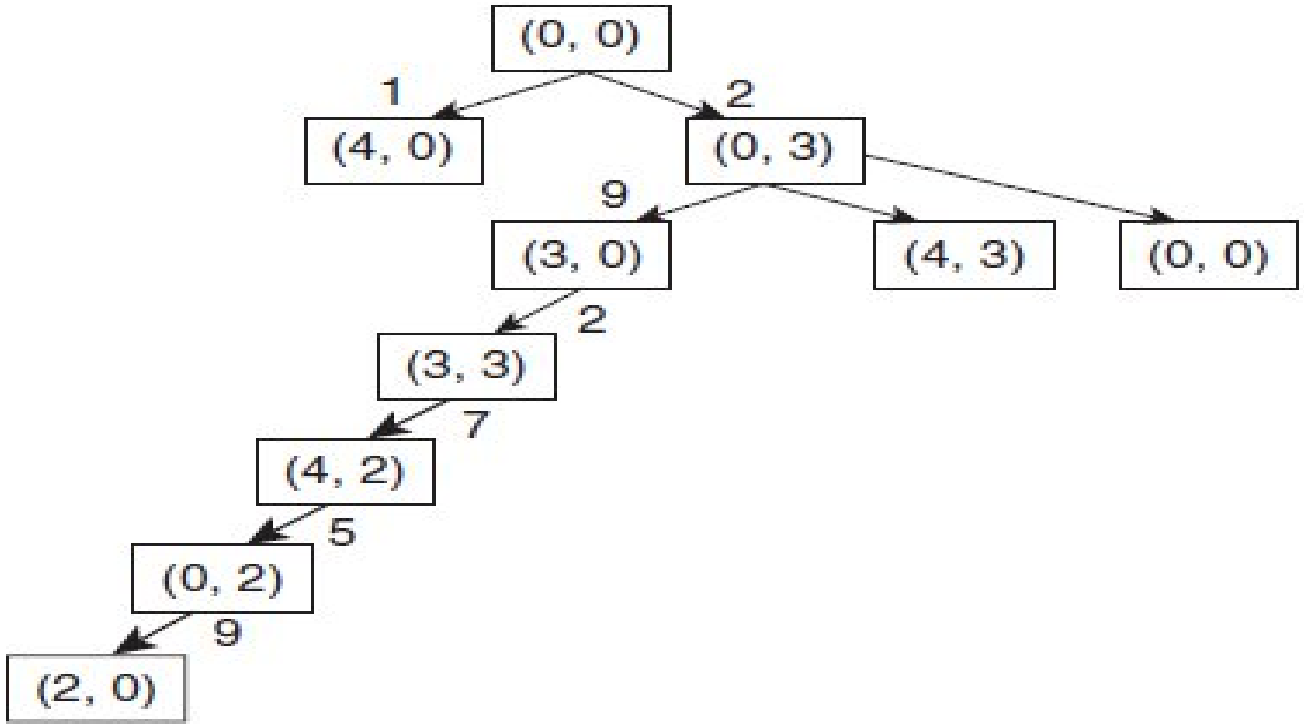
$(x, y) \rightarrow (x + y, 0)$ if $(y > 0)$ and $(x + y) \leq 4$

Rule 10: Pour all water from 4G jug to 3G jug

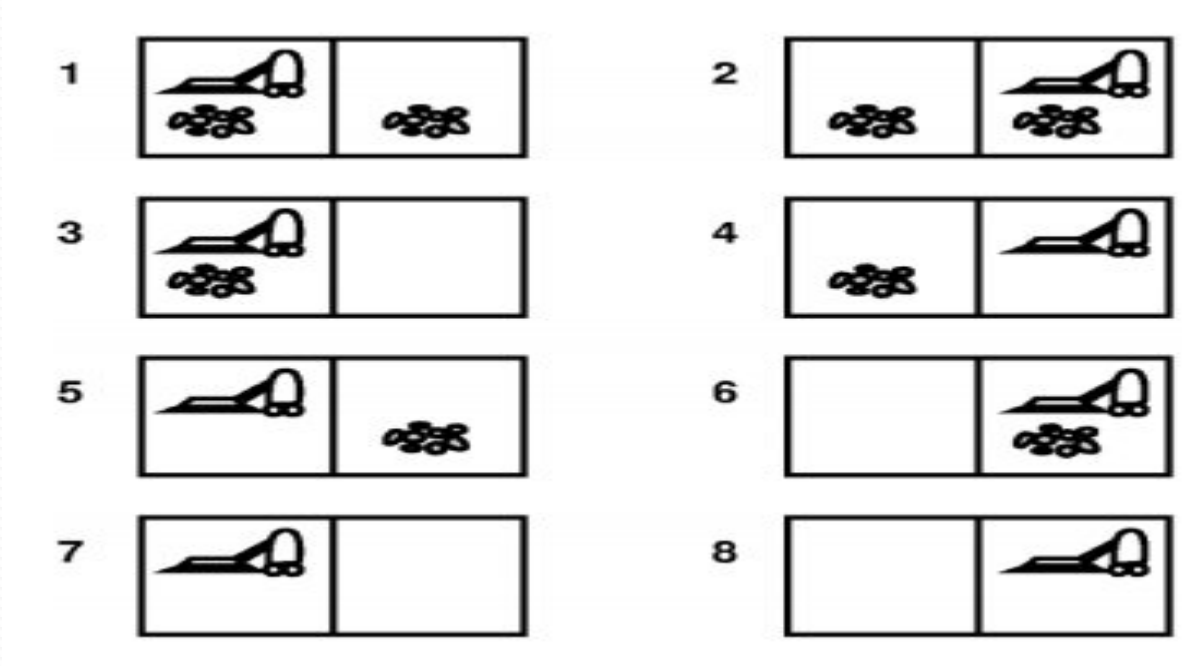
$(x, y) \rightarrow (0, x + y)$ if $(x > 0)$ and $(x + y) \leq 3$

Problem formulation of the Water Jug Problem

- We generate a tree which represents all the possible states in the problem, and this is called as *state space. In a state space, we will find a path* to the goal this is called a *solution path. Solution path is found from state space by state space search.* This approach of solution is called as *state-space approach.*
- **Solution Tree: To represent all the possible states in the state space.**

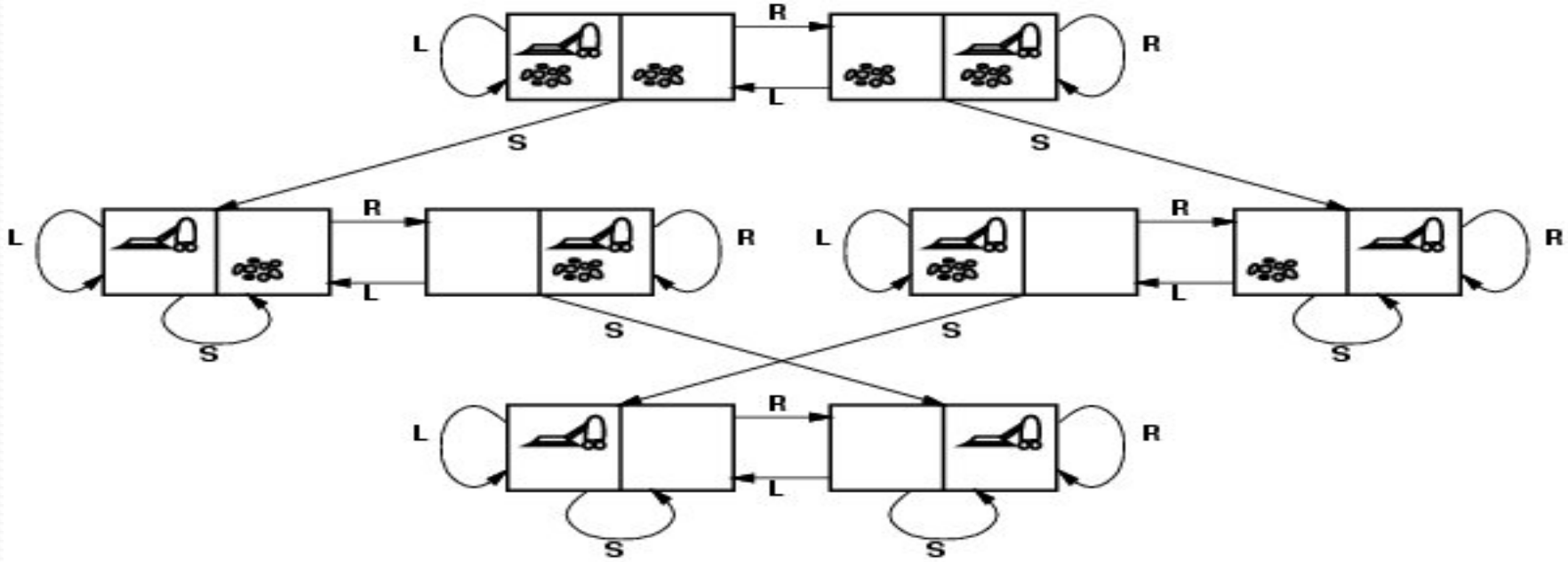


Vacuum world state



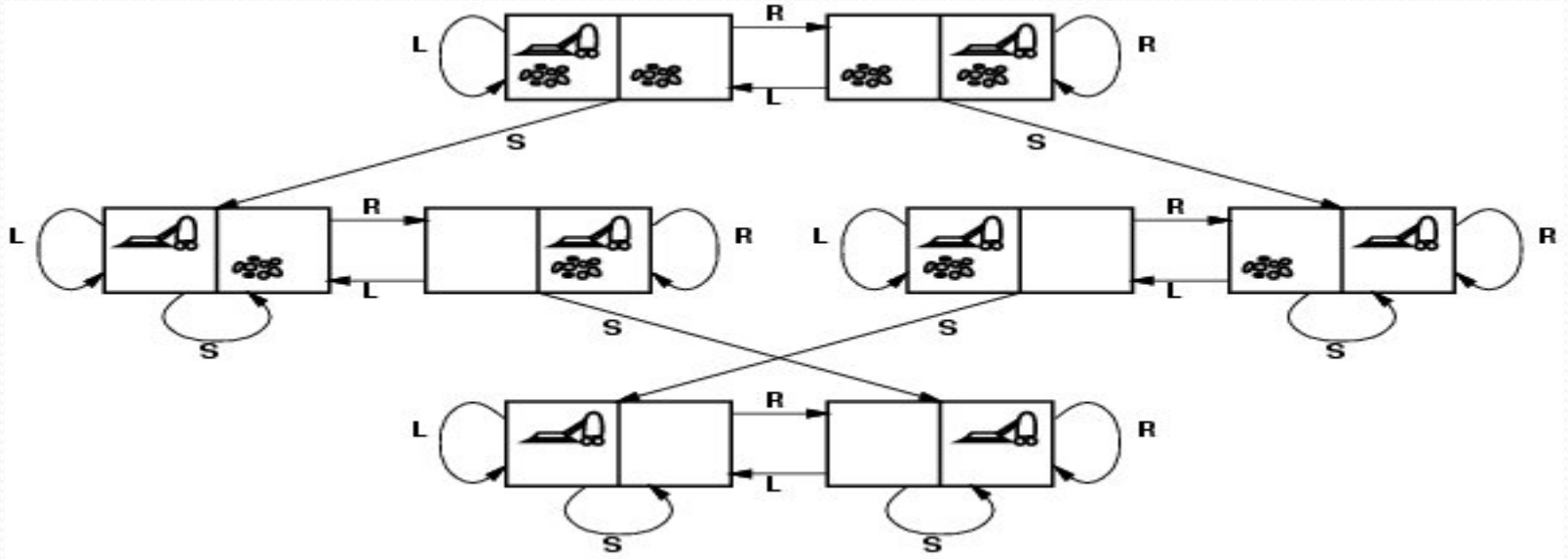
- states?
- Initial state?
- actions?
- goal test?
- path cost?

Vacuum world state space graph



- states?
- Initial state?
- actions?
- goal test?
- path cost?

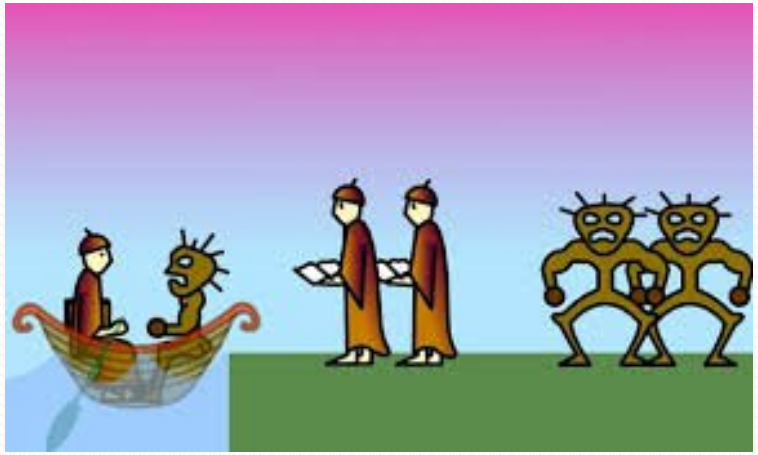
Vacuum world state space graph



- states? integer dirt and robot location
- Initial state? Dirt in both locations and the vacuum cleaner in one of them
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations

Missionaries and Cannibals

- Three missionaries and three cannibals are on one side of a river that they wish to cross.
- A boat is available that can hold at most two people.
- You must never leave a group of missionaries outnumbered by cannibals on the same bank.
- Find an action sequence that brings everyone safely to the opposite bank.



Missionaries and Cannibals

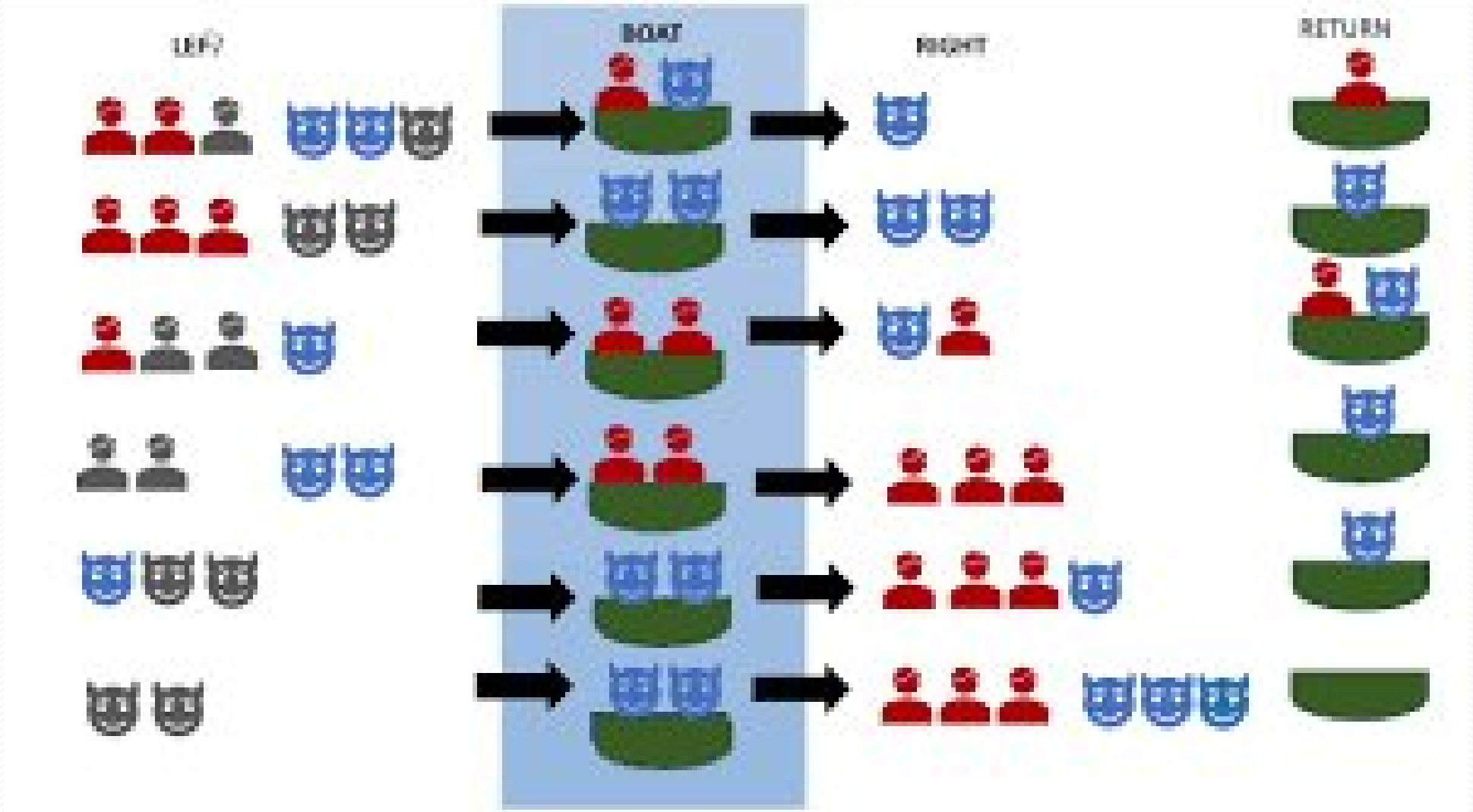
- **Goal**: Move all the missionaries and cannibals across the river.
- **Constraint:** Missionaries can never be outnumbered by cannibals on either side of river, or else the missionaries are killed.
- **State:** configuration of missionaries and cannibals and boat on each side of river.
- **Initial State:** 3 missionaries, 3 cannibals and the boat are on the near bank
- **Operators:** Move boat containing some set of occupants across the river (in either direction) to the other side.

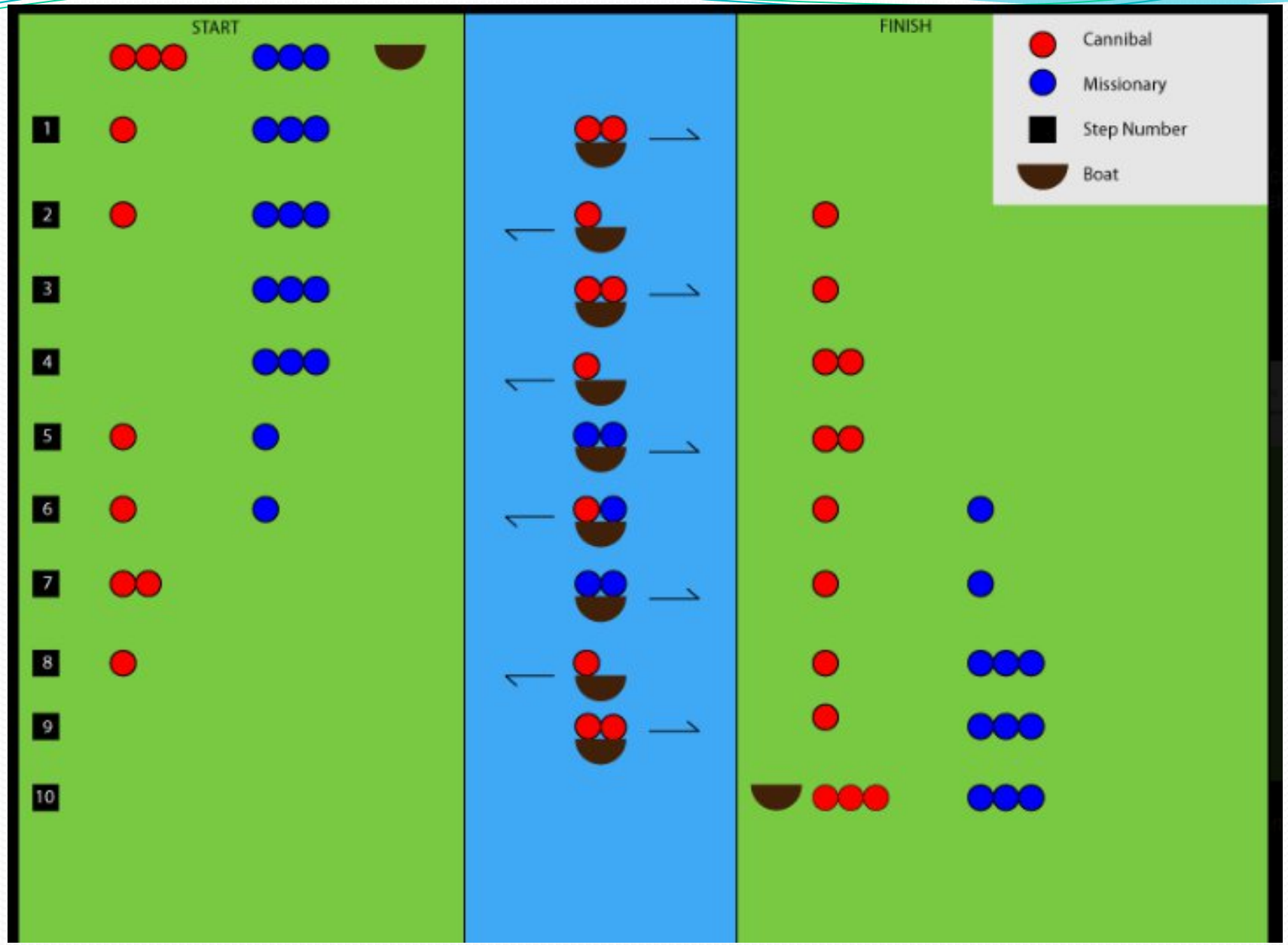
Solution steps:

SI = [3M, 3C, L]

State of Left Side of river	*Action*	*State of Right side of river*	*Score*
[3M, 1C, L]	→ Send 2C	[0M, 2C, R]	6
[3M, 2C, L]	← Send 1C	[0M, 1C, R]	6
[3M, 0C, L]	→ Send 2C	[0M, 3C, R]	6
[3M, 1C, L]	← Send 1C	[0M, 2C, R]	4
[1M, 1C, L]	→ Send 2M	[2M, 2C, R]	5 average taken = (4 + 6)/2
[2M, 2C, L]	→ Send 2M	[2M, 1C, R]	4
[0M, 2C, L]	→ Send 2M	[3M, 1C, R]	6
[0M, 3C, L]	← Send 1C	[3M, 0C, R]	6
[0M, 1C, L]	→ Send 2C	[3M, 2C, R]	6
[0M, 2C, L]	← Send 1C	[3M, 1C, R]	6
[0M, 0C, L]	→ Send 2C	[3M, 3C, R]	Goal state $\sum = 61$

Total cost = 61 units

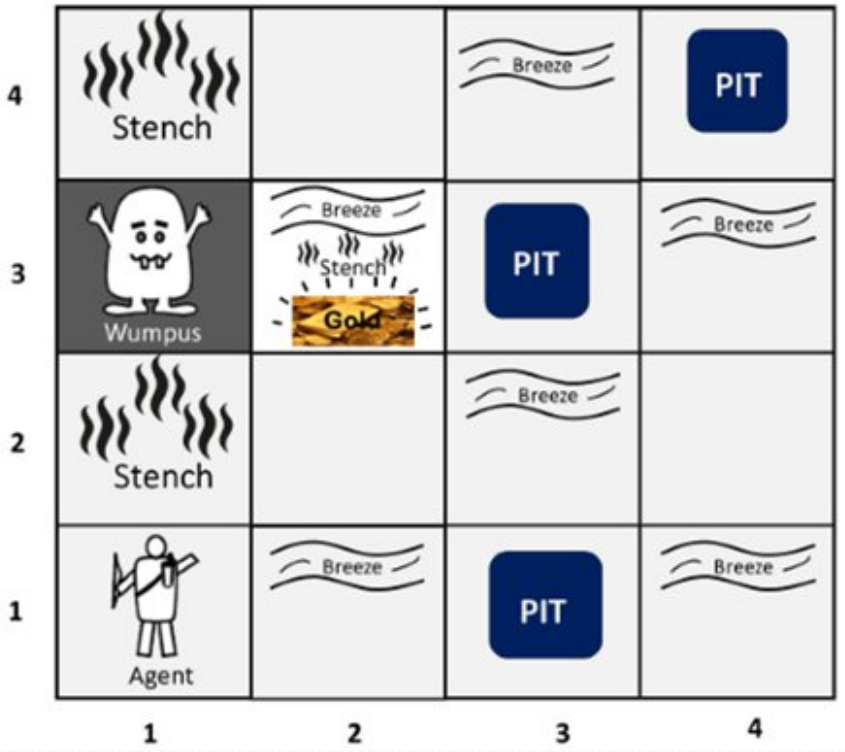

LEFT
BOAT
RIGHT
RETURN


START
FINISH
Cannibal
Missionary
Step Number
Boat
1
2
3
4
5
6
7
8
9
10

Wumpus World Problem

Problem definition: A wumpus world problem comprises of a 4 × 4 grid with the heap of gold in one of the squares. An agent enters the grid and the aim is to find the heap of gold.

The Hurdles: The grid has live wumpus in one of the squares. The agent is eaten up by the wumpus if enters the wumpus square also there are pits in some square and agent dies falling into pits the agent enters any of these squares. Also, there exist walls in between some of the cells.



Agent in a Wumpus world:

- The agent perceives
  - a stench in the square containing the wumpus and in the adjacent squares (not diagonally)
  - a breeze in the squares adjacent to a pit
  - a glitter in the square where the gold is
  - a bump, if it walks into a wall
  - a woeful scream everywhere in the cave, if the wumpus is killed.
- The percepts will be given as a five-symbol list:
  - If there is a stench, and a breeze, but no glitter, no bump, and no scream, the percept is

    [Stench, Breeze, None, None, None]

The actions of the agent in Wumpus game are:

- **go forward**
- **turn right**
- **turn left**
- **grab** means pick up an object that is in the same square as the agent
- **shoot** means fire an arrow in a straight line in the direction the agent is looking.
  - The arrow continues until it either hits and kills the wumpus or hits the wall.
  - The agent has only one arrow.
  - Only the first shot has any effect.
- **climb** is used to leave the cave.
  - Only effective in start field.
- **die** if the agent enters a square with a pit or a live wumpus

The agent's goal

The agent's goal is to find the gold and bring it back to the start as quickly as possible, without getting killed.

- 1000 points reward for climbing out of the cave with the gold
- -1 point deducted for every action taken
- 10000 points penalty for getting killed

The Wumpus agent's first step

Now, consider the environment as shown:

4	4, 1	4, 2	4, 3	4, 4
3	Wumpus 3, 1	Gold 3, 2	Pit 3, 3	3, 4
2	2, 1	2, 2	2, 3	2, 4
1	Start 1, 1	1, 2	Pit 1, 3	1, 4
	1	2	3	4

For the grid shown earlier, we explore it till the agent reaches the square-containing gold and give the final solutions and steps.

Step 1: The agent is at the start position 1, 1.
Sensors {none, none, none, none, none}
Action → Move to (1, 2)
Score → −1

4	4, 1	4, 2	4, 3	4, 4
3	Wumpus 3, 1	Gold 3, 2	Pit 3, 3	3, 4
2	2, 1	2, 2	2, 3	2, 4
1	Start 1, 1 **Agent**	1, 2	Pit 1, 3	1, 4
	1	2	3	4

Step 2: Now after Step 1, the agent is at the position 1, 2
Sensors {none, breeze, none, none, none}
Action → Move to (1, 1)
Score → −1 − 1 = −2

4	4, 1	4, 2	4, 3	4, 4
3	Wumpus 3, 1	Gold 3, 2	Pit 3, 3	3, 4
2	2, 1	2, 2	2, 3	2, 4
1	Start 1, 1	1, 2 **Agent**	Pit 1, 3	1, 4
	1	**2**	**3**	**4**

Step 3: Now after Step 2 the agent is at the position 1, 1.
Sensors {none, none, none, none, none}
Action → Move to (2, 1)
Score → −2 − 1 = −3

4	4, 1	4, 2	4, 3	4, 4
3	Wumpus 3, 1	Gold 3, 2	Pit 3, 3	3, 4
2	2, 1	2, 2	2, 3	2, 4
1	Start 1, 1 **Agent**	1, 2	Pit 1, 3	1, 4
	1	**2**	**3**	**4**

Step 4: After Step 3, the agent is at the position 2, 1.
Sensors {stench, none, none, none, none}
Action → Shoot at [3, 1]
Score → −3 − 10 = −13

4	4, 1	4, 2	4, 3	4, 4
3	Wumpus 3,1	Gold 3, 2	Pit 3, 3	3, 4
2	2, 1 **Agent**	2, 2	2, 3	2, 4
1	Start 1, 1	1, 2	Pit 1, 3	1, 4
	1	**2**	**3**	**4**

Sensors {none, none, none, none, none}
Action → Move to [3, 1]
Score → −3 − 10 − 1 = −14
Now, the wumpus is killed after shoot action hence the agent can move to [3, 1].

Step 5: Now after Step 4, the agent is at the position 3, 1.
Sensors {none, none, none, none, none}
Action → Move to [3, 2]
Score → −15

4	4, 1	4, 2	4, 3	4, 4
3	3, 1 **Agent**	Gold 3, 2	Pit 3, 3	3, 4
2	2, 1	2, 2	2, 3	2, 4
1	Start 1, 1	1, 2	Pit 1, 3	1, 4
	1	**2**	**3**	**4**

The agents get the gold after step 5 when it moves to [3, 2]

Final Goal

Sensors {none, none, glitter, none, none}

Action → Grab Gold

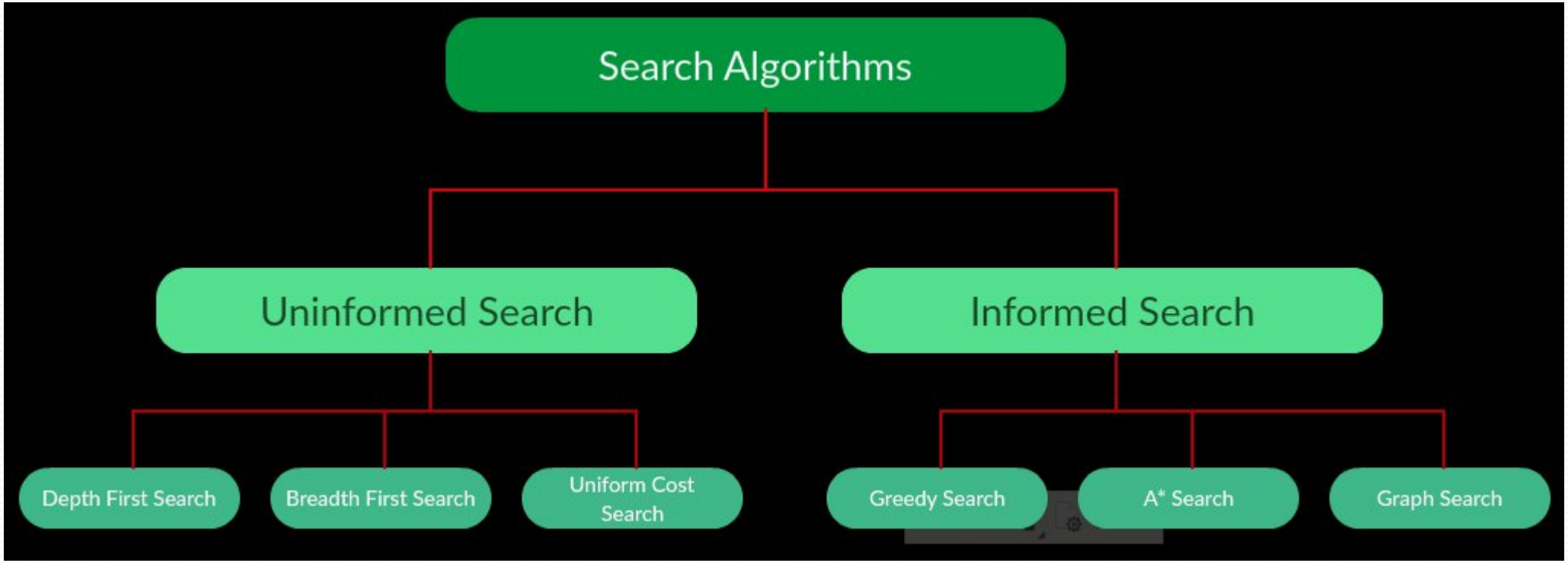
Score → −15 + 1000 = 985

4	4, 1	4, 2	4, 3	4, 4
3	3, 1	Gold 3, 2 **Agent**	Pit 3, 3	3, 4
2	2, 1	2, 2	2, 3	2, 4
1	Start 1, 1	1, 2	Pit 1, 3	1, 4
	1	2	3	4

Search Algorithm

- The process of looking for a sequence of actions that reaches the goal is called search.
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.

Search Algorithms



Uninformed search

Uninformed search strategies use only the information available in the problem definition.

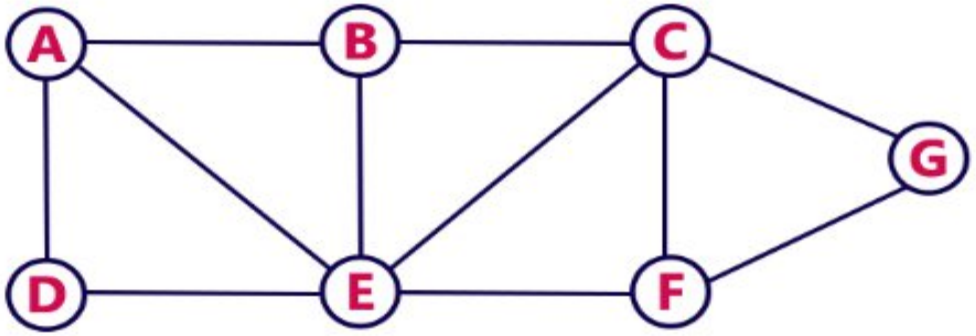
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

BFS

- BFS is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, etc.
- BFS is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion.
- This is achieved by using a FIFO queue.

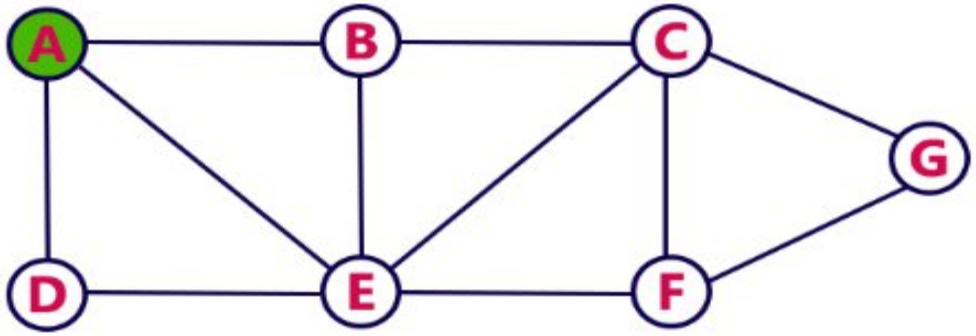
BFS Example

Consider the following example graph to perform BFS traversal



Step 1:

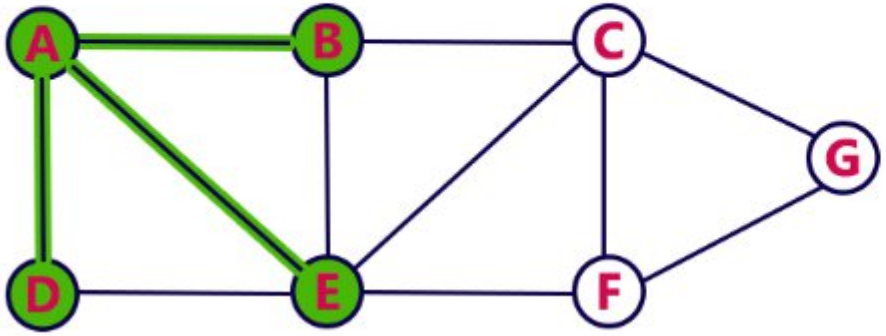
- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



Queue

Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..

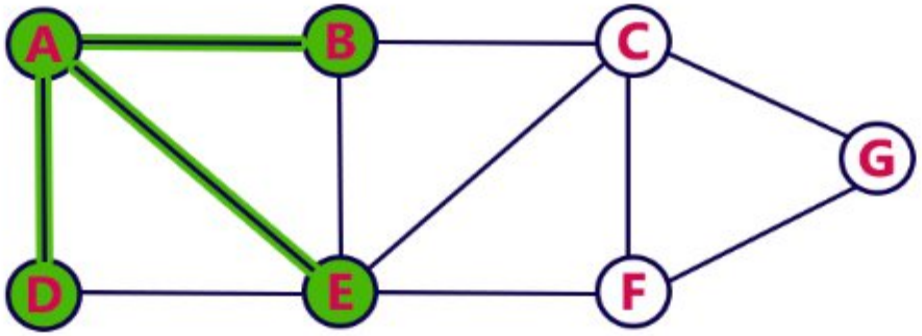


Queue

	D	E	B			

Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.

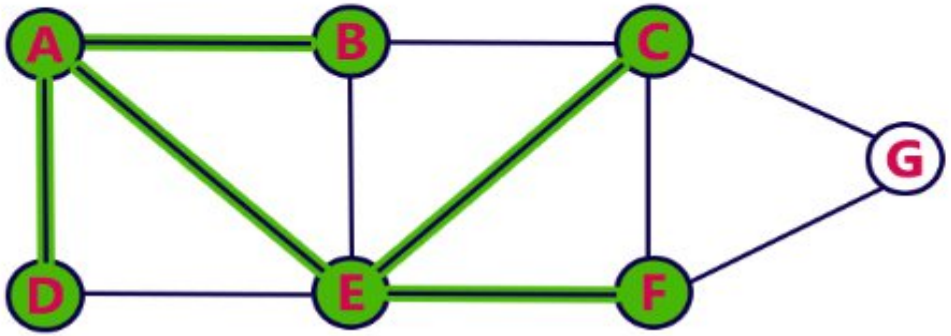


Queue

		E	B			

Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.

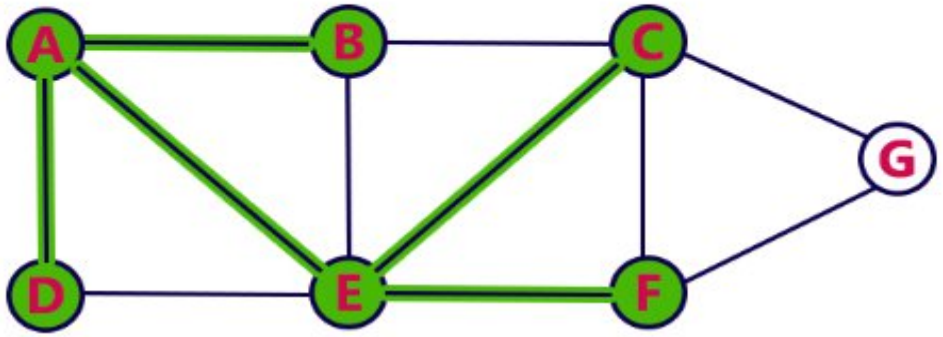


Queue

			B	C	F	

Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.

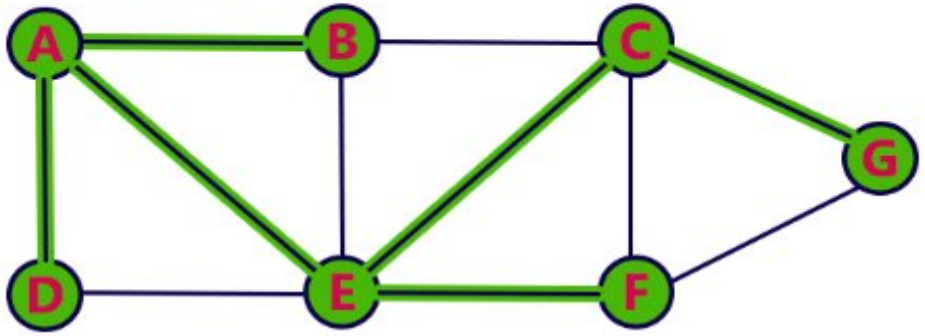


Queue

				C	F	

Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.

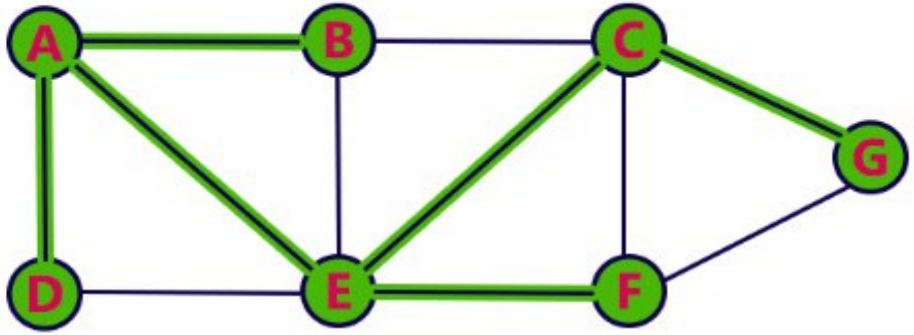


Queue

					F	G

Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

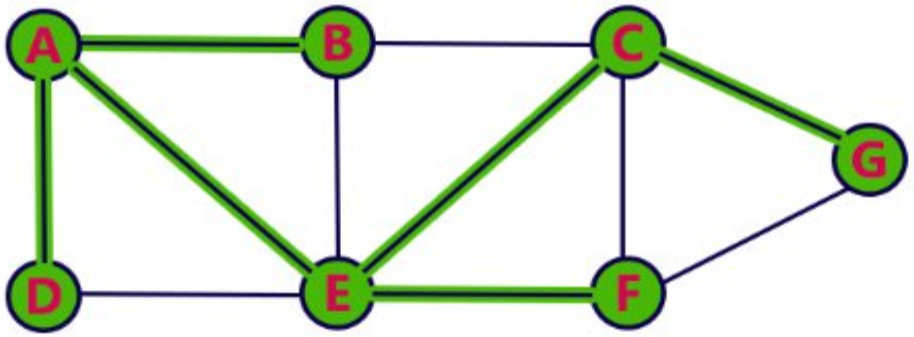


Queue

						G

Step 8:

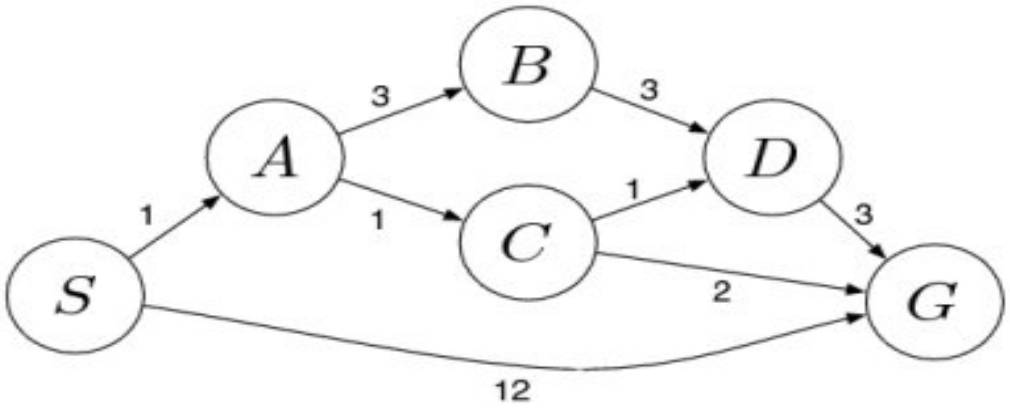
- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



Queue

Uniform-cost search

- Expand least-cost unexpanded node



Initialization: {[S, 0]}

Iteration1: {[S->A, 1], [S->G, 12]}

Iteration2: {[S->A->C, 2], [S->A->B, 4], [S->G, 12]}

Iteration3: {[S->A->C->D, 3], [S->A->B, 4], [S->A->C->G, 4], [S->G, 12]}

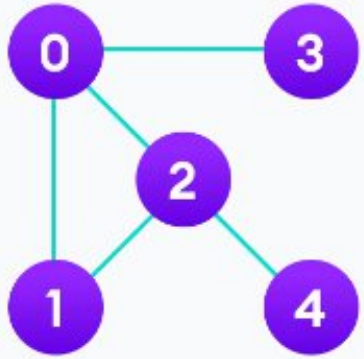
Iteration4: {[S->A->B, 4], [S->A->C->G, 4], [S->A->C->D->G, 6], [S->G, 12]}

Iteration5: {[S->A->C->G, 4], [S->A->C->D->G, 6], [S->A->B->D, 7], [S->G, 12]}

Iteration6 gives the final output as S->A->C->G.

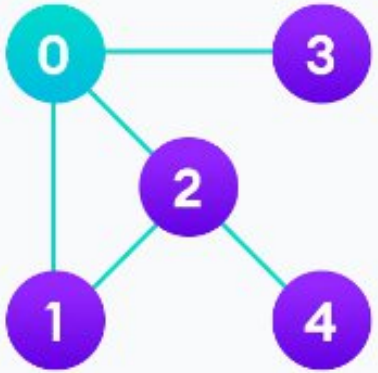
DFS

- DFS always expands the deepest node in the current frontier of the search tree. The search proceeds immediately to the deepest level fo the search tree, where the nodes have no successors.
- DFS is an instance of the general graph-search algorithm which uses a LIFO queue.



Visited

Stack

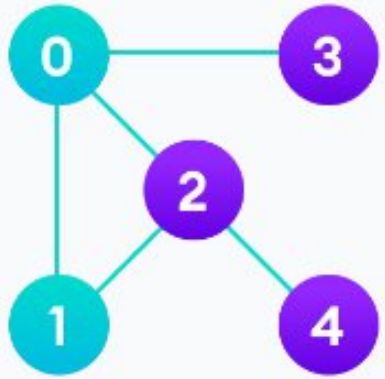


0				

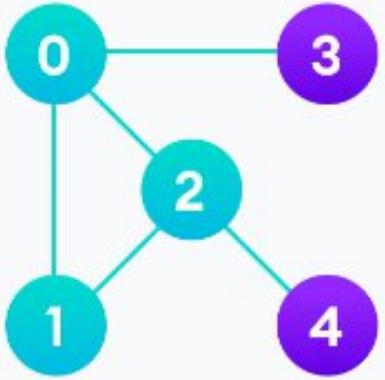
Visited

1	2	3		

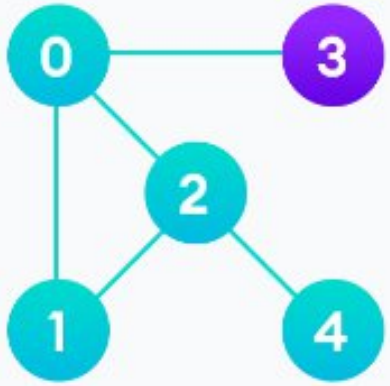
Stack



0	1				Visited
2	3				Stack



0	1	2			Visited
4	3				Stack

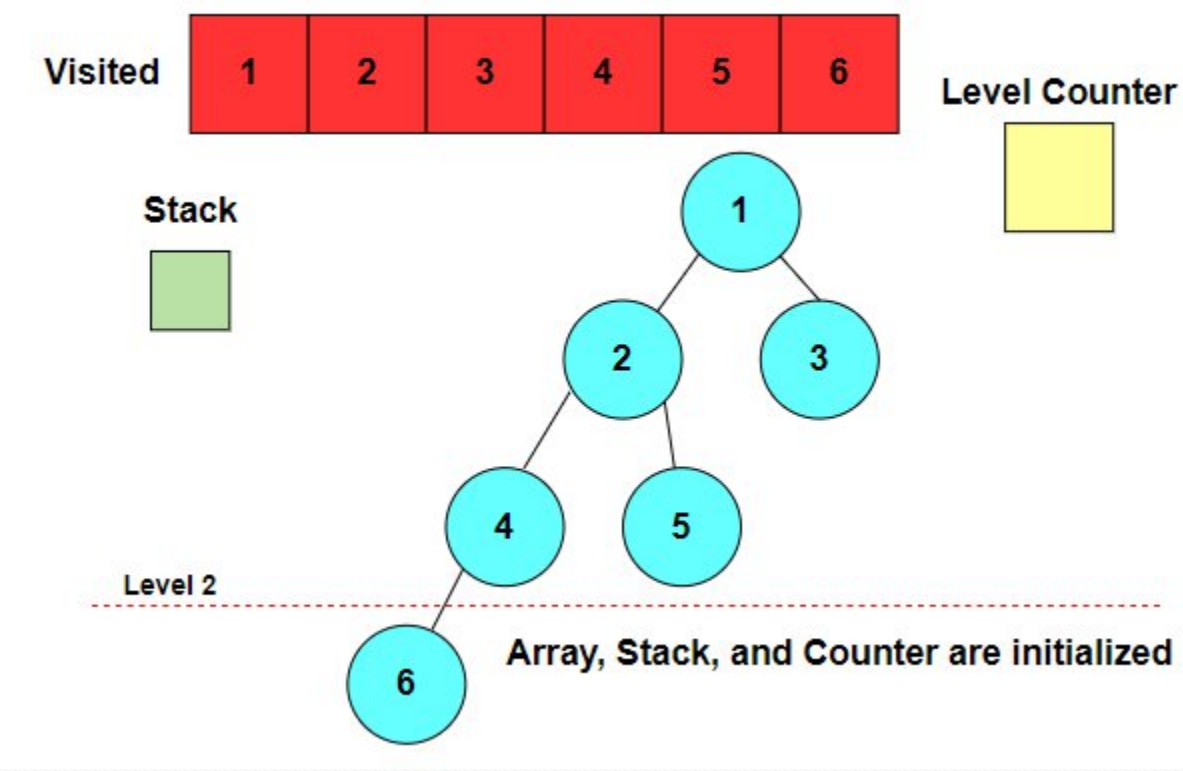


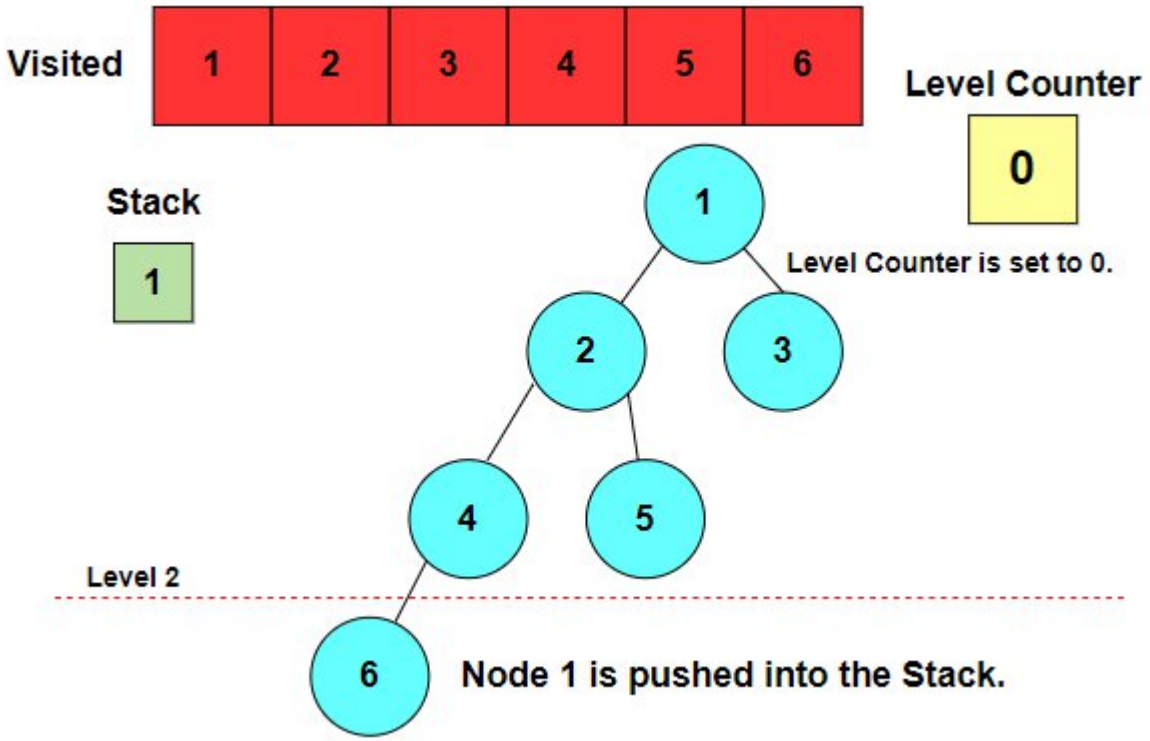
0	1	2	4		Visited
3					Stack

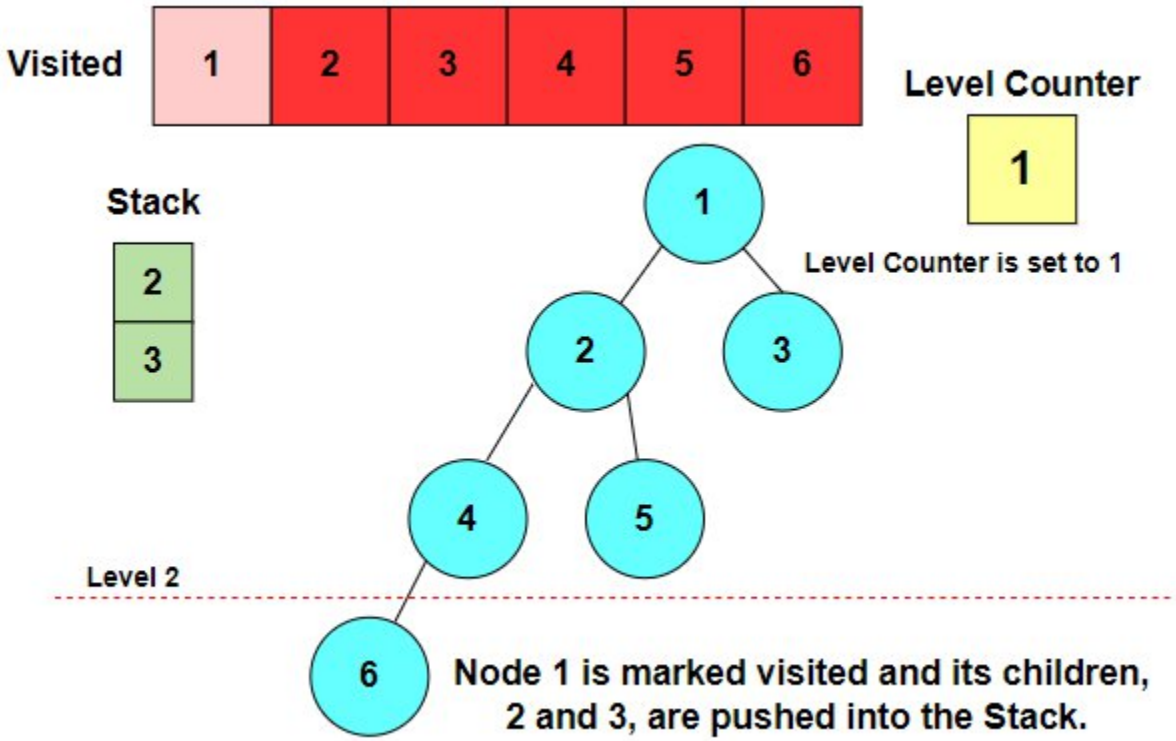
0	1	2	4	3	Visited
					Stack

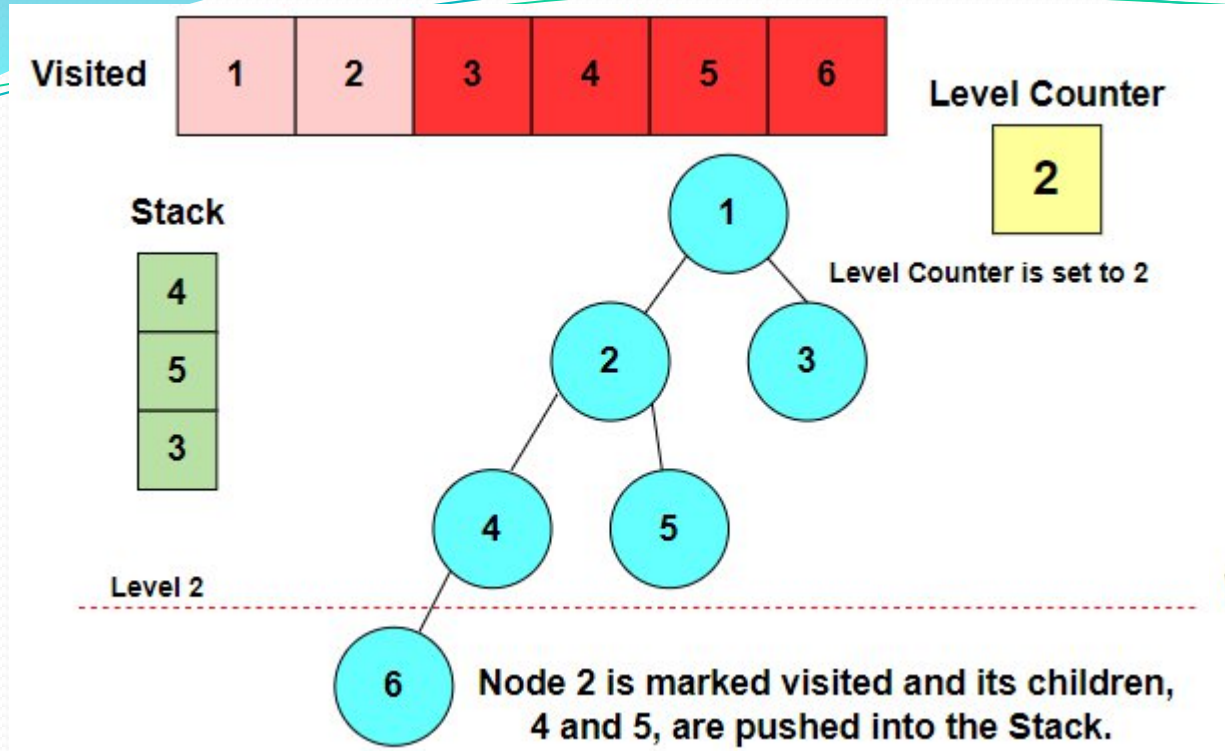
Depth-limited search

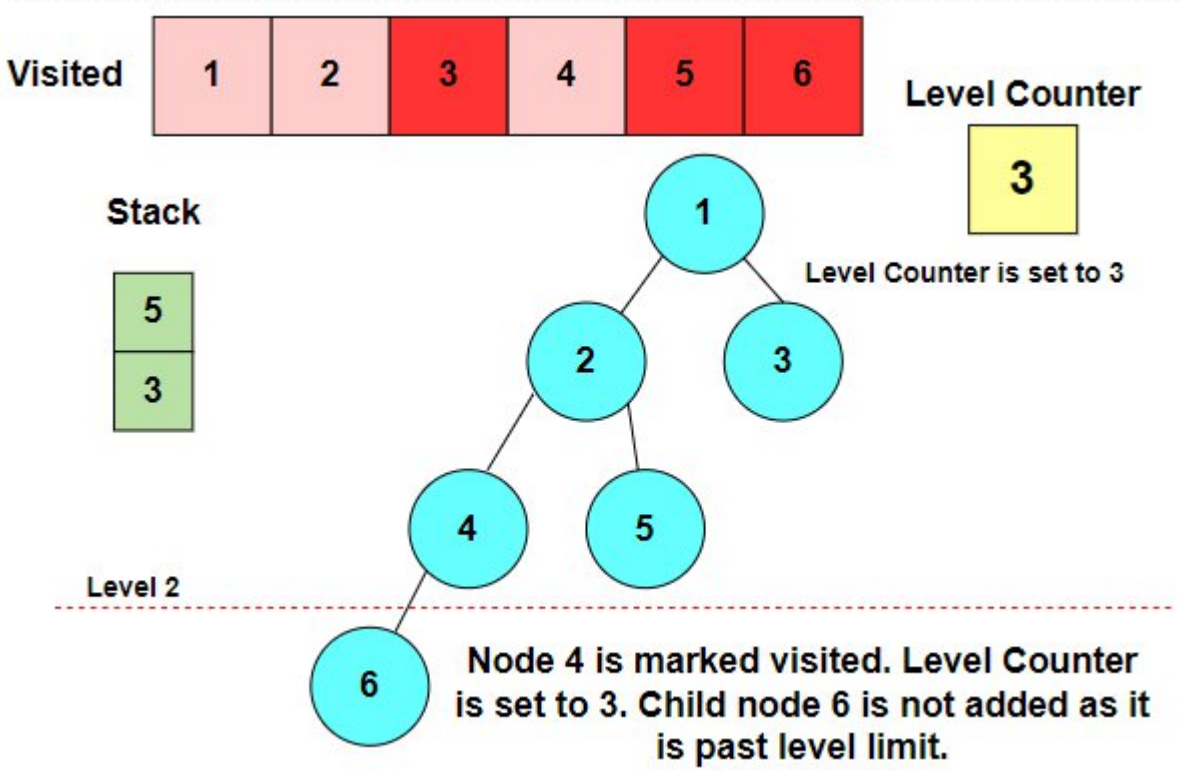
- The failure of DFS in infinite state spaces can be alleviated by making DFS more flexible with a pre-determined depth limit l, i.e., nodes at depth l have no successors.

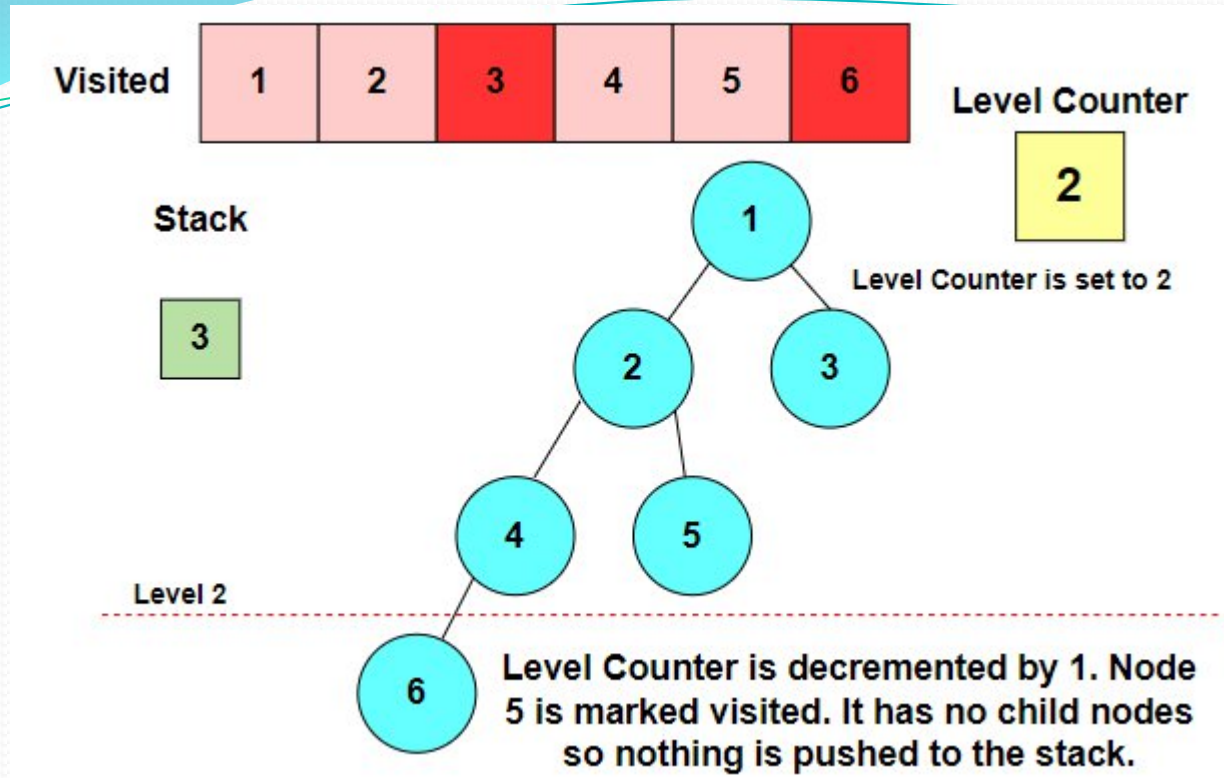


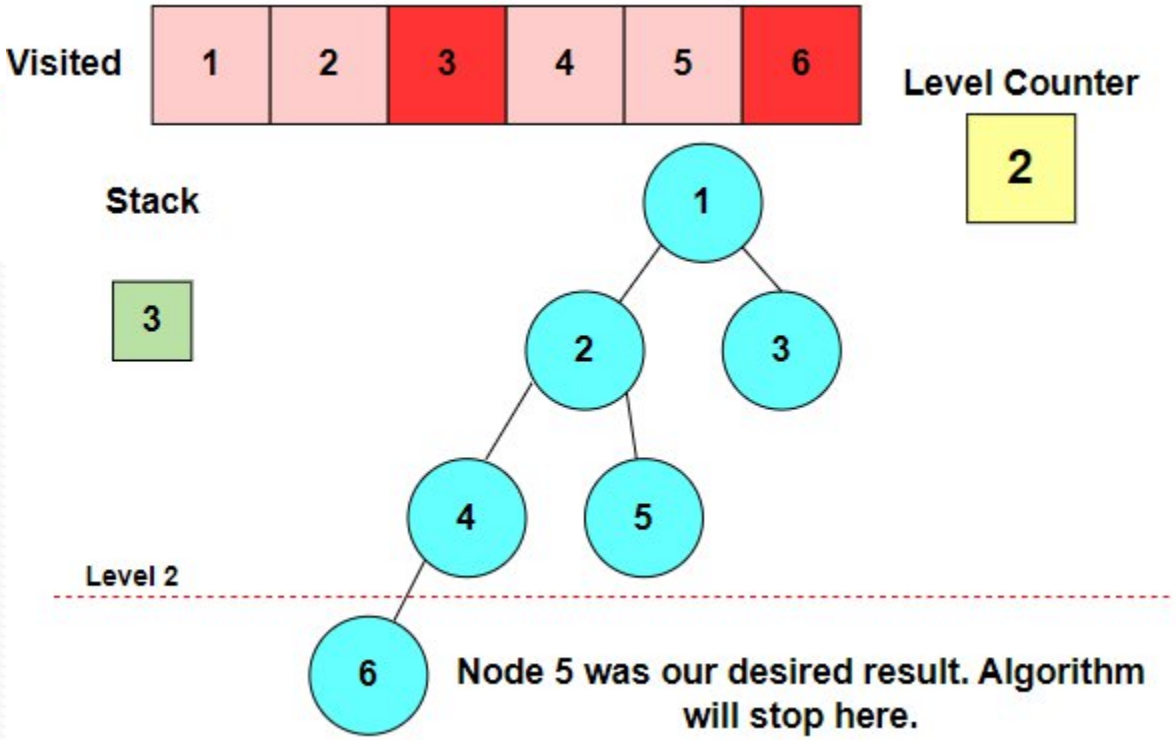

Visited
1
2
3
4
5
6
Level Counter
0
Stack
1
Level Counter is set to 0.
1
2
3
4
5
Level 2
6
Node 1 is pushed into the Stack.


Visited
1
2
3
4
5
6
Level Counter
1
Stack
2
3
Level Counter is set to 1
1
2
3
4
5
Level 2
6
Node 1 is marked visited and its children,
2 and 3, are pushed into the Stack.


Visited
1
2
3
4
5
6
Level Counter
2
Stack
4
5
3
1
2
3
4
5
6
Level Counter is set to 2
Level 2
Node 2 is marked visited and its children, 4 and 5, are pushed into the Stack.


Visited
1
2
3
4
5
6
Level Counter
3
Stack
5
3
1
2
3
4
5
6
Level Counter is set to 3
Level 2
Node 4 is marked visited. Level Counter is set to 3. Child node 6 is not added as it is past level limit.


Visited
1
2
3
4
5
6
Level Counter
2
Stack
3
Level Counter is set to 2
1
2
3
4
5
Level 2
6
Level Counter is decremented by 1. Node 5 is marked visited. It has no child nodes so nothing is pushed to the stack.

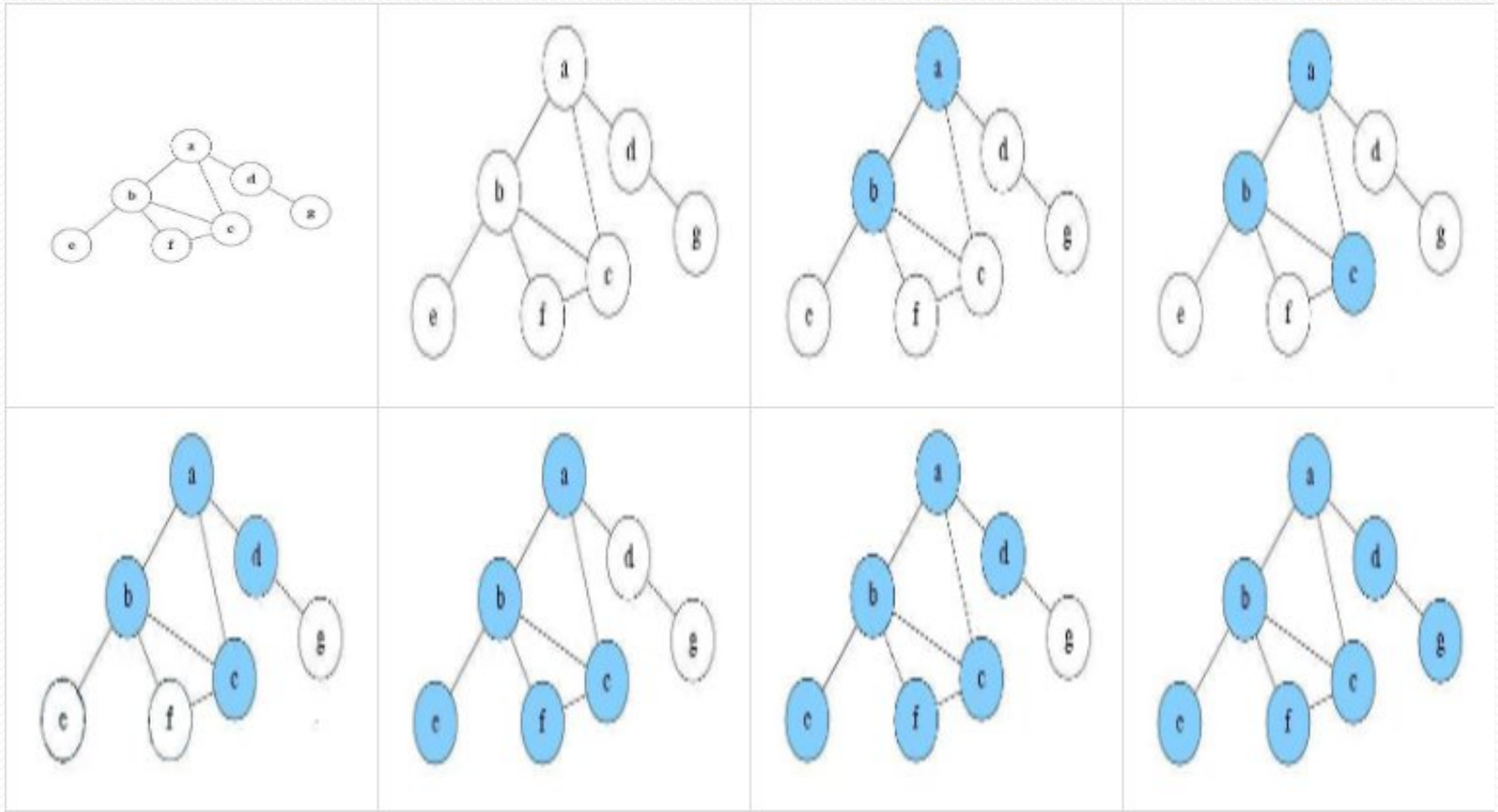

Visited
1
2
3
4
5
6
Level Counter
2
Stack
3
1
2
3
4
5
Level 2
6
Node 5 was our desired result. Algorithm will stop here.

Iterative deepening search

- Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-limited search, that finds the best depth limit.
- It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.
- This will occur when the depth limit reaches d, the depth of the shallowest goal node.

Example for IDS

- Suppose the source node is node ‘a’. Assume also that the ‘solution’ node is node ‘g’.

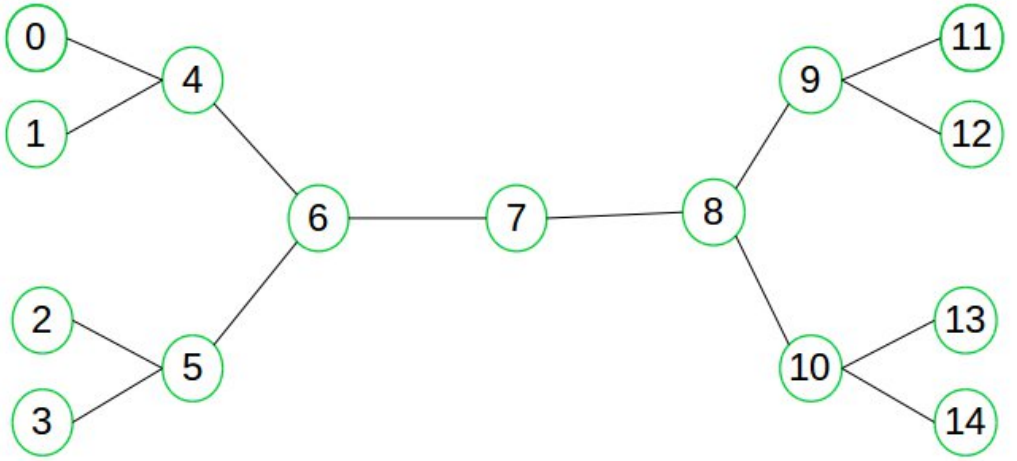


Bidirectional search

- Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs.
- One starting from initial vertex and other starting from goal vertex.
- **The search terminates when two graphs intersect.**

Example for Bidirectional search

- Suppose we want to find if there exists a path from vertex 0 to vertex ’14’. Here we can execute two searches, one from vertex ‘0’ and other from vertex ’14’.
- When both forward and backward search meet at vertex 7, we know that we have found a path from node 0 to 14 and search can be terminated now.



Comparing Uniform Search Strategies

- A. Time Complexity The time complexity of an algorithm is an expression for the worst – case amount of time it will take to run.
- B. Space Complexity The space complexity of an algorithm is an expression for the worst – case amount of memory that the algorithm will use (number of nodes).
- C. Optimality A search algorithm is optimal if, when it finds a solution it is the best solution.
- D. Completeness A search algorithm is complete if, whenever at least one solution exists, the algorithm is guaranteed to find a solution within a finite amount of time.

- Evaluation of search strategy, b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit.

Criteria	BFS	DFS	IDS	UCS	DLS
Time Complexity	$O(b^{d+1})$	$O(b^{m})$	$O(b^{d})$	$O(b^{\lfloor 1+C^*/e \rfloor})$	$O(b^{l})$
Space Complexity	$O(b^{d+1})$	O(bm)	O(bd)	$O(b^{\lfloor 1+C^*/e \rfloor})$	O(b*l*)
Optimality	Yes	No	Yes	Yes	No
Completeness	Yes	No	Yes	Yes	No

Hill Climbing

- This algorithm is a classic optimization method that mimics the process of ascending a hill to reach the peak (the optimal solution).
- It iteratively improves the current solution by making small changes, and it's particularly suited for local optimization problems.

Core Components of Hill Climbing

- **Initial State:** This initial state serves as the starting point for the algorithm's exploration.
- **Successor Function:** The successor function is responsible for generating neighboring solutions based on the current state. It explores the immediate neighborhood of the current solution, typically by making small, incremental changes.
- **Objective Function:** The objective function plays a pivotal role in Hill Climbing. It evaluates the quality of a solution, quantifying how well it performs in the context of the problem being solved. The objective function guides the algorithm by providing a measure of "goodness."

Variants of Hill Climbing

1. Simple Hill Climbing

- Simple Hill Climbing is the most basic form of the algorithm.
- It iteratively makes small moves to neighboring solutions, accepting a new solution only if it improves the objective function.
- If it doesn't, the algorithm terminates.
- This method is limited by its tendency to get stuck in local optima.

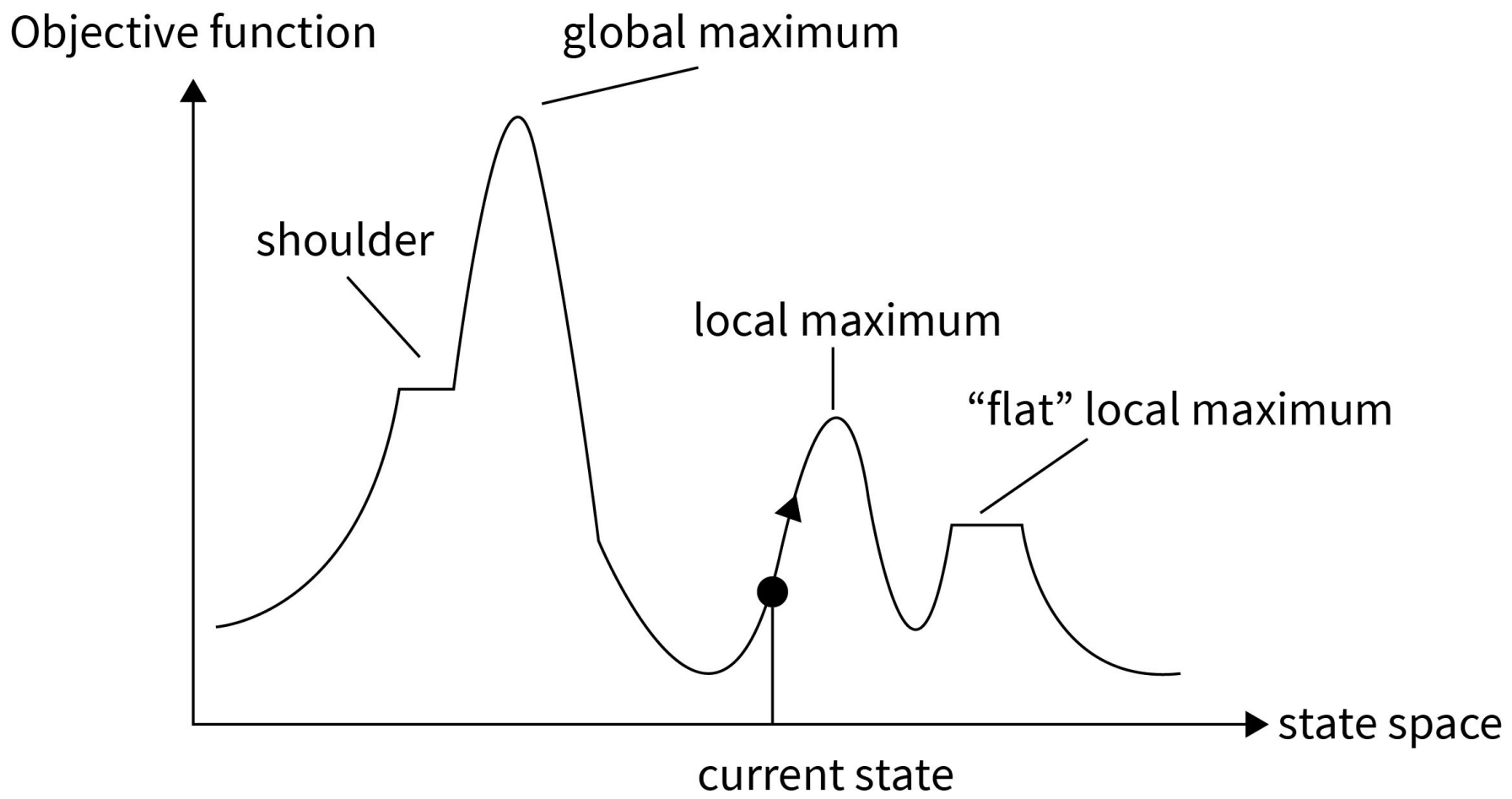
2. Steepest-Ascent Hill Climbing

- Steepest-Ascent Hill Climbing, also known as steepest ascent or gradient ascent, takes a more rigorous approach.
- It explores all neighboring solutions and selects the one that offers the most significant improvement in the objective function.
- This helps mitigate the problem of getting stuck in local optima to some extent.

3. Stochastic Hill Climbing

- Stochastic Hill Climbing introduces an element of randomness.
- Instead of always selecting the best neighboring solution, it probabilistically accepts solutions that are worse than the current one.
- This randomness allows the algorithm to explore a broader solution space, potentially escaping local optima.

State-space Diagram for Hill Climbing



Informed Search Strategies

INFORMED SEARCH

- This search strategy uses additional information or heuristics to make more accurate decisions about which paths to explore first.
- These heuristics provide estimates of how close a given state is to the goal, guiding the search toward more promising solutions.
- Informed search is particularly useful in solving complex problems efficiently, as it can significantly reduce the search space and improve the speed of finding solutions.

Best First Search

- Start with an initial node and add it to the open list.
- While the open list is not empty.
  - Select the node with the lowest estimated cost (based on a heuristic function) from the open list.
  - If the selected node is the goal, return the solution.
  - Otherwise, expand the selected node by generating its successors.
  - Evaluate each successor node using the heuristic function and add them to the open list.
- If the open list becomes empty without reaching the goal, the search fails.

A* Search

A* Algorithm works as-

- It maintains a tree of paths originating at the start node.
- It extends those paths one edge at a time.
- It continues until its termination criterion is satisfied.

- A* Algorithm extends the path that minimizes the following function-

$$\mathbf{f(n) = g(n) + h(n)}$$

  - Here, ‘n’ is the last node on the path
  - g(n) is the cost of the path from start node to node ‘n’
  - h(n) is a heuristic function that estimates cost of the cheapest path from node ‘n’ to the goal node

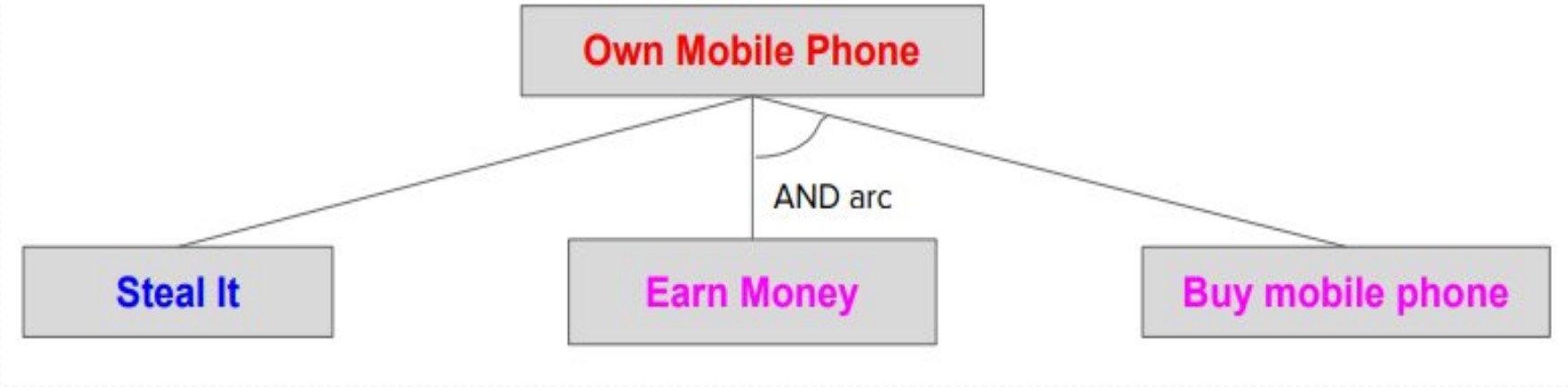
Differences between Best First Search and A*search

- Best-First search is a searching algorithm used to find the shortest path which uses distance as a heuristic. The distance between the starting node and the goal node is taken as heuristics. It defines the evaluation function for each node **n** in the graph as **f(n) = h(n)** where **h(n)** is heuristics function.
- A*search is a searching algorithm used to find the shortest path which calculates the cost of all its neighboring nodes and selects the minimum cost node. It defines the evaluation function **f(n) = g(n) + h(n)** where, **h(n)** is heuristics function and g(n) is the past knowledge acquired while searching.

Parameters	Best-First Search	A* Search
Evaluation Function	The evaluation function for best-first search is f(n) = h(n).	The evaluation function for A* search is f(n) = h(n) + g(n).
Past Knowledge	This search algorithm does not involve past knowledge.	This search algorithm involves past knowledge.
Completeness	Best-first search is not complete.	A* search is complete.
Optimal	Best-first search is not optimal as the path found may not be optimal.	A* search is optimal as the path found is always optimal.

AO* Search

- The AO* method **divides** any given difficult **problem into a smaller group** of problems that are then resolved **using the AND-OR** graph concept.
- AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems.



Differences between A* and AO*

A*	AO*
It represents an OR graph algorithm that is used to find a single solution (either this or that). It is a computer algorithm which is used in path-finding and graph traversal.	It represents an AND-OR graph algorithm that is used to find more than one solution by ANDing more than one branch.
It is used in the process of plotting an efficiently directed path between a number of points called nodes.	In this algorithm a similar procedure is followed but there are constraints traversing specific paths.
In this algorithm the tree is traversed in depth and keep moving and adding up the total cost of reaching the cost from the current state to the goal state and add it to the cost of reaching the current state.	The cost of all the paths which originate from the preceding node are added till that level, where you find the goal state regardless of the fact whether they take you to the goal state or not.

Differences between Informed Search and Uninformed Search

Parameters	Informed Search	Uninformed Search
Known as	It is also known as Heuristic Search.	It is also known as Blind Search.
Using Knowledge	It uses knowledge for the searching process.	It doesn't use knowledge for the searching process.
Performance	It finds a solution more quickly.	It finds solution slow as compared to an informed search.
Completion	It may or may not be complete.	It is always complete.
Cost Factor	Cost is low.	Cost is high.
Time	It consumes less time because of quick searching.	It consumes moderate time because of slow searching.
Direction	There is a direction given about the solution.	No suggestion is given regarding the solution in it.
Implementation	It is less lengthy while implemented.	It is more lengthy while implemented.

Differences between Informed Search and Uninformed Search

Parameters	Informed Search	Uninformed Search
Efficiency	It is more efficient as efficiency takes into account cost and performance. The incurred cost is less and speed of finding solutions is quick.	It is comparatively less efficient as incurred cost is more and the speed of finding the Breadth-Firstsolution is slow.
Computational requirements	Computational requirements are lessened.	Comparatively higher computational requirements.
Size of search problems	Having a wide scope in terms of handling large search problems.	Solving a massive search task is challenging.
Examples of Algorithms	•Greedy Search •A* Search •AO* Search •Hill Climbing Algorithm	•Depth First Search (DFS) •Breadth First Search (BFS) •Branch and Bound

Crypt arithmetic

- Find an assignment of digits (0, ..., 9) to letters so that a given arithmetic expression is true. examples: SEND + MORE = MONEY and

FORTY **Solution: 29786**
+ TEN **850**
+ TEN **850**
----- **-----**
SIXTY **31486**

F=2, O=9, R=7, etc.

Crypt-arithmetic

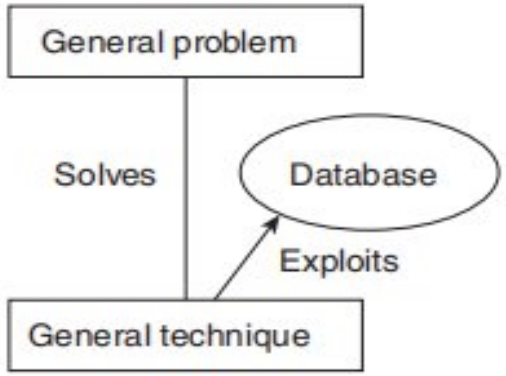
Find an assignment of digits to letters so that a given arithmetic expression is true. examples: SEND + MORE = MONEY and

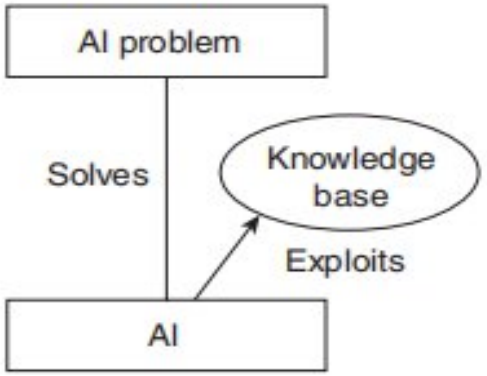
```
 FORTY      Solution:  29786
+  TEN                   850
+  TEN                   850
 -----                 -----
 SIXTY                 31486
F=2, O=9, R=7, etc.
```

- **State:** mapping from letters to digits
- **Initial State:** empty mapping
- **Operators:** assign a digit to a letter
- **Goal Test:** whether the expression is true given the complete mapping

Difference between Conventional Problem and AI Problem

AI Technique	*Procedural Technique*
It implements symbolic reasoning processing.	It implements numeric processing.
Implements heuristic search techniques.	Implements algorithmic search technique.
Solution steps are not explicit.	Solution steps are explicit.
Knowledge is imprecise.	Knowledge is precise.
It requires frequent modifications.	Modifications are rare.
It works on or implements inferential mechanism.	It works on or implements repetition mechanism.
It has a large knowledge base.	It has a large database.
Results are usually satisfactory.	Results are usually optimal.





Searching

Given the representation of a domain in the terms just described, a number of techniques can be applied to find solution.

1. Generate and Test
2. Random Search
3. Search Spaces

Assumptions in Basic Search

- The world is static
- The world is discretizable
- The world is observable
- The actions are deterministic

Features/characteristics of problem

Some of the main key features of a problem are listed as follows:

- Is the problem decomposable into set of subproblems?
- Can the solution step be ignored or undone?
- Is the problem universally predictable?
- Is a good solution to the problem obvious without comparison to all the possible solutions?
- Is the desire solution a state of world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem?
- Will the solution of the problem required interaction between the computer and the person?

UNINFORMED AND INFORMED SEARCH STRATEGIES

Search Algorithms

Search algorithms are one of the most important areas of Artificial Intelligence.

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.

Search Algorithm Terminologies:

1. **Search:** step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
   - **Search Space**: represents a set of possible solutions
   - **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.
1. **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.
2. **Actions:** It gives the description of all the available actions to the agent.

4. Transition model: A description of what each action do, can be represented as a transition model.

5. Path Cost: It is a function which assigns a numeric cost to each path.

6. Solution: It is an action sequence which leads from the start node to the goal node.

7. **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 searches

1. **Uninformed Search/Blind Search** :The search strategy, which has no information about the number of steps or path cost from the current state to goal state, but can only distinguish a goal state from a non goal state is known as *Uninformed search strategy* or *blind search strategy.*
2. **Informed Search**: Strategies that know whether one non-goal state is more promising than the other is called as *heuristic/ informed search techniques*. The name means that this kind of search strategies are aware whether the search process is moving in the direction of goal or in the opposite direction of goal.