➢Related Terminology

- Performance Measure of Agent – It is the criteria, which determines how successful an agent is.
- Behavior of Agent – It is the action that agent performs after any given sequence of percepts.
- Percept – It is agent's perceptual inputs at a given instance.
- Percept Sequence – It is the history of all that an agent has perceived till date.
- Agent Function – It is a map from the precept sequence to an action.

❖Characteristics of an Intelligence Agent

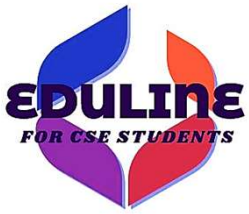
- Must sense
- Must act
- Must autonomous (to some extend)
- Must rational

❖Examples of Agents

- Human agent: eyes, ears, and other organs for sensors; – hands, legs, mouth, and other body parts for actuators
- Robotic agent: cameras and infrared range finders for sensors; – various motors, wheels, and speakers for actuators
- A software agent: function (Input) as sensors — function as actuators (output-Screening)

MODULE 2

PROBLEM SOLVING



PROBLEM STATE SPACE

- A Problem is the issue which comes across any system. A solution is needed to solve that particular problem.
- Technically, a problem is defined by its 'elements' and their 'relations'.
- To provide a formal description of a problem, we need to understand the following terms:

➢ Define a **state space** that contains all the possible configurations of the relevant objects, including some impossible ones.

➢Specify one or more states that describe possible situations, from which the problem solving process may start. These states are called **initial states.**

➢Specify one or more states that would be acceptable solution to the problem. These states are called **goal states.**

- A **state** is a representation of problem elements at a given moment.
- **A State space is the set of all states reachable from the initial state.**

❖ **A state space consists of the following elements**

➢ **A (possibly infinite) set of states**

- Out of the possible states, one state represents the start state that is the initial state of the problem.
- Each state represents some configuration reachable from the start state
- Out of the possible states, some states may be goal states (solutions)

➢**A set of rules**

- Applying a rule to the current state, transforms it to another or a new state in the state space
- All operators may not be applicable to all states in the state space

➢State spaces are used extensively in Artificial Intelligence (AI) to represent and solve problems.

❖Examples of Problems in Artificial Intelligence

➢The most prevalent problems that artificial intelligence has resolved are the following:

1. Chess
2. N-Queen problem
3. Tower of Hanoi Problem
4. Travelling Salesman Problem
5. Water-Jug Problem

❖Some Real World Problems

➢The real world problems that artificial intelligence has resolved are the following:

1. Route-finding problems
2. Touring problems
3. VLSI layout problem
4. Robot navigation
5. Automatic assembly sequencing
6. Internet searching
7. Speech Recognition System etc.

❖Problem characteristics

➢In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several dimensions.

- Is the problem decomposable into a set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?

- Is the desired solution a state of the world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
- Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

❖An Example:

Scenario:

➢You want to go on holiday to city 'Chandigarh' but currently you live in city 'Kottayam' And the flight for 'Chandigarh' leaves from another city 'Kochi'.

Goal:

- To reach city 'Kochi' from 'Kottayam' to catch the flight for 'Chandigarh'.

Formulate Problem:

- **States:** various cities
- **Action:** movement between the cities

Solution:

- Appropriate sequence of the cities.
- Say, **Kottayam->X->Y->Z->Kochi**

❖Problem Solving

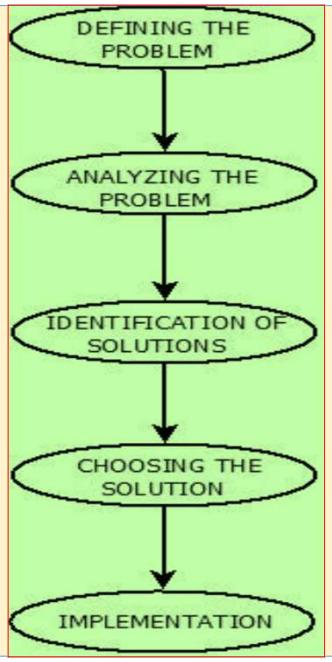
- Problem-solving is commonly known as the method to reach the desired goal or finding a solution to a given situation.
- In computer science, problem-solving refers to artificial intelligence techniques, including various techniques such as:

✓forming efficient algorithms,

✓heuristics, and

✓performing root cause analysis to find desirable solutions.

- **Steps in Problem Solving in AI**



❖Steps in Problem Solving in AI

- **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure.
- **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal.
- There are following five components involved in problem formulation:

✓**Initial State:** It is the starting state or initial step of the agent towards its goal.

✓**Actions:** It is the description of the possible actions available to the agent.

✓**Transition Model:** It describes what each action does.

✓**Goal Test:** It determines if the given state is a goal state.

✓**Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure.

- Remember, **an optimal solution has the lowest path cost among all the solutions.**

❖Problem space

- 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/actions* and *objects* that achieve the *goals*.

STATE SPACE SEARCH

➢**A state space consists of**

- A representation of the *states* the system can be in.
- 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.

➢A *state space* represents a problem in terms of *states* and *operators* that change states.

S : {S ,A, Action(S),Result(S,A),Cost(S,A)}

Where

- S - start, goal state.
- A - set of possible actions.
- Action(S) - the action performed from the set of states.
- Result(S,A) - Final sate
- Cost(S,A) – A constant value either in terms of time or distance.

❖Measuring problem-solving performance

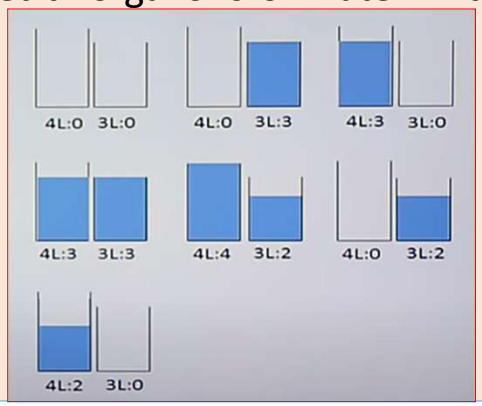
- **Completeness:** Is a solution found if one exists?
- **Optimality:** Does the strategy find the optimal solution?
- **Time Complexity:** How long does it take to find a solution?
- **Space Complexity:** How much memory is needed to perform the search?

➢Time and space complexity are measured in terms of problem difficulty defined by:

- b - branching factor of the search tree.
- d - depth of the shallowest goal node.
- m - maximum length of any path in the state space.

WATER JUG PROBLEM

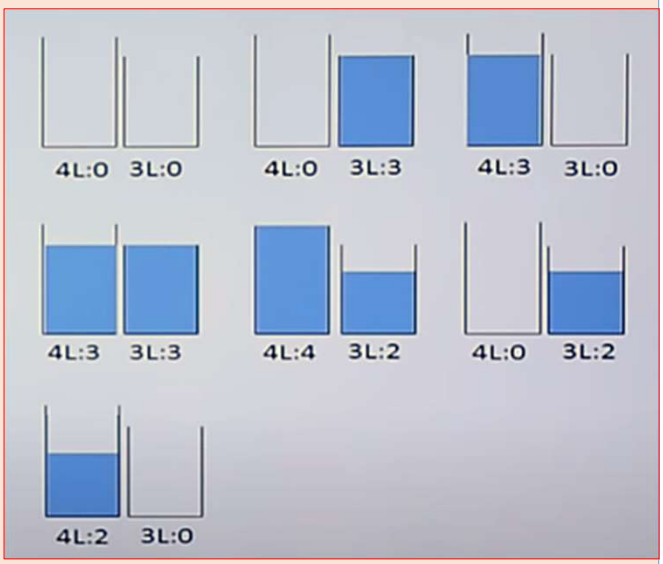
❖In this problem, we use two jugs called **four** and **three**; four holds a maximum of four gallons of water and **three** a maximum of three gallons of water. How can we get two gallons of water in the **four** jug?



- The start state is (0, 0) and the goal state is (2, n) where n may be any but it is limited to three holding from 0 to 3 gallons of water or empty.
- **Three** and **four** shows the name and numerical number shows the amount of water in jugs for solving the water jug problem.
- The major production rules for solving this problem are shown below:

Production Rules:

1. (x,y)=(4,y)
2. (x,y)=(x,3)
3. (x,y)=(x-d,y) if x>0
4. (x,y)=(x,y-d) if y>0
5. (x,y)=(0,y) if x>0
6. (x,y)=(x,0) if y>0
7. (x,y)= (4,y-(4-x)) if y>0
8. (x,y)=(x-(3-y),3) if x>0
9. (x,y)=(x+y,0) , if x+y<=4,y>0



8 PUZZLE PROBLEM

- The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame.
- One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell.

➢**The production system consists of**

- Production set
- Working Memory content
- Start State
- Goal State
- Control engine

Start state:

2	8	3
1	6	4
7		5

Goal state:

1	2	3
8		4
7	6	5

Production set:

Condition		Action
goal state in working memory	→	halt
blank is not on the left edge	→	move the blank left
blank is not on the top edge	→	move the blank up
blank is not on the right edge	→	move the blank right
blank is not on the bottomedge	→	move the blank down

Working memory is the present board state and goal state.

Control regime:

1. Try each production in order.
2. Do not allow loops.
3. Stop when goal is found.

SEARCH ALGORITHM TERMINOLOGIES

- **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 solution among all solutions.

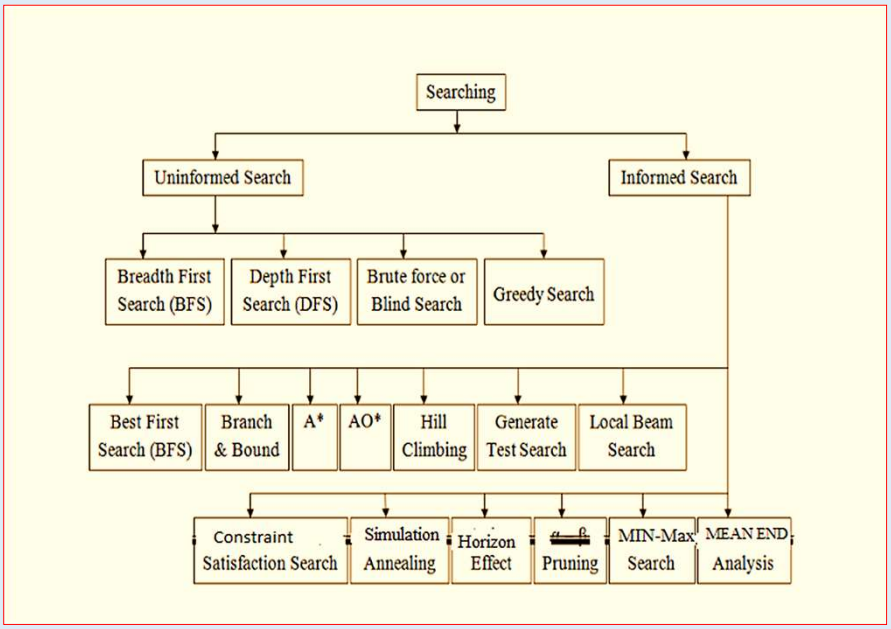
❖Properties of Search Algorithms

1. **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.
2. **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.
3. **Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.
4. **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.

❖Type of Search Strategies

- **Uninformed search** (Also known as **blind search**) –Uninformed search algorithms have no additional information on the goal node other than the one provided in the problem definition
- **Informed search** (Also known as **heuristic search**) – Search strategies know whether one state is more promising than another.

Uninformed searching	Informed searching
Search without information	Search with information
No knowledge	Use knowledge to find steps to solutions
Time consuming	Quick solutions
More complexity (time, space)	Less complexity (time, space)
DFS, BFS etc.	A*, Best first search etc.

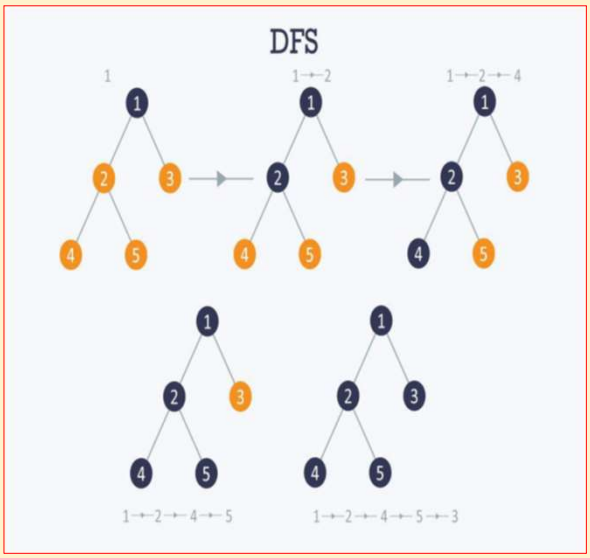


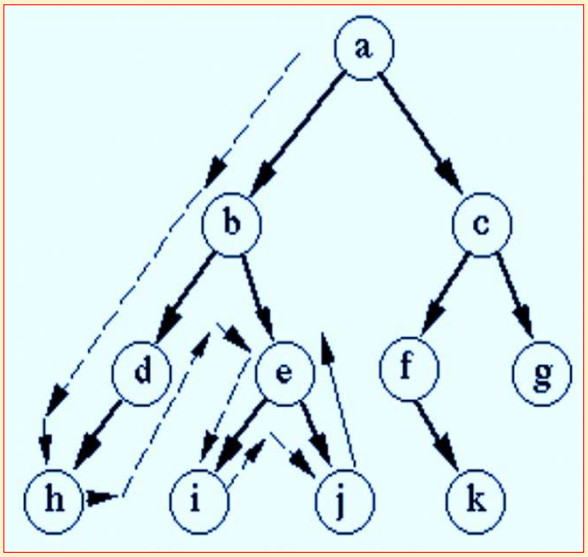
DEPTH FIRST SEARCH (DFS)

- The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.
- Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.
- This recursive nature of DFS can be implemented using **stacks**.

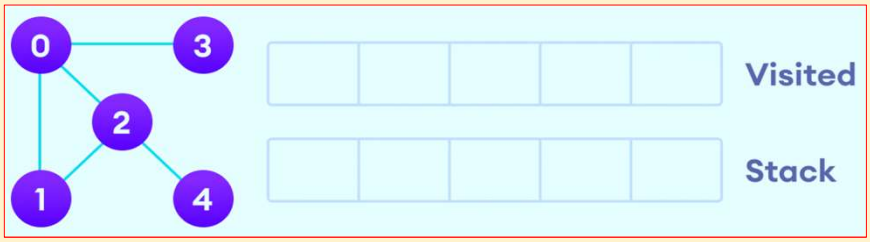
❖The basic idea is as follows:

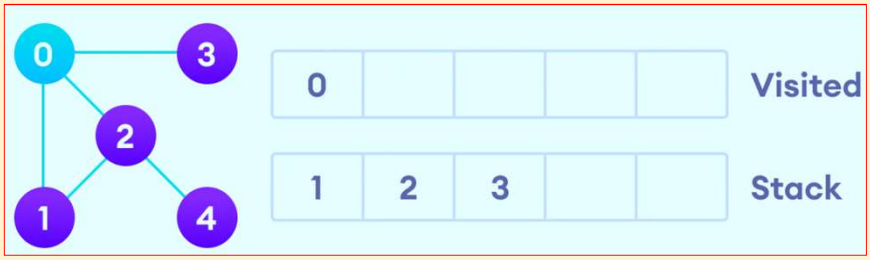
- ➢Pick a starting node and push all its adjacent nodes into a stack.
- ➢Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
- ➢Repeat this process until the stack is empty.
- However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once.
- If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.



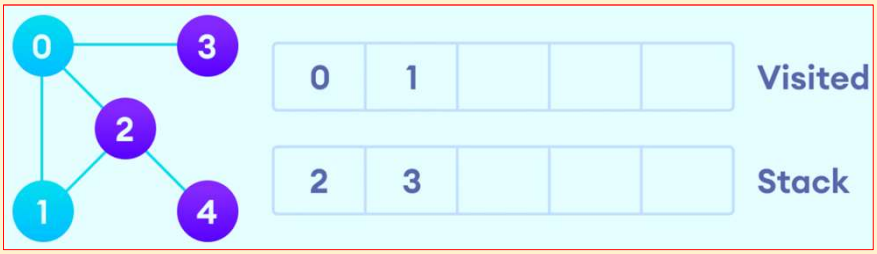


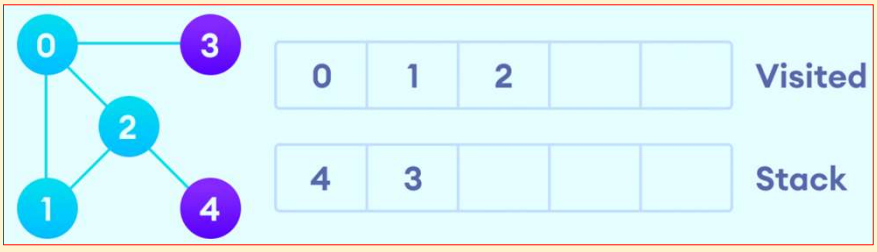
EXAMPLE



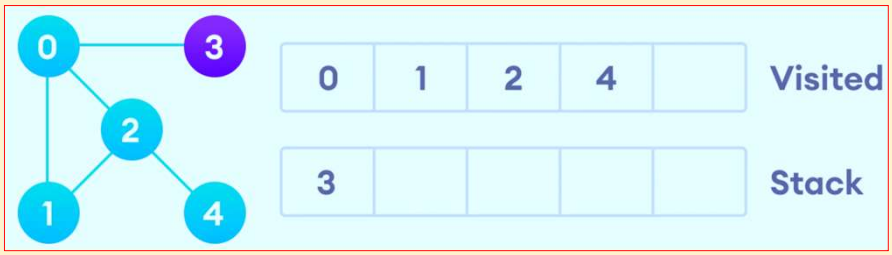


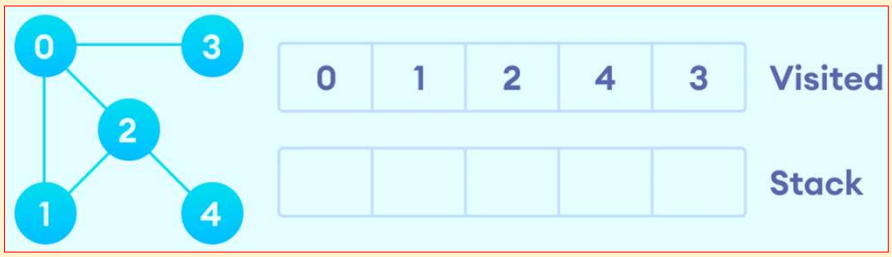
Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 33





Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 34





- DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

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

$$T(n)= 1+ n^2+ n^3 +.....+ n^m= \mathbf{O(n^m)}$$

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

- Not guaranteed that DFS will give you solution.

- **Completeness:** DFS will provide completeness feature when state space is finite.

BREADTH FIRST SEARCH (BFS)

- In Breadth First Search(BFS), the root node of the graph is expanded first, then all the successor nodes are expanded and then their successor and so on i.e. the nodes are expanded level wise starting at root level.
- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- BFS algorithm starts searching from the root node of the tree and expands all successor nodes at the current level before moving to nodes of next level.
- Breadth-first search implemented using FIFO queue data structure.

EXAMPLE

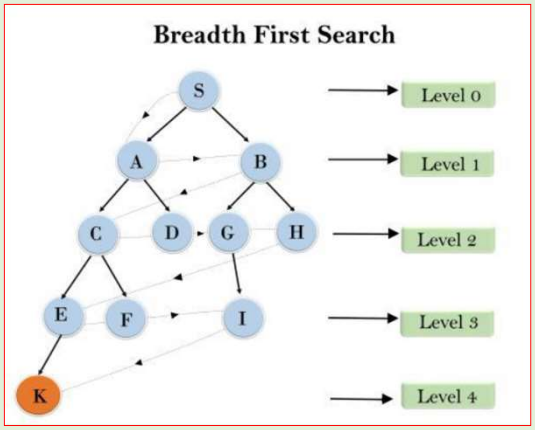
S= source node

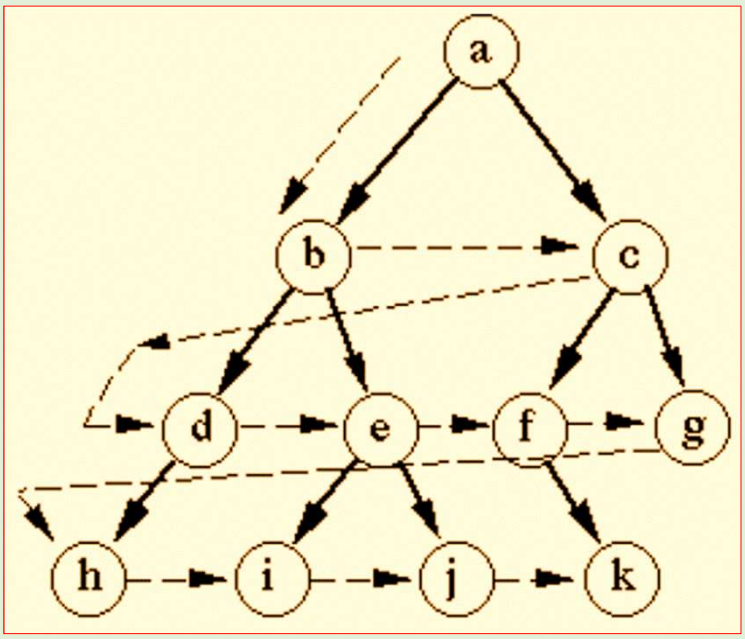
K= destination node

Path will be according to BFS concept.

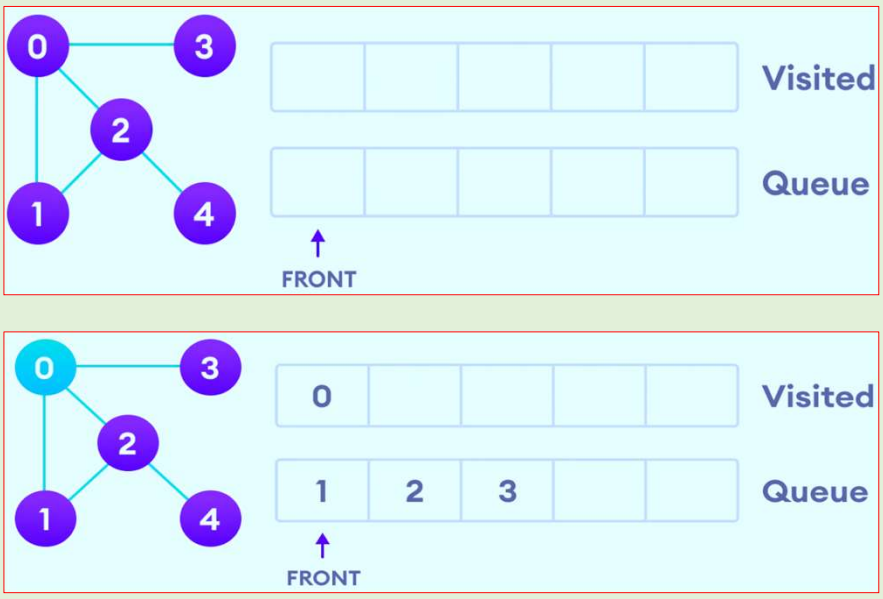
S---> A--->B---->C--->D---->G--->H--->

E---->F---->I ---->K

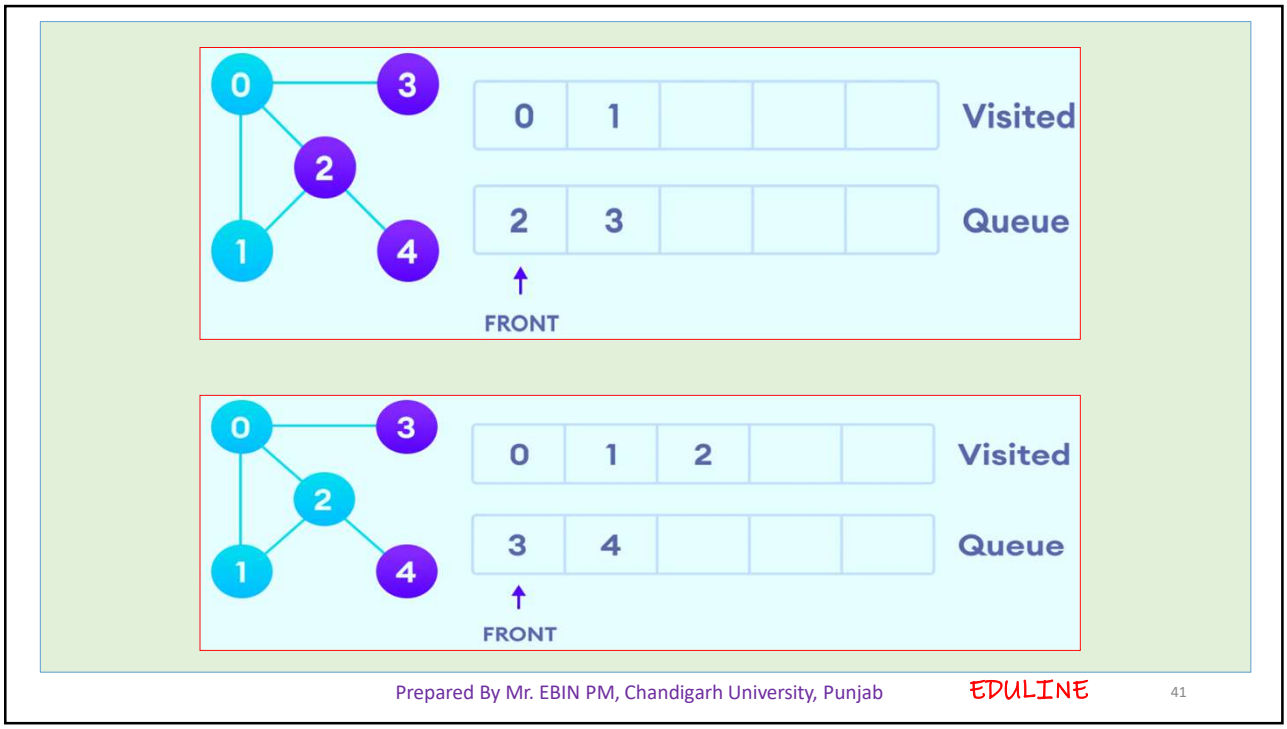


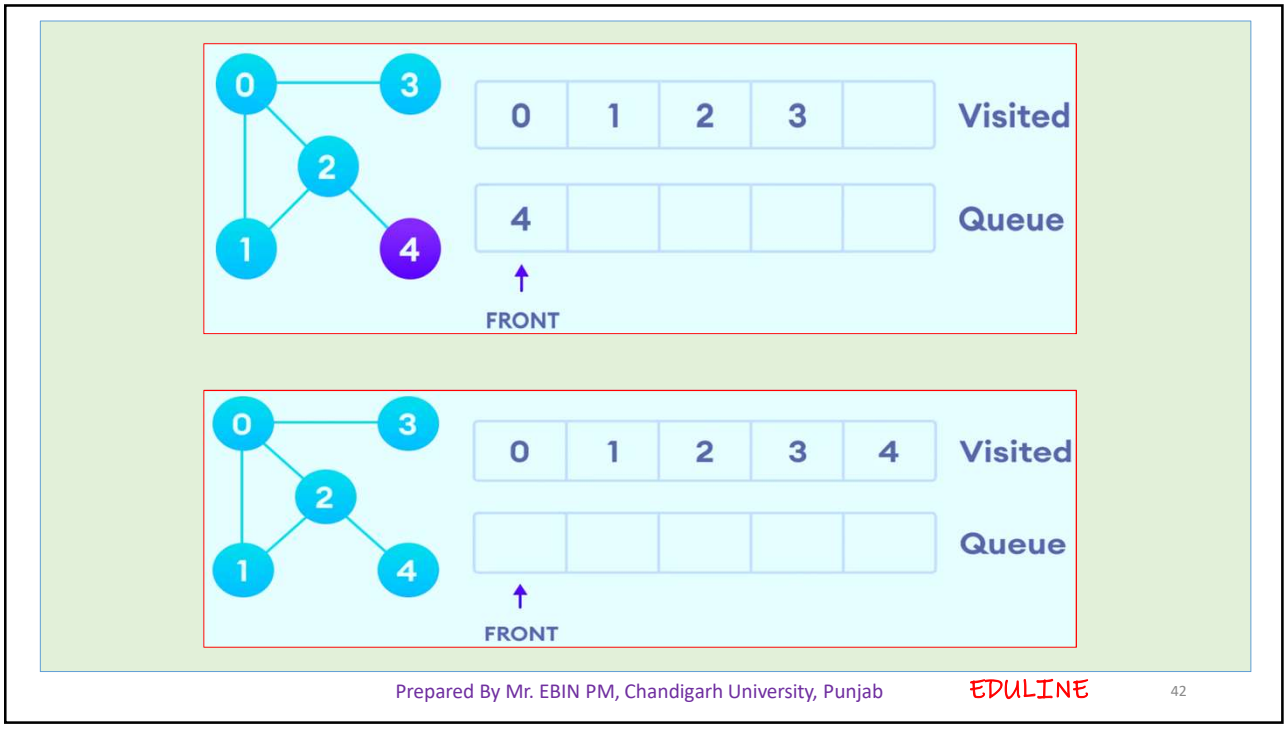


Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 39



Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 40


0
3
2
1
4
0
1
Visited
2
3
Queue
FRONT
0
3
2
1
4
0
1
2
Visited
3
4
Queue
FRONT
Prepared By Mr. EBIN PM, Chandigarh University, Punjab
EDULINE
41


0
3
2
1
4
0
1
2
3
Visited
4
Queue
FRONT
0
3
2
1
4
0
1
2
3
4
Visited
Queue
FRONT
Prepared By Mr. EBIN PM, Chandigarh University, Punjab
EDULINE
42

- **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.

$$T(b) = 1+b^2+b^3+.....+ b^d = \mathbf{O(b^d)}$$

- In this procedure at any way it will find the goal.
- There is nothing like useless path in BFS, since it searches level by level.
- BFS consumes large memory space. Its time complexity is more.

ITERATIVE DEEPENING DFS

- 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.

❖**Advantages:**

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

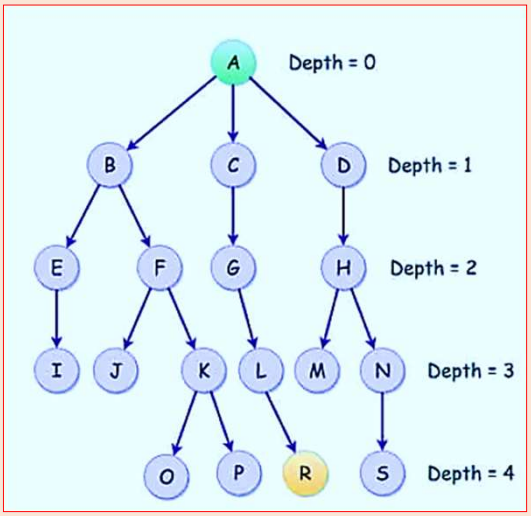
❖**Disadvantages:**

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

➢If b is the branching factor, and d is the depth of the goal node or the depth at which the iteration of IDDFS function terminates, the time complexity is $\mathbf{O(b^d)}$

Example 1

- Here in the given tree, the starting node is A and the depth initialized to 0.
- The goal node is R where we have to find the depth and the path to reach it. The depth from the figure is 4.
- In this example, we consider the tree as a finite tree, while we can consider the same procedure for the infinite tree as well.
- We knew that in the algorithm of IDDFS we first do DFS till a specified depth and then increase the depth at each loop.
- This special step forms the part of DLS or Depth Limited Search. Thus the following traversal shows the IDDFS search.



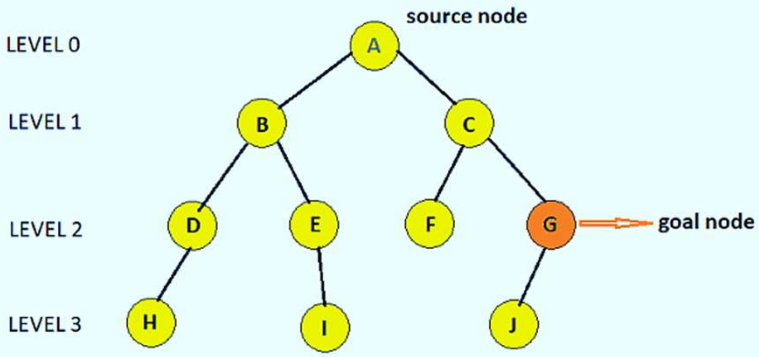
The tree can be visited as: A B E F C G D H

DEPTH = {0, 1, 2, 3, 4}

DEPTH LIMITS	IDDFS
0	A
1	A B C D
2	A B E F C G D H
3	A B E I F J K C G L D H M N
4	A B E I F J K O P C G L R D H M N S

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 47

EXAMPLE 2



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

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 48

INFORMED SEARCH ALGORITHMS

- So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space.
- But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called **Heuristic search**.

❖Heuristics function

- Heuristic is a function which is used in Informed Search, and it finds the most promising path.
- It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- Heuristic function estimates how close a state is to the goal. It is represented by **h(n)**, and it calculates the cost of an optimal path between the pair of states.
- The value of the heuristic function is always positive.

h(n) <= h*(n)

- Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

❖Pure Heuristic Search

- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

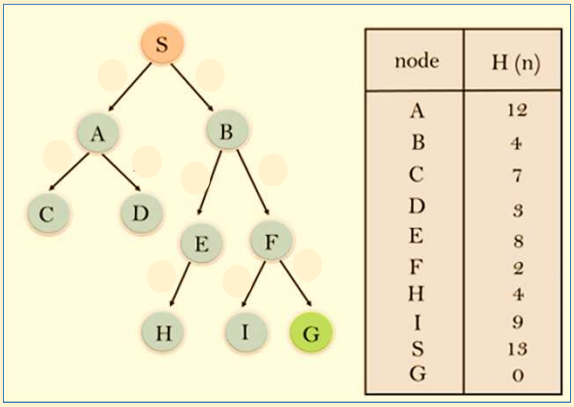
❖Best-first Search Algorithm (Greedy Search)

- Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

h(n)= g(n)

- Were, h(n)= estimated cost from node n to the goal.

EXAMPLE

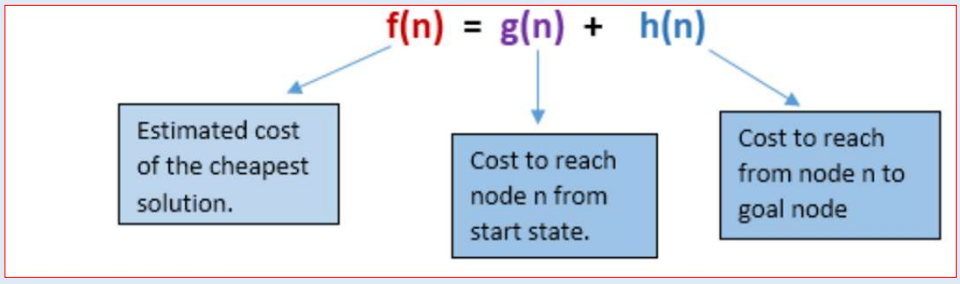


node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

- **Initialization:** Open [A, B], Closed [S]
- **Iteration 1:** Open [A], Closed [S, B]
- **Iteration2:** Open [E, F, A], Closed [S, B]
  : Open [E, A], Closed [S, B, F]
- **Iteration 3:** Open [I, G, E, A], Closed [S, B, F]
  : Open [I, E, A], Closed [S, B, F, G]
- Hence the final solution path will be: **S----> B----->F----> G**

A* ALGORITHM

- It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster.
- A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).
- In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



➢Advantages

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

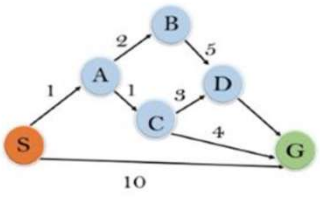
➢Disadvantages

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Algorithm of A* search:

- **Step1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- **Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
- **Step 6:** Return to **Step 2**.

Example



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

- **Initialization:** {(S, 5)}
- **Iteration1:** {(S--> A, 4), (S-->G, 10)}
- **Iteration2:** {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}
- **Iteration3:** {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}
- **Iteration 4** will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

- The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula :

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

where g(n) is the cost to reach any node from start state.