Parul® University

Artificial Intelligence

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

Search techniques

Parul® University NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Outline

- BFS and DFS
- Generate-And-Test
- Hill Climbing
- Best-First Search
- A* Algorithm
- Problem Reduction
- AO* Algorithm
- Constraint Satisfaction
- Means-Ends Analysis

Introduction

- Search Techniques can be classified as:

1. Uninformed/Blind Search Control Strategy:
   - Do not have additional information about states beyond problem definition.
   - Total search space is looked for the solution.
   - Example: Breadth First Search (BFS), Depth First Search (DFS), Depth Limited Search (DLS).

2. Informed/Directed Search Control Strategy:
   - Some information about problem space is used to compute the preference among various possibilities for exploration and expansion.
   - Examples: Best First Search, Problem Decomposition, A*, Mean end Analysis

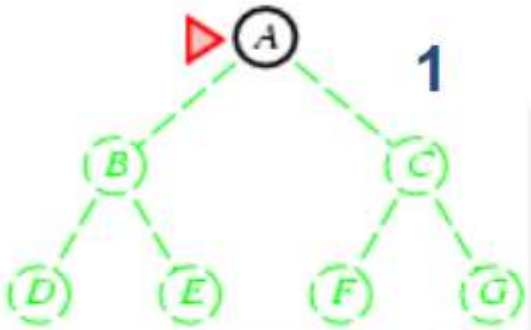
Parul® University

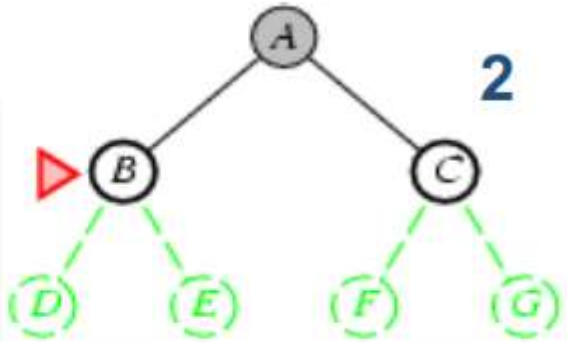
NAAC A++ ACCREDITED UNIVERSITY

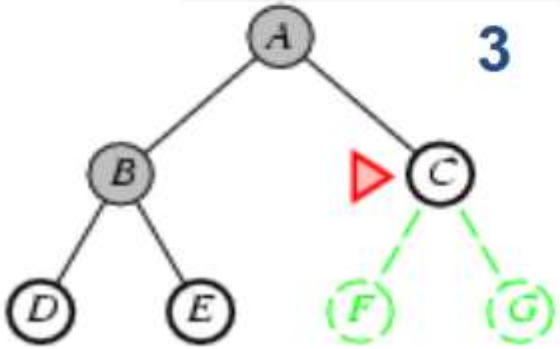
Uniformed Search Technique -BFS

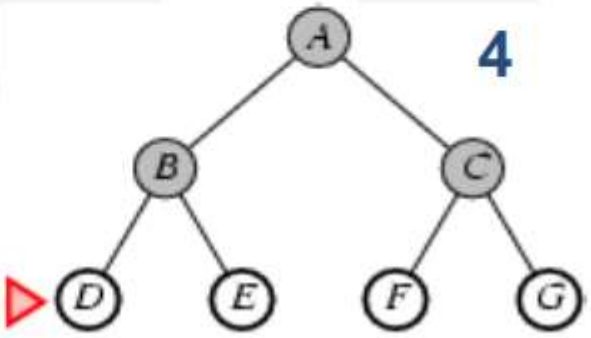
- FIFO – Queue
- Time Consuming
- Traverse Level by level
- Optimal Solution

Blind Search : Breadth First Search (BFS)







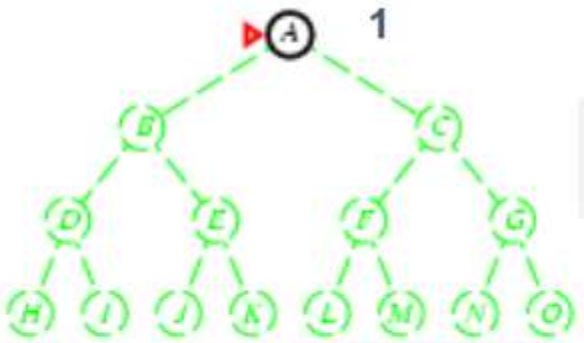


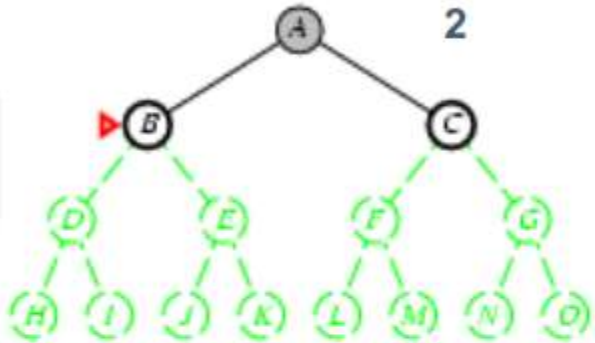
Parul® University NAAC A++ ACCREDITED UNIVERSITY

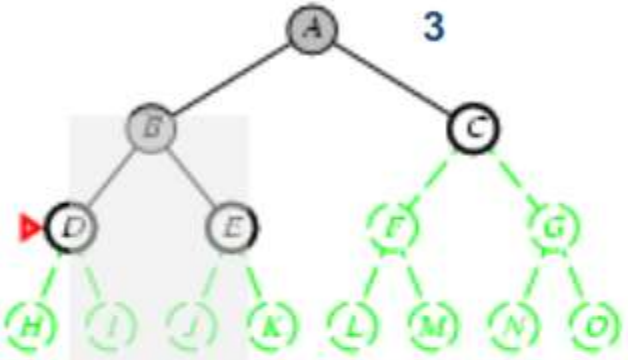
Uniformed Search Technique -DFS

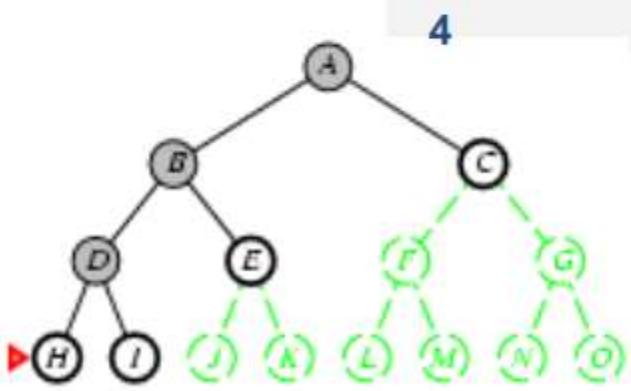
- LIFO – stack
- Quick Solution
- Traverse depth wise
- No optimal solution

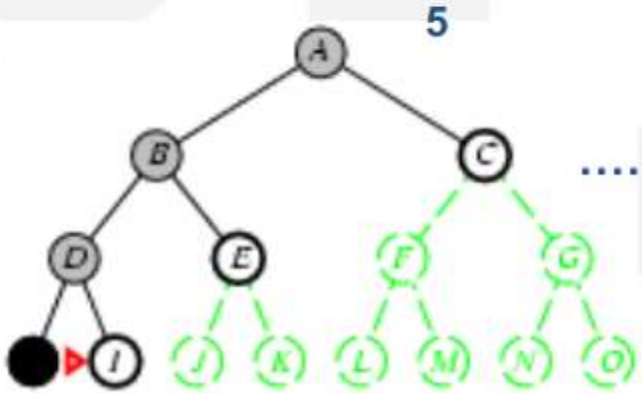
Blind Search : Depth First Search (DFS)

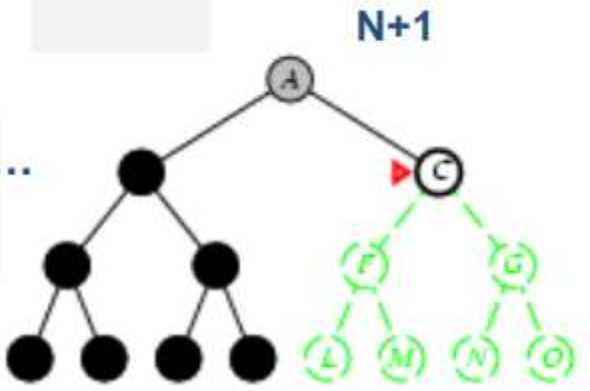












Parul® University NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Difference Between BFS and DFS

Depth First Search	Breath First Search
DFS requires less memory since only the nodes on the current path are stored.	BFS guarantees that the space of possible moves is systematically examined; this search requires considerably more memory resources.
By chance, DFS may find a solution without examining much of the search space at all. Then it finds a solution faster.	The search systematically proceeds testing each node that is reachable from a parent node before it expands to any child of those nodes.
If the selected path does not reach to the solution node, DFS gets stuck into a blind alley.	BFS will not get trapped exploring a blind alley.
Does not guarantee to find solution. Backtracking is required if wrong path is selected.	If there is a solution, BFS is guaranteed to find it.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Heuristic Search Technique

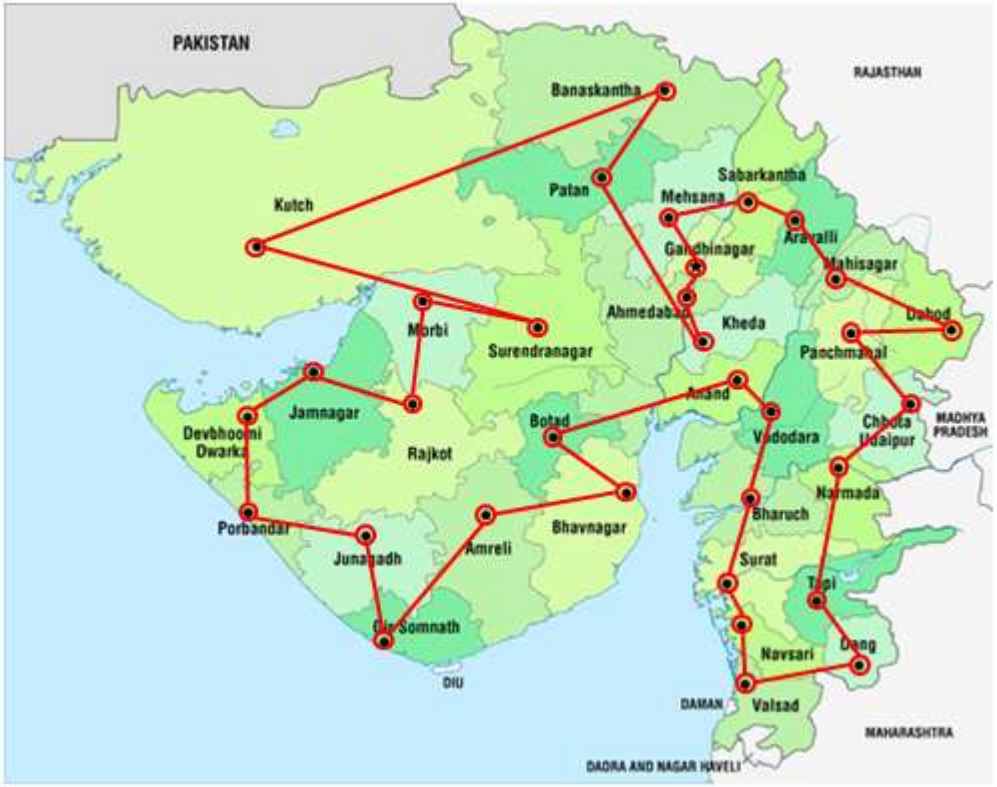
- Every search process can be viewed as a traversal of a directed graph, in which the nodes represent problem states and the arcs represent relationships between states.
- The search process must find a path through this graph, starting at an initial state and ending in one or more final states.
- Domain-specific knowledge must be added to improve search efficiency.
- The Domain-specific knowledge about the problem includes the nature of states, cost of transforming from one state to another, and characteristics of the goals.
- This information can often be expressed in the form of Heuristic Evaluation Function.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Visiting each District of Gujarat State



- Start with any random city.
- Go to the next nearest city.

Nearest Neighbor

↑

Heuristic

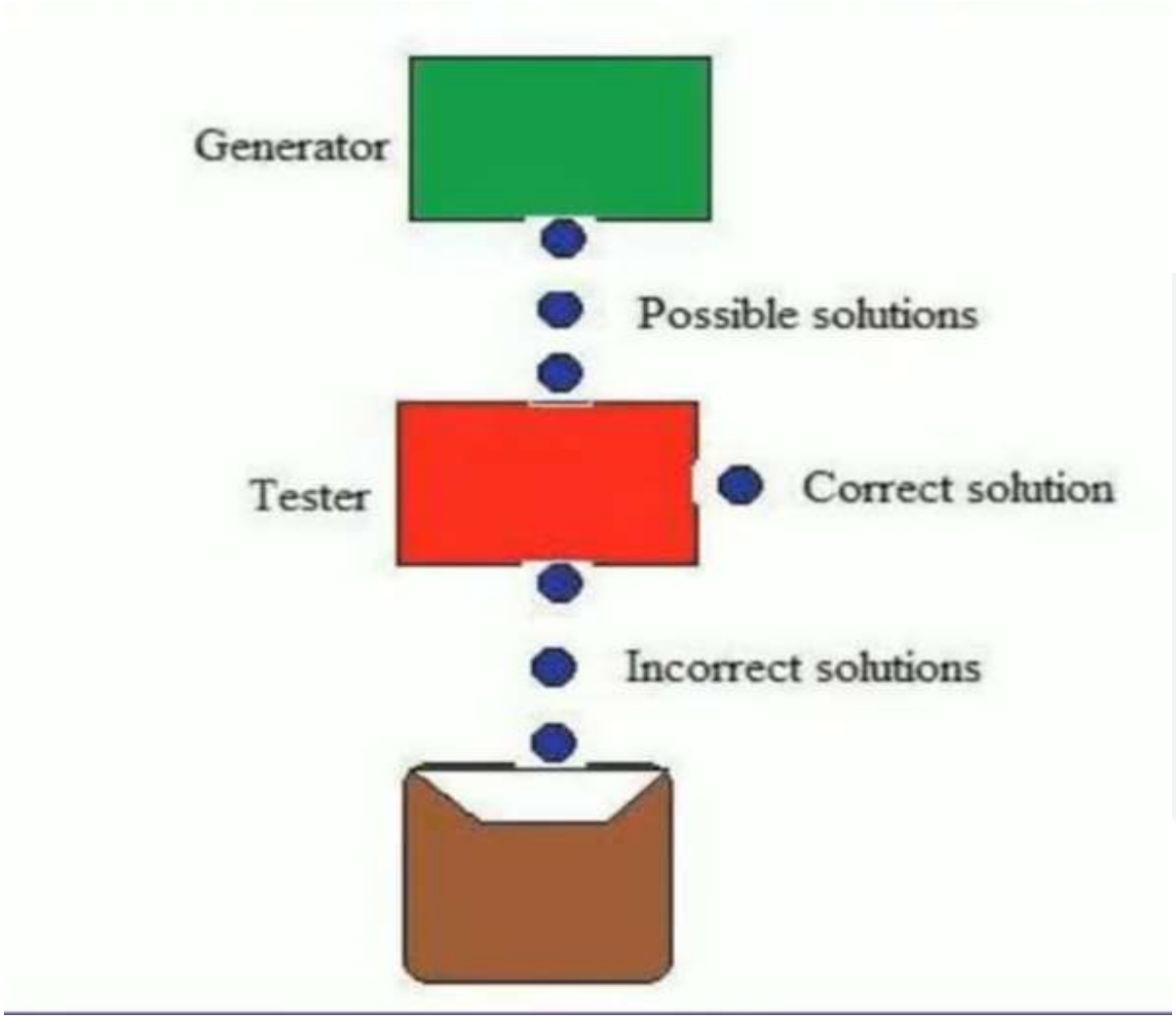
- Heuristic function maps from problem state descriptions to the measures of desirability, usually represented as numbers.
- The value of the heuristic function at a given node in the search process gives a good estimate of whether that node is on the desired path to a solution.
- Well-designed heuristic functions can play an important role in efficiently guiding a search process toward a solution.
- In general, heuristic search improves the quality of the path that is explored.
- In such problems, the search proceeds using current information about the problem to predict which path is closer to the goal and follow it, although it does not always guarantee to find the best possible solution.
- Such techniques help in finding a solution within reasonable time and space (memory).

Generate And Test Strategy

- Generate-and-test search algorithm is a very simple algorithm that guarantees to find a solution if done systematically and there exists a solution.

```
Algorithm: Generate-And-Test
1.Generate a possible solution.
2.Test to see if this is the expected solution.
3.If the solution has been found quit else go to step 1.
```

- Potential solutions that need to be generated vary depending on the kinds of problems.
- For some problems the possible solutions may be particular points in the problem space and for some problems, paths from the start state.


Generator
Possible solutions
Tester
Correct solution
Incorrect solutions

- To approaches are followed while generating solutions,
  - Generating complete solutions and
  - Generating random solutions

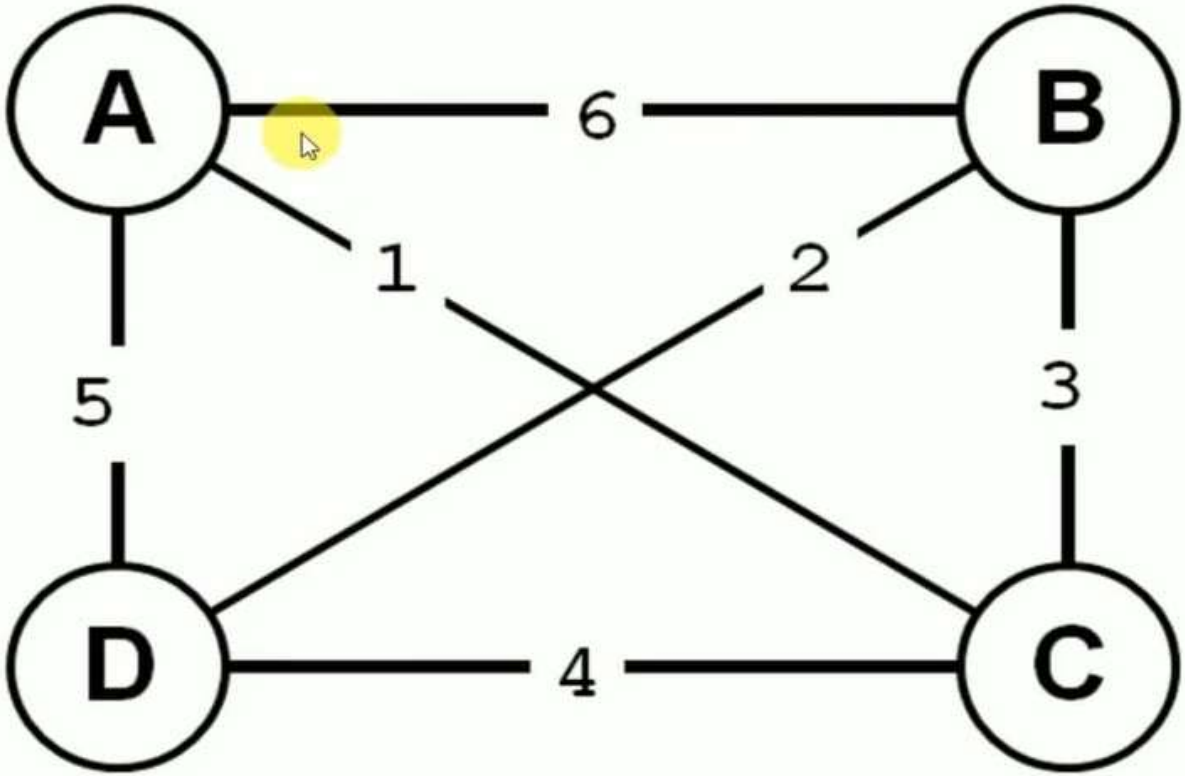
Generate-and-test Algorithm - Example

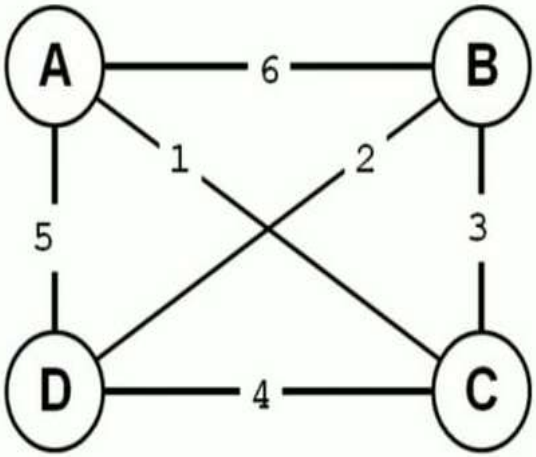
Example - Traveling Salesman Problem (TSP)

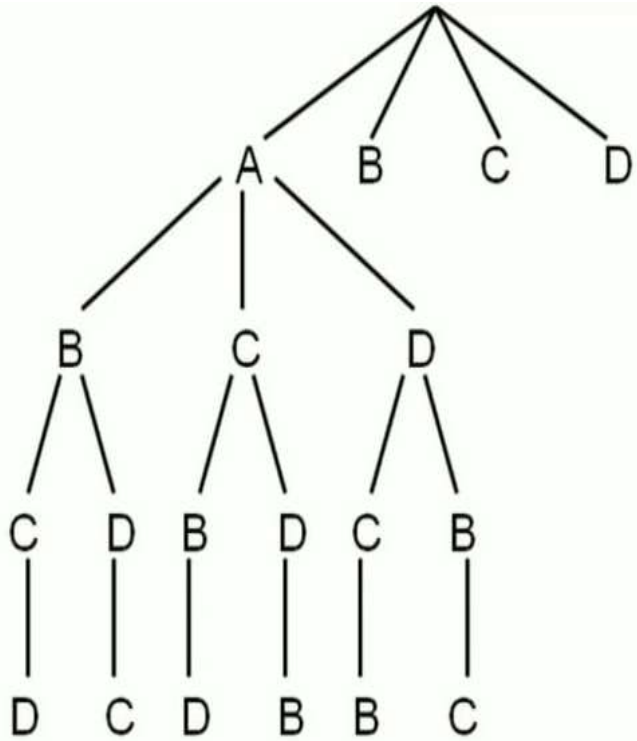
A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow for the shortest possible round trip that both starts and finishes at any one of the cities.

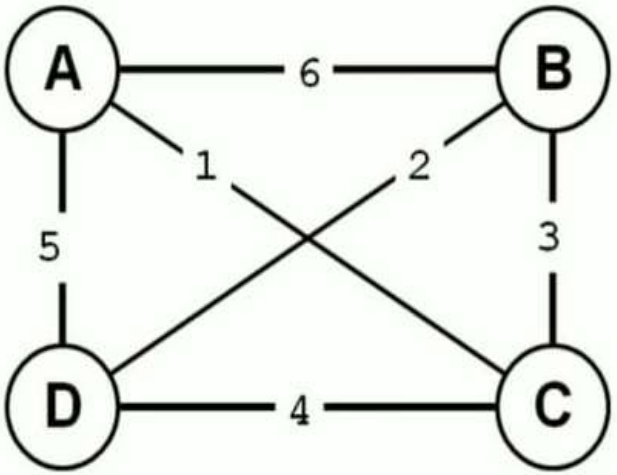
- Traveler needs to visit n cities.
- Know the distance between each pair of cities.
- Want to know the shortest route that visits all the cities once.

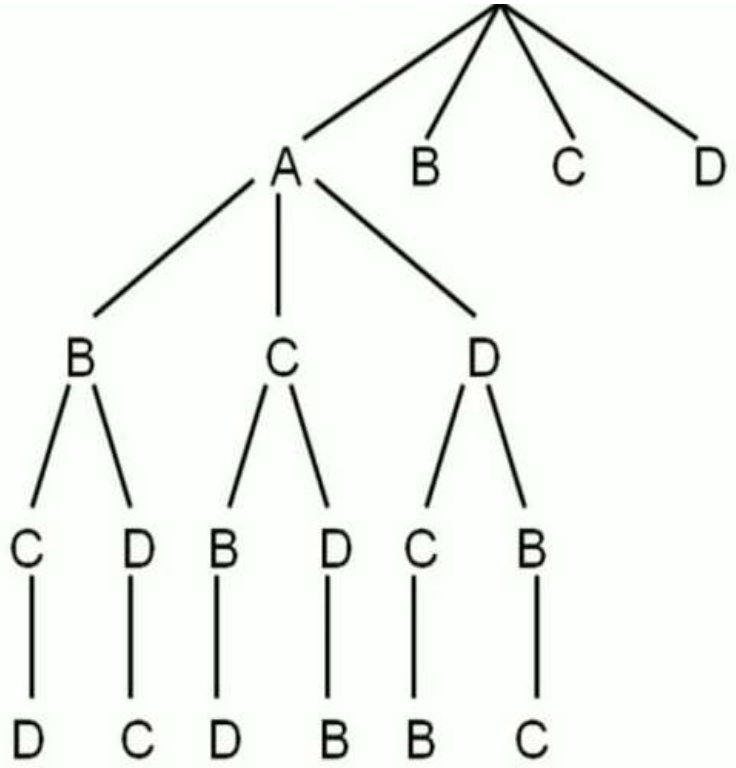
Generate-and-test Algorithm - Example




A
B
C
D
6
1
2
5
3
4


A
B
C
D
B
C
D
C
D
B
D
C
B
D
C
D
B
B
C





Search flow with Generate and Test

Search for	Path	Length of Path
1	ABCD	19
2	ABDC	18
3	ACBD	12
4	ACDB	13
5	ADBC	16
Dst.....		

Parul® University NAAC A++ ACCREDITED UNIVERSITY

Properties of Good Generators:

- ***Complete:*** *they should generate all the possible solutions*
- ***Non Redundant:*** *Good Generators should not yield a duplicate solution*
- ***Informed:*** *Good Generators have the knowledge about the search space*

Parul® University NAAC A++ ACCREDITED UNIVERSITY

Hill Climbing

•Basic of hill climbing is , you always go ahead when the next state Is better than the current one.(in terms of heuristic value)

•For example, smaller cost, smaller distance, more profit etc(you should be always in win condition)

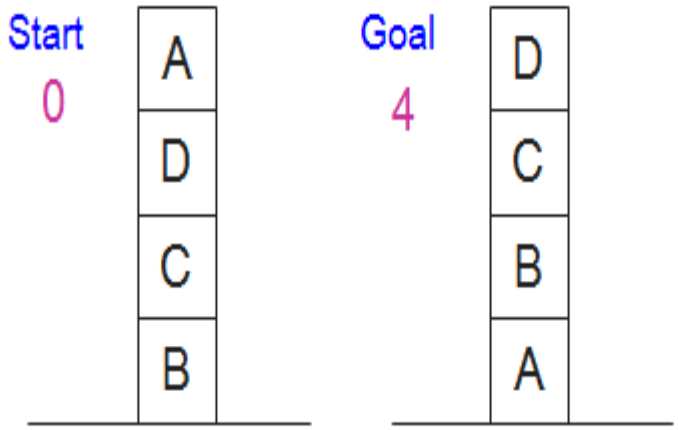
Parul® University NAAC A++ ACCREDITED UNIVERSITY

Simple Hill Climbing Algorithm

1. Evaluate the initial state. If it is also goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   b. Evaluate the new state
      i. If it is the goal state, then return it and quit.
      ii. If it is not a goal state but it is better than the current state, then make it the current state.
      iii. If it is not better than the current state, then continue in the loop.

Parul® University NAAC A++ ACCREDITED UNIVERSITY

Hill Climbing- Blocks World Problem

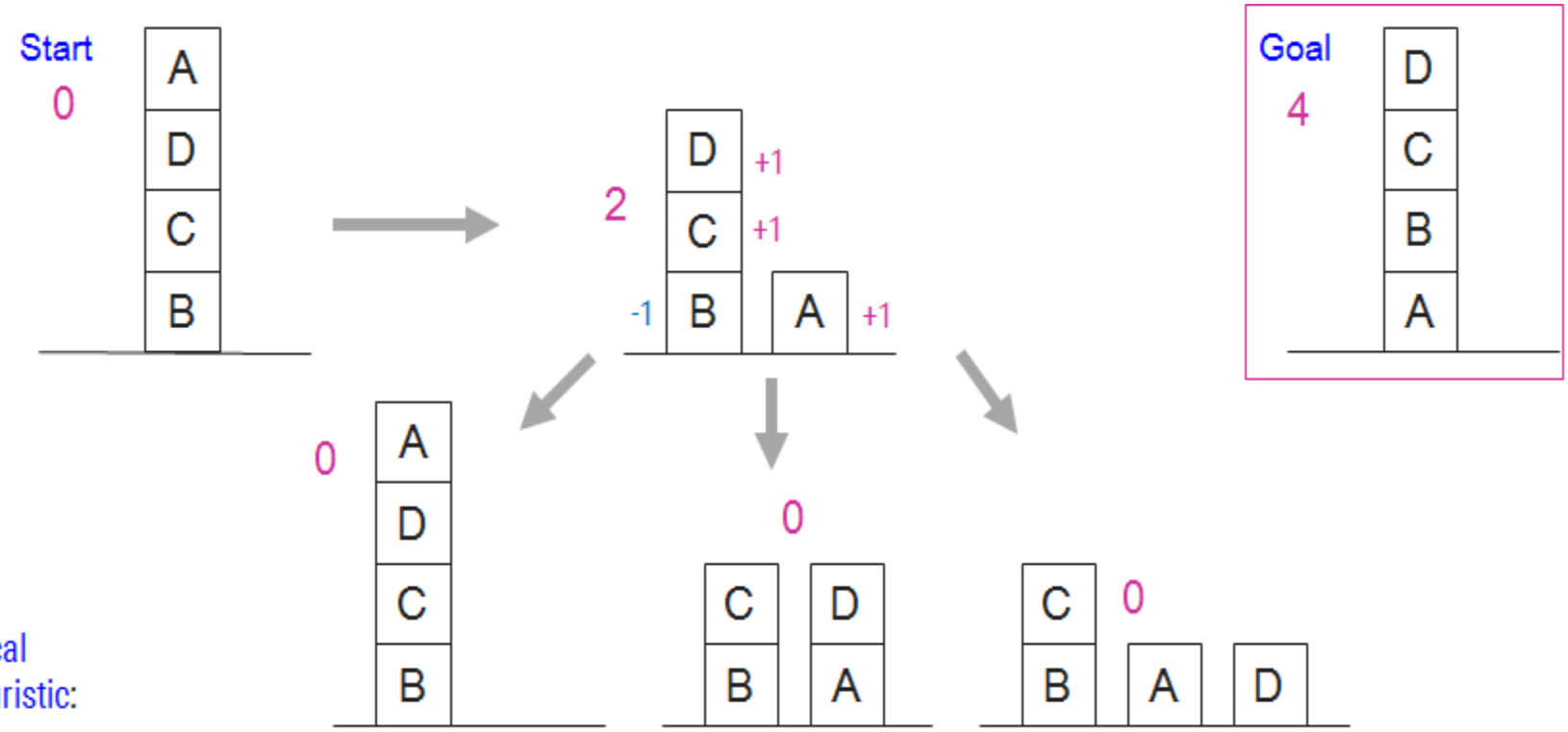


Local heuristic:

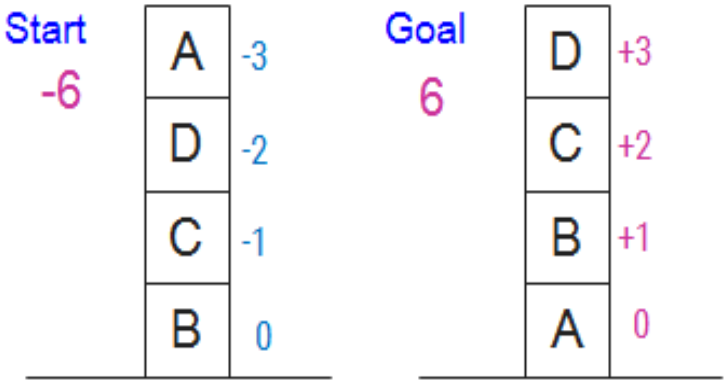
+1 for each block that is resting on the thing it is supposed to be resting on.

-1 for each block that is resting on a wrong thing.

Blocks World Problem



Blocks World Problem



Global heuristic:

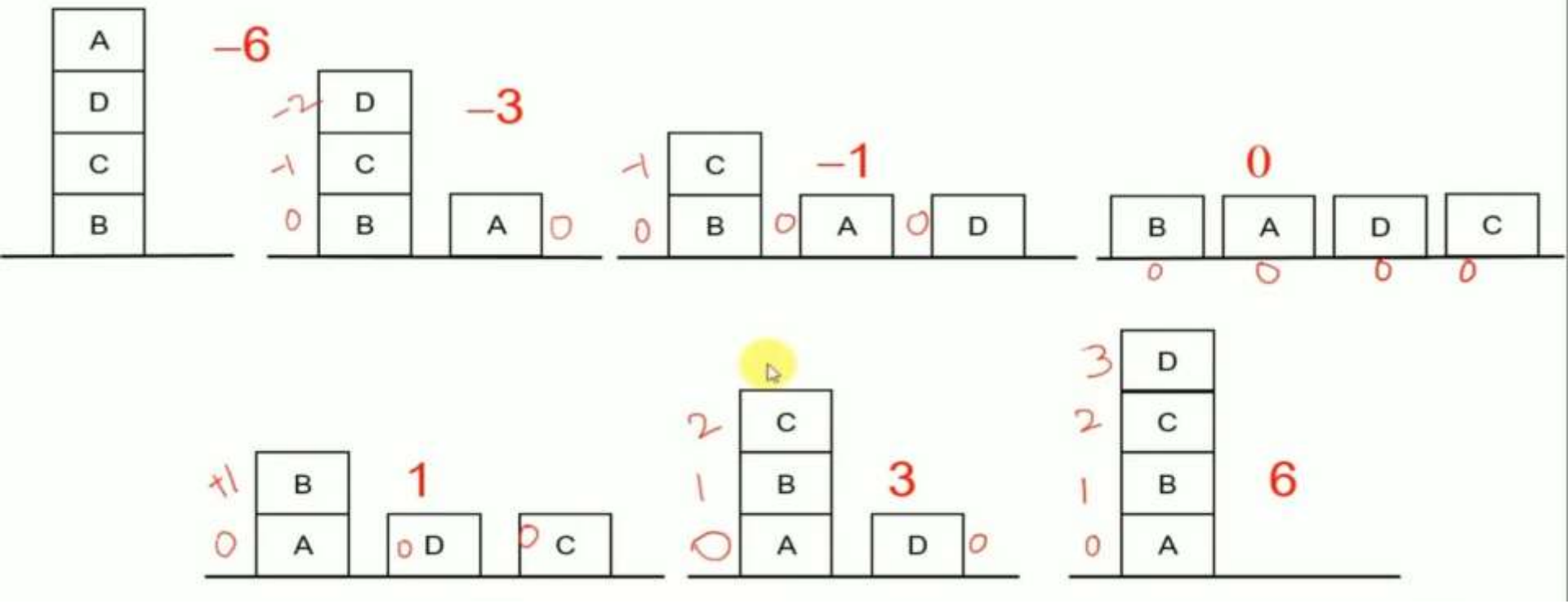
For each block that has the correct support structure:

+1 to every block in the support structure.

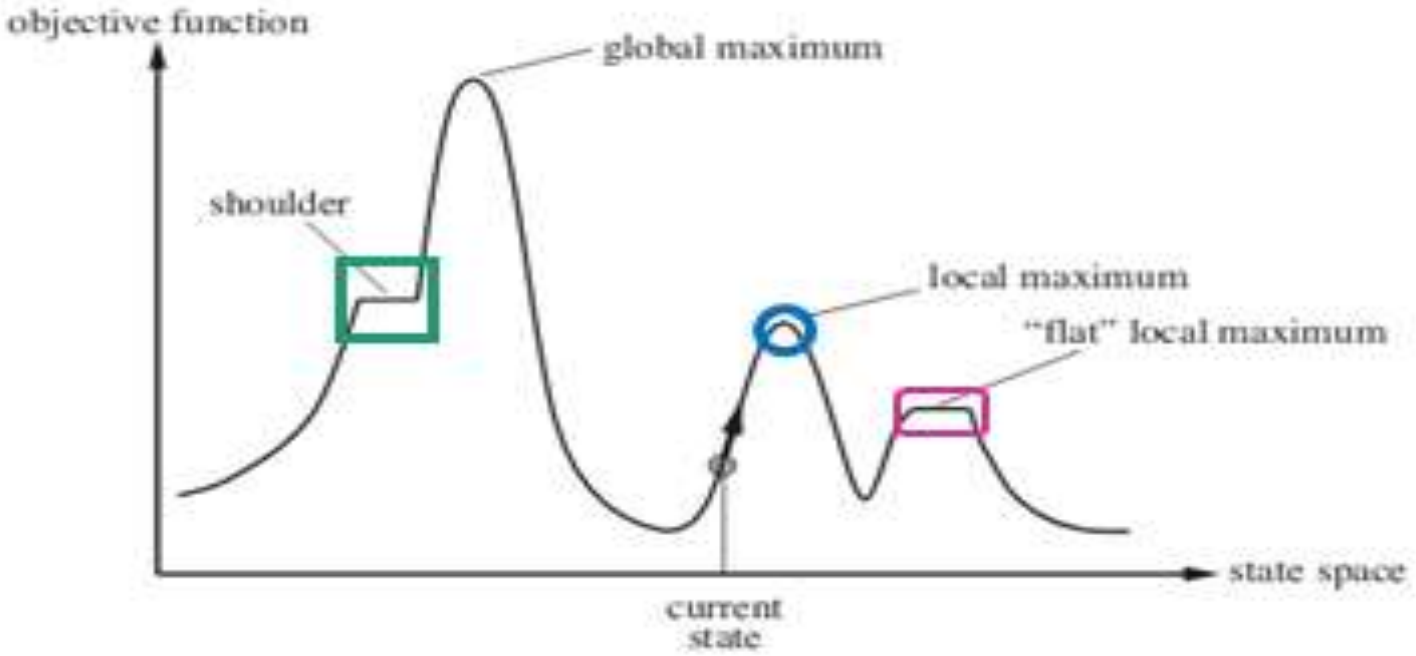
For each block that has a wrong support structure:

-1 to every block in the support structure.

Hill Climbing: Global Heuristic function



Drawbacks of Hill Climbing



- Local Maxima: a local maximum is a state that is better than all its neighbors but is not better than some other states further away.
- To overcome local maximum problem: Utilize backtracking technique. Maintain a list of visited states and explore a new path.

- Plateau: a plateau is a flat area of the search space in which, a whole set of neighboring states have the same values.
- To overcome plateaus: Make a big jump. Randomly select a state far away from current state.

- Ridge: is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has slop.
- To overcome Ridge: In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Steepest Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let S be a state such that any possible successor of the current state will be better than S.
   b. For each operator that applies to the current state do:
      - Apply the operator and generate a new state
      - Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to S. If it is better, then set S to this state. If it is not better, leave S alone.
   c. If the S is better than the current state, then set current state to S.

In simple hill climbing, the **first closer node** is chosen, whereas in steepest ascent hill climbing all successors are compared and the **closest to the solution** is chosen.

Parul® University NAAC A++ ACCREDITED UNIVERSITY

Advantages of Hill Climbing

- simple and intuitive algorithm that is easy to understand and implement.
- efficient in finding local optima good choice for problems where a good solution is needed quickly.
- The algorithm can be easily modified and extended

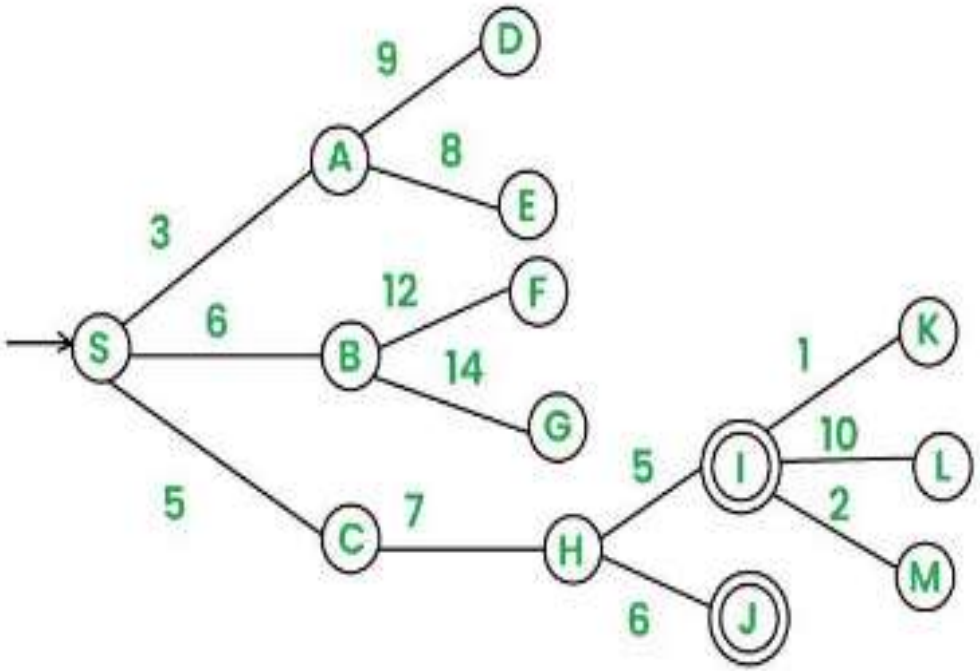
Disadvantages of Hill Climbing

- It can get stuck in local maxima, and may never find the global maxima of the problem.
- a poor initial solution may result in a poor final solution.
- does not explore the search space very thoroughly
- It may be less effective than other optimization algorithms, such as genetic algorithms or simulated annealing, for certain types of problems.

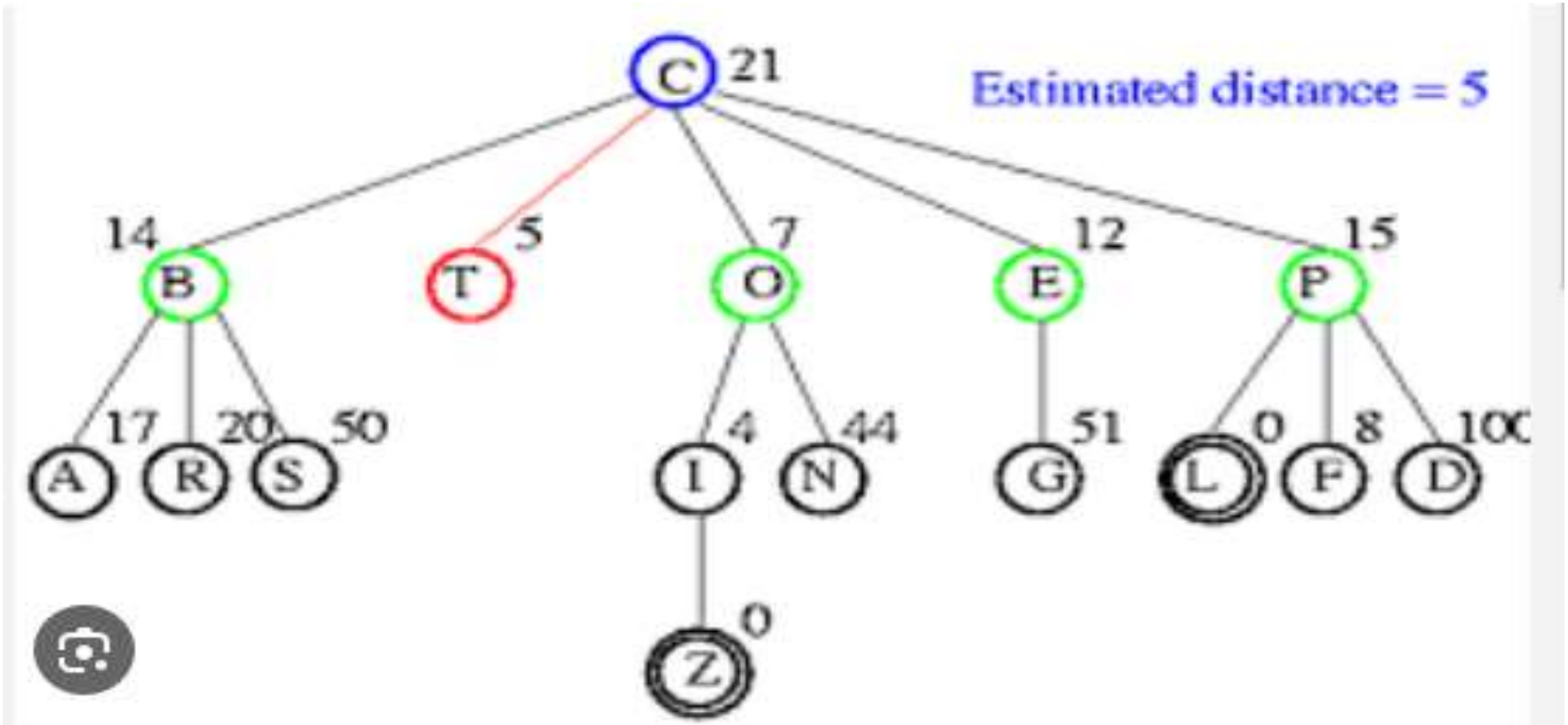
Best First Search

- DFS is good because it allows a solution to be found without expanding all competing branches. BFS is good because it does not get trapped on dead end paths.
- Best first search combines the advantages of both DFS and BFS into a single method.
- One way of combining BFS and DFS is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.
- At each step of the Best First Search process, we select the most promising of the nodes we have generated so far.
- This is done by applying an appropriate heuristic function to each of them.
- We then expand the chosen node by using the rules to generate its successors.
- If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far.

Example



Example


C 21
Estimated distance = 5
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Algorithm- Best First Search

1. Start with *OPEN* containing just the initial state.

2. Until a goal is found or there are no nodes left on *OPEN* do:

(a) Pick them best node on *OPEN*.
(b) Generate its successors.
(c) For each successor do:

(i) If it has not been generated before, evaluate it, add it to *OPEN*, and record its parent.
(ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already, have.

An **informed search strategy** uses problem-specific knowledge beyond the definition of the problem itself, so it can find solutions more efficiently than an uninformed strategy. **Best-first search** is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function, f (n) .**

• A key component of these algorithms is a heuristic function denoted *h(n) :*

h(n) = estimated cost of the cheapest path from node n to a goal node, *if n is a goal node, then h(n) = 0.*

- Advantages:
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.
- Disadvantages:
- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.
- **Time Complexity:** The worst case time complexity of Greedy best first search is $O(b^m)$.
- **Space Complexity:** The worst case space complexity of Greedy best first search is $O(b^m)$.
- Where b is the number of legal moves at each point and m is the maximum depth of the tree.

Parul® University NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

A* Algorithm

- A* search algorithm is informed search algorithm.
- Used to find the optimal path from the initial state to the goal state.
- A* search algorithm evaluates nodes by using the **function**,

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

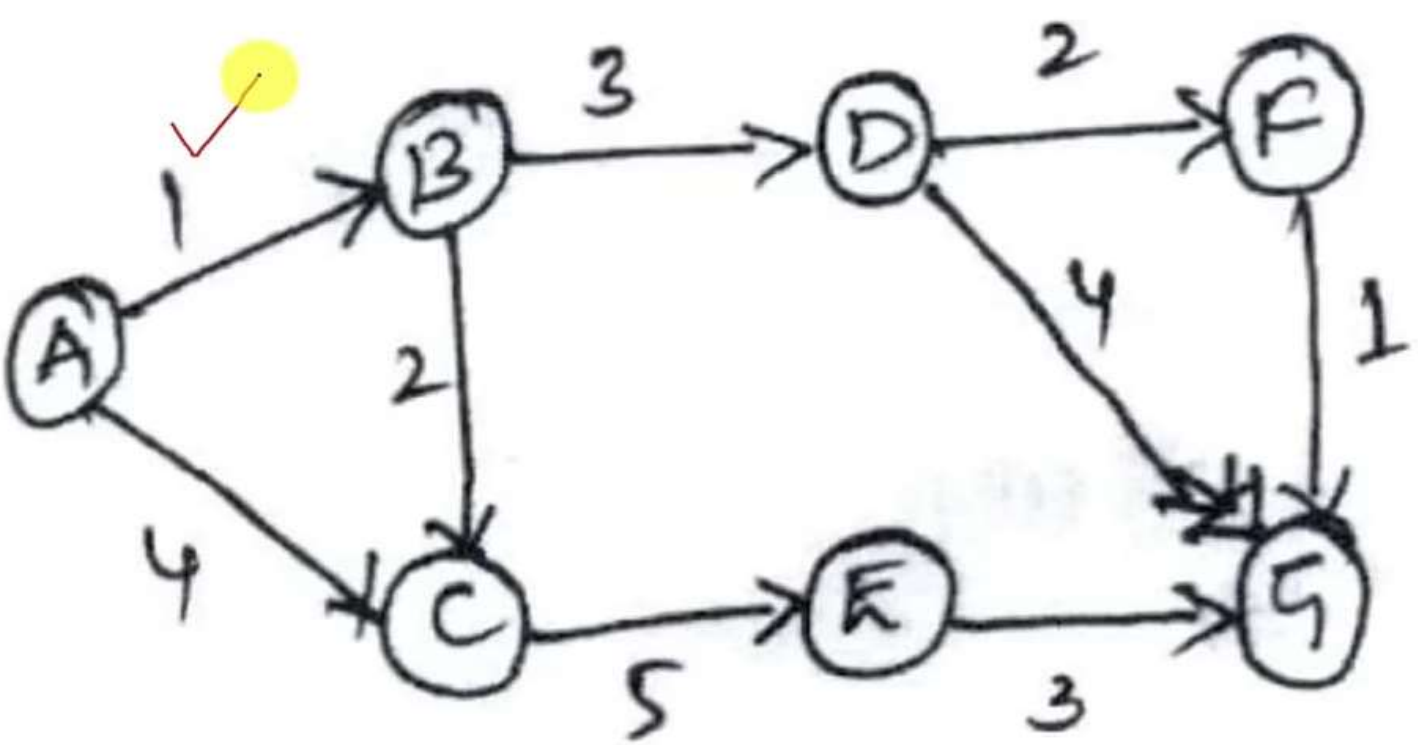
- g(n) = Cost from initial state to the state at the current node n
- h(n) = Estimated cost from the state at node n to a goal state

Parul® University

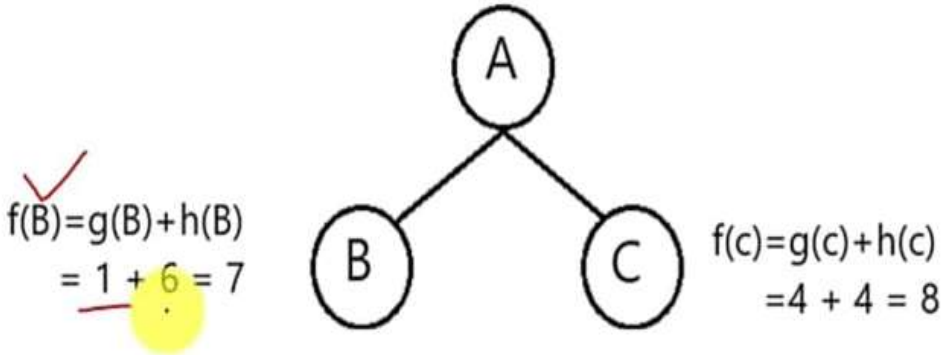
NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

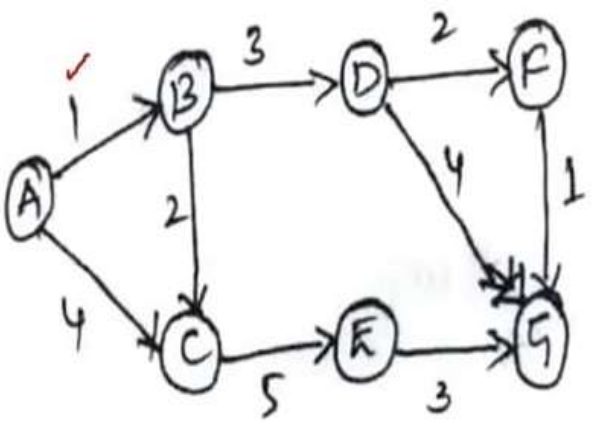
A* Algorithm – Example 1



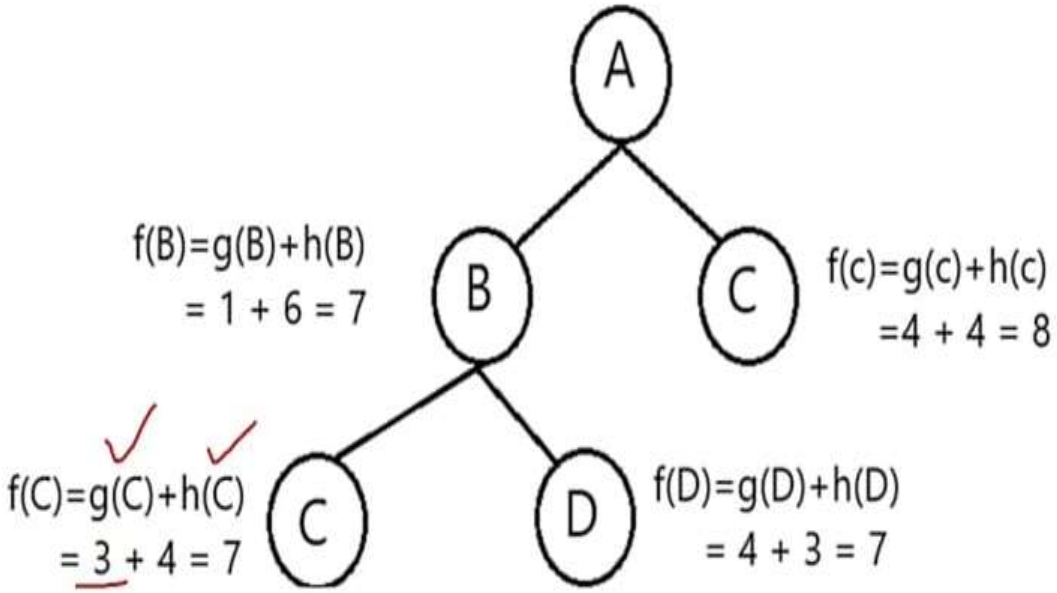
A	5
B	6
C	4
D	3
E	3
F	1
G	0



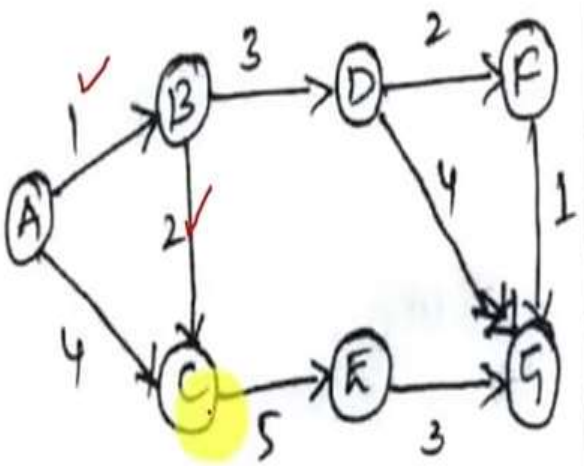
$$f(n) = g(n) + h(n)$$



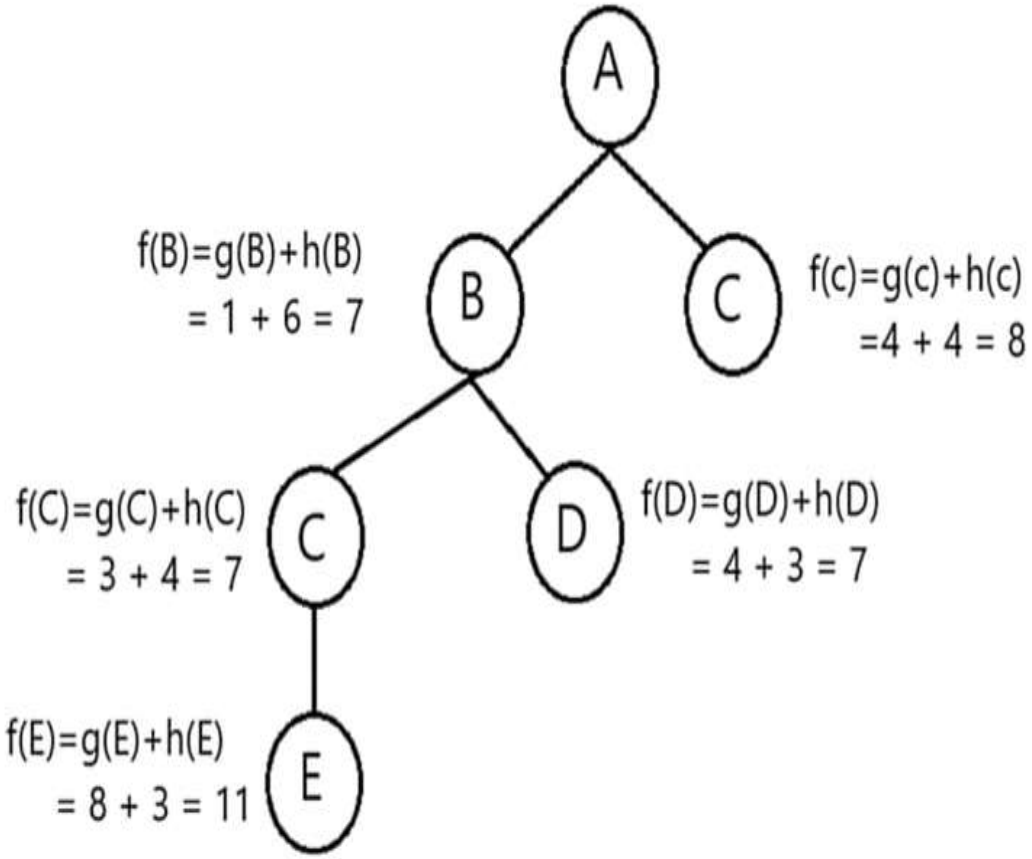
A	5
B	6
C	4
D	3
E	3
F	1
G	0



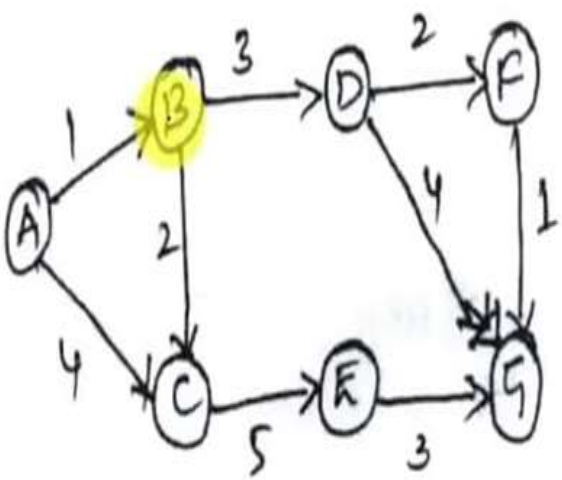
$$f(n) = g(n) + h(n)$$



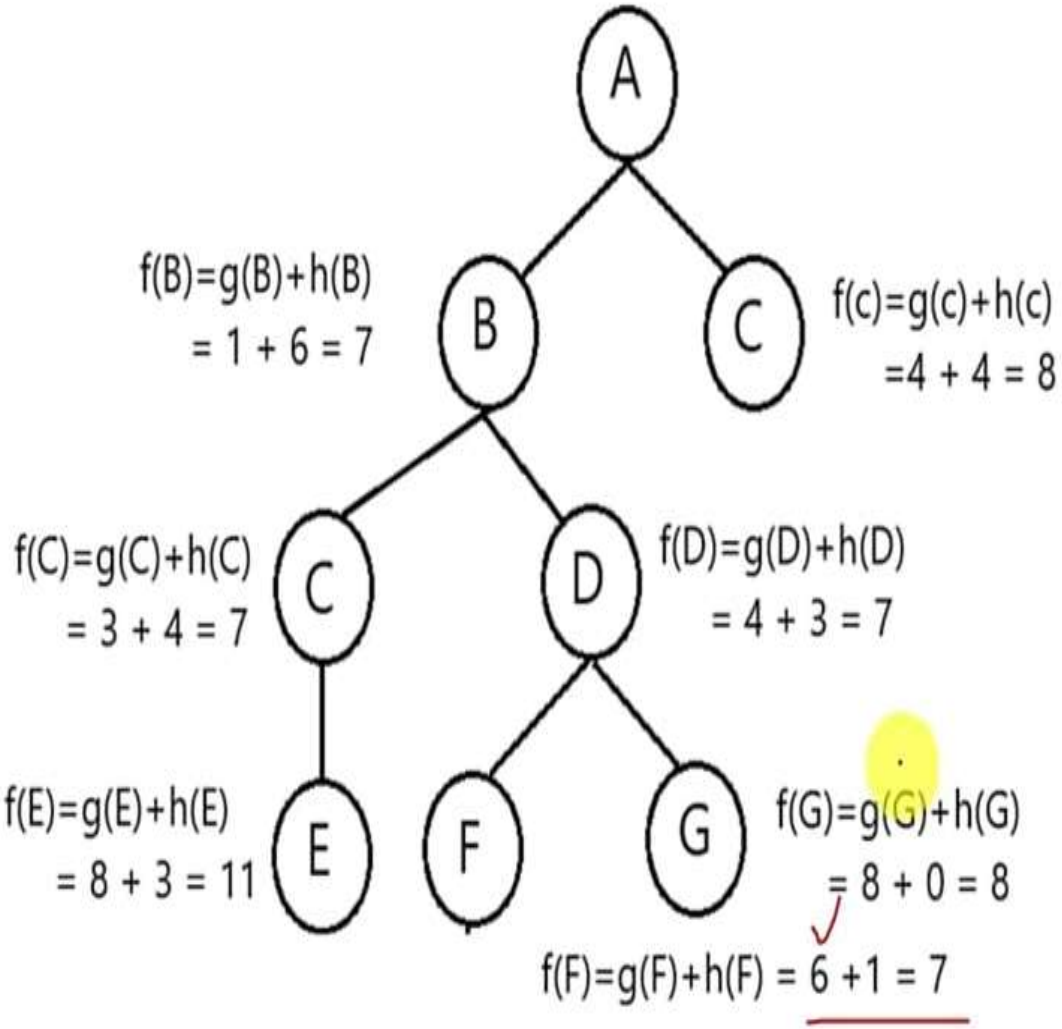
A	5
B	6
C	4
D	3
E	3
F	1
G	0



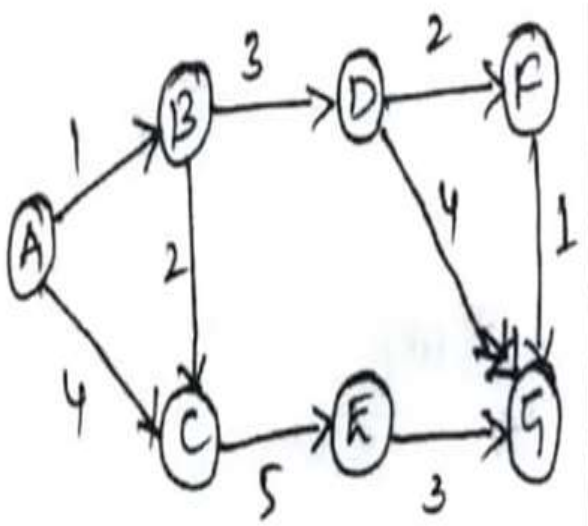
$$f(n) = g(n) + h(n)$$



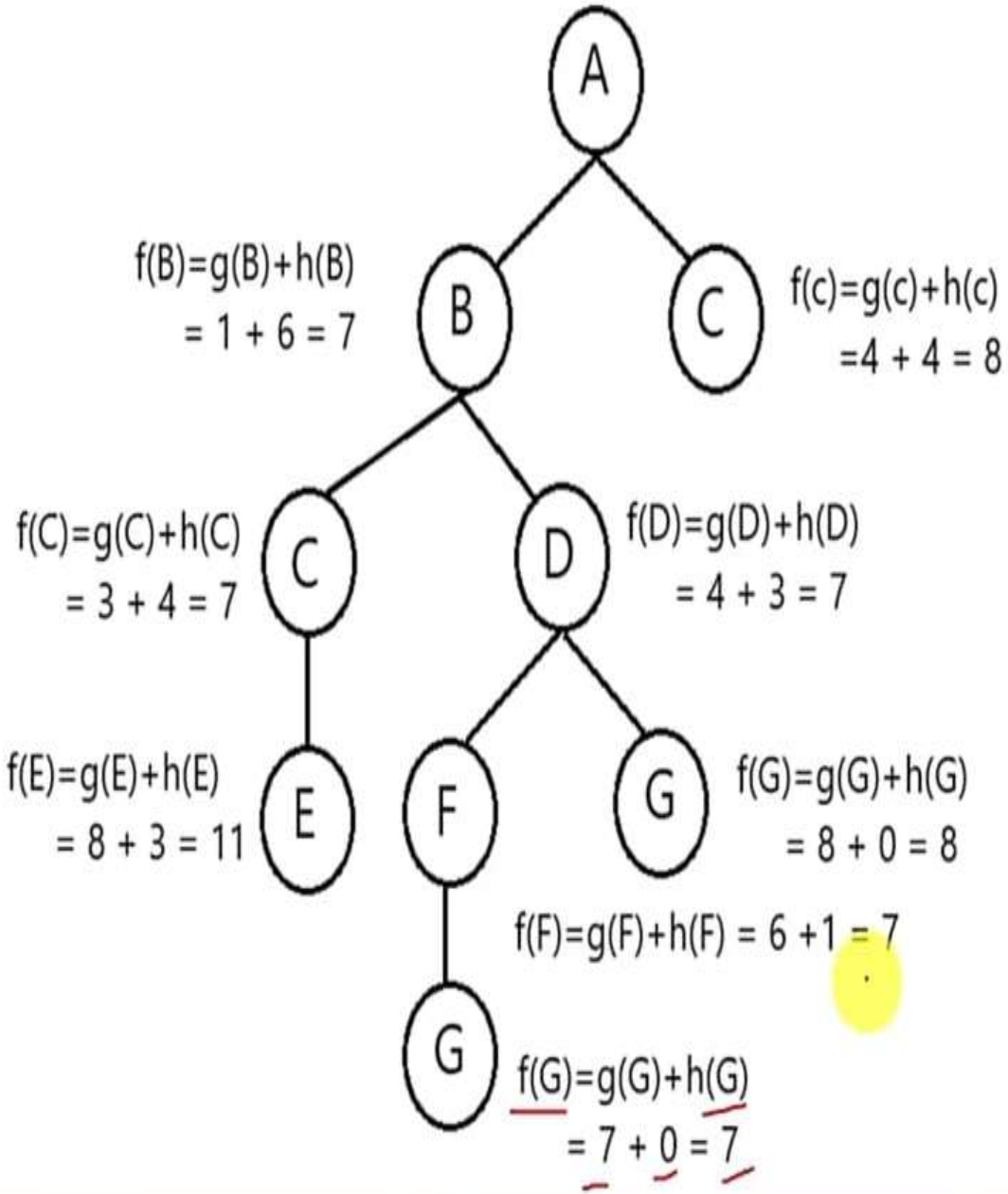
A	5
B	6
C	4
D	3
E	3
F	1
G	0



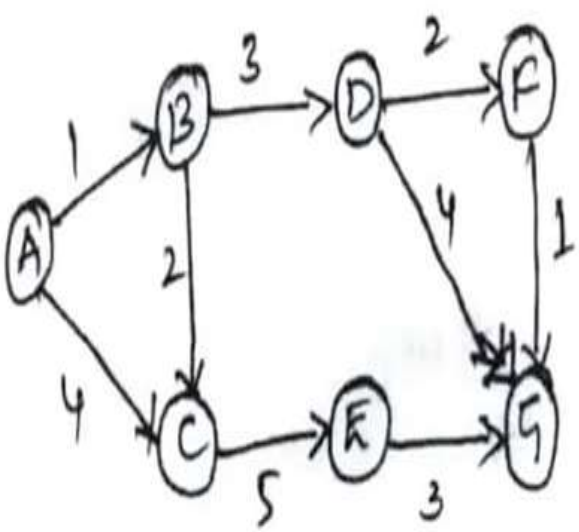
$$f(n) = g(n) + h(n)$$



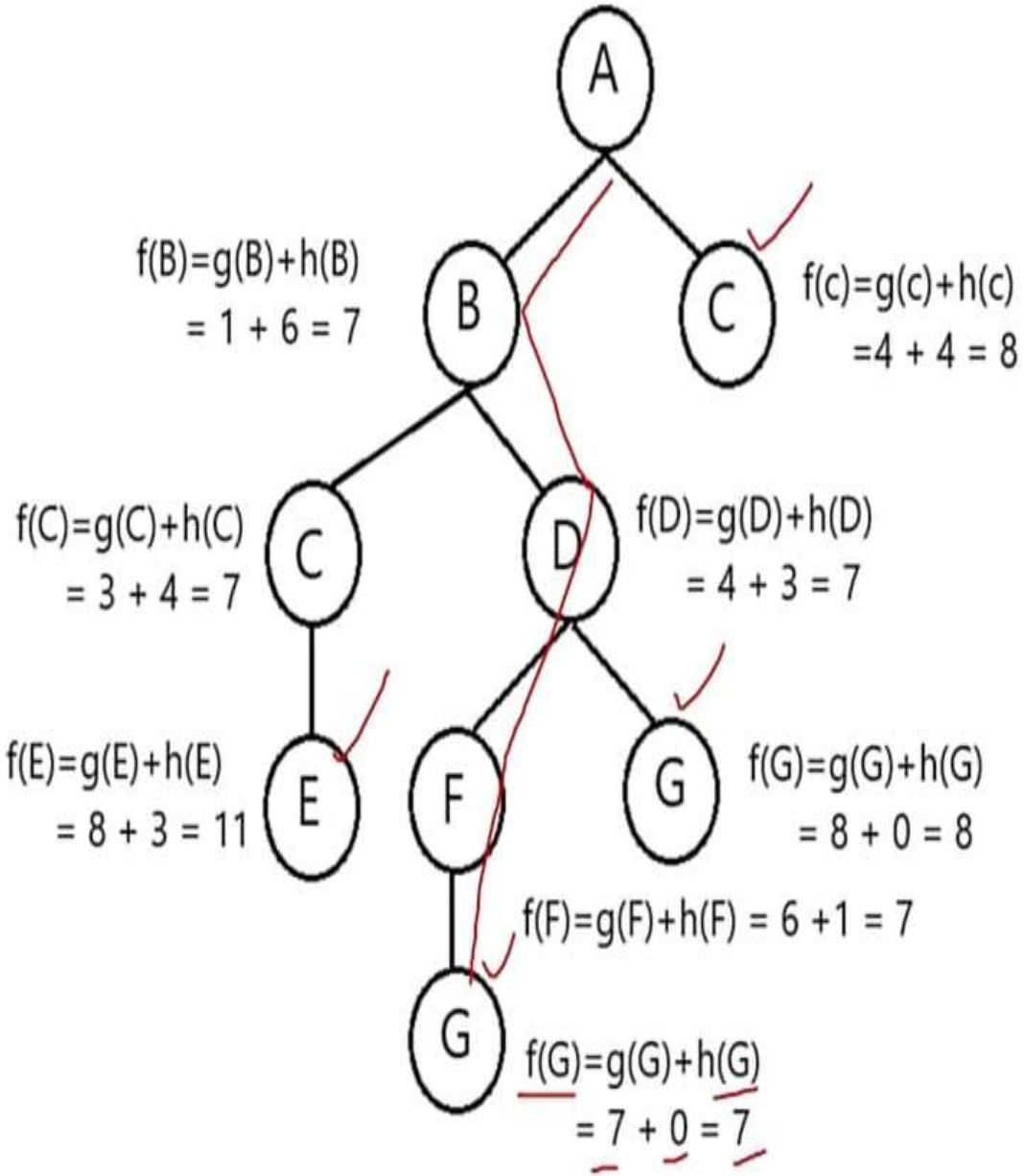
A	5
B	6
C	4
D	3
E	3
F	1
G	0



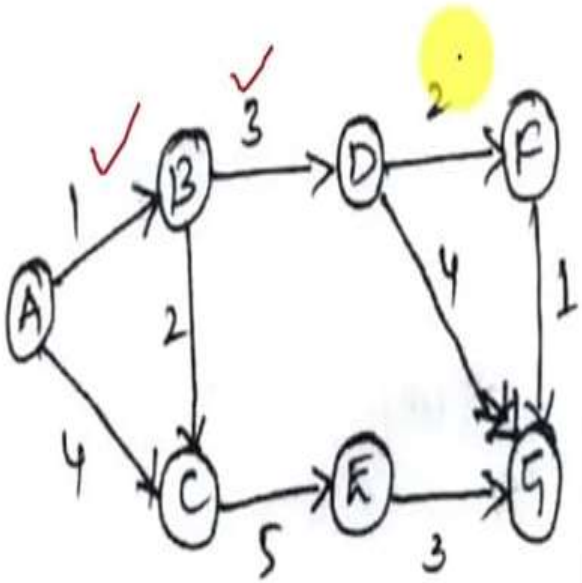
$$f(n) = g(n) + h(n)$$



A	5
B	6
C	4
D	3
E	3
F	1
G	0



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

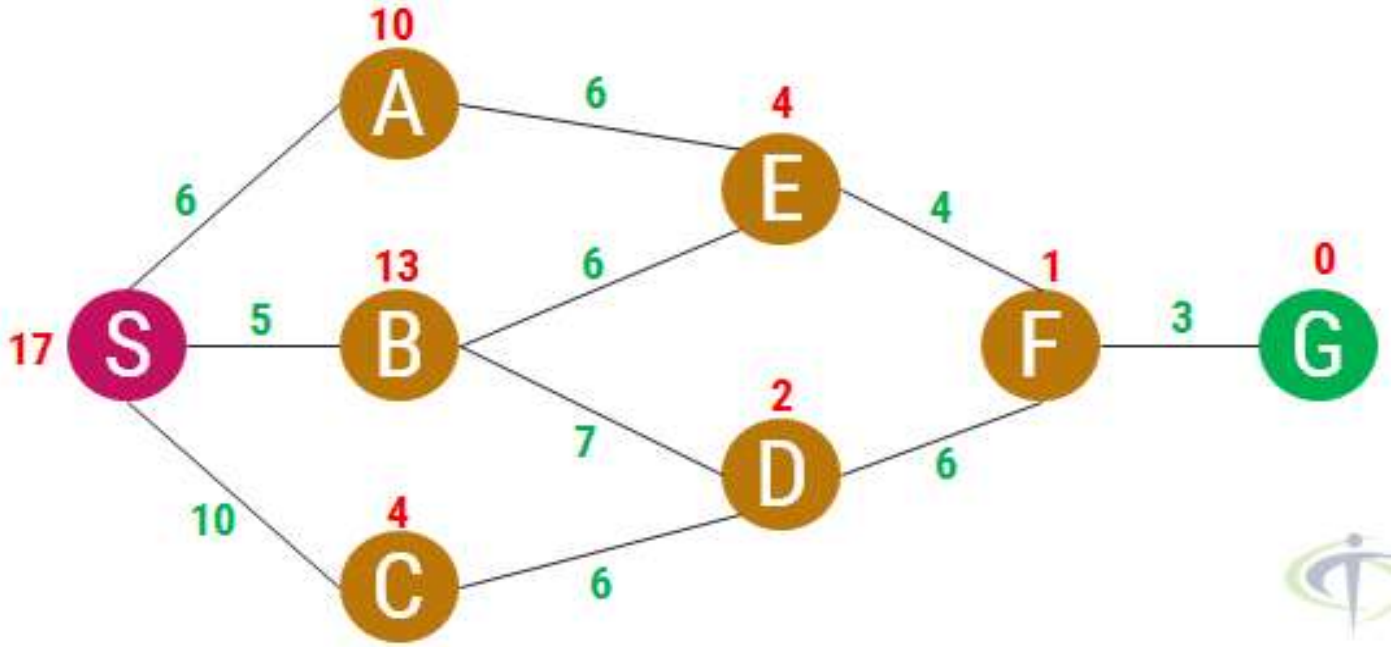


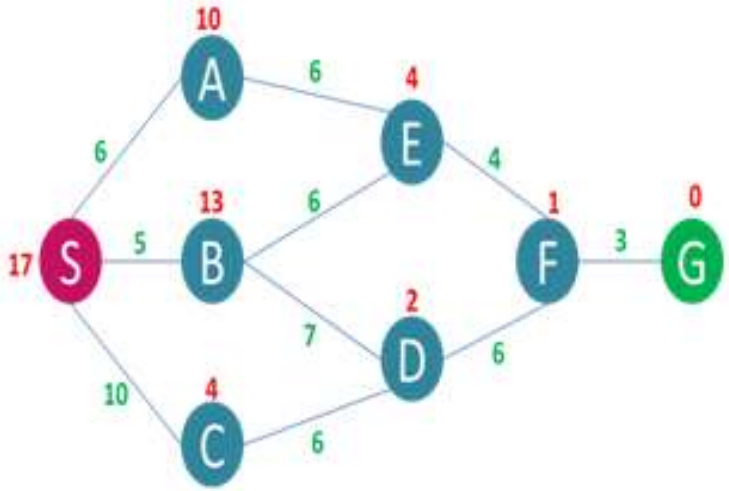
A	5
B	6
C	4
D	3
E	3
F	1
G	0

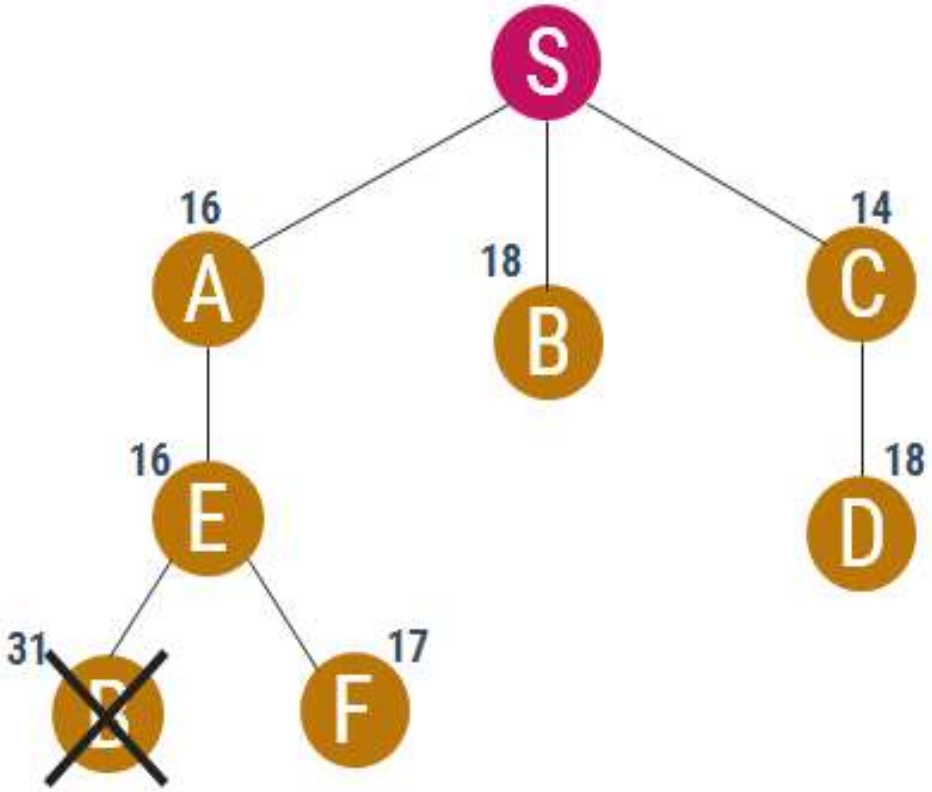
Path: A→B→D→F→G

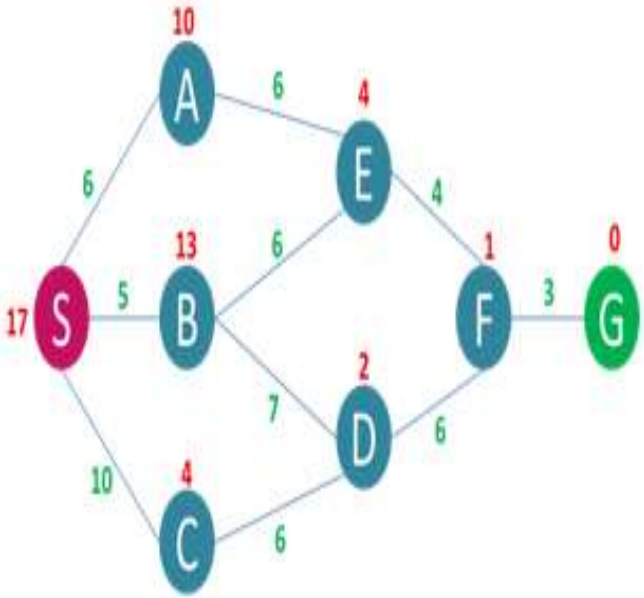
Path Cost: 7

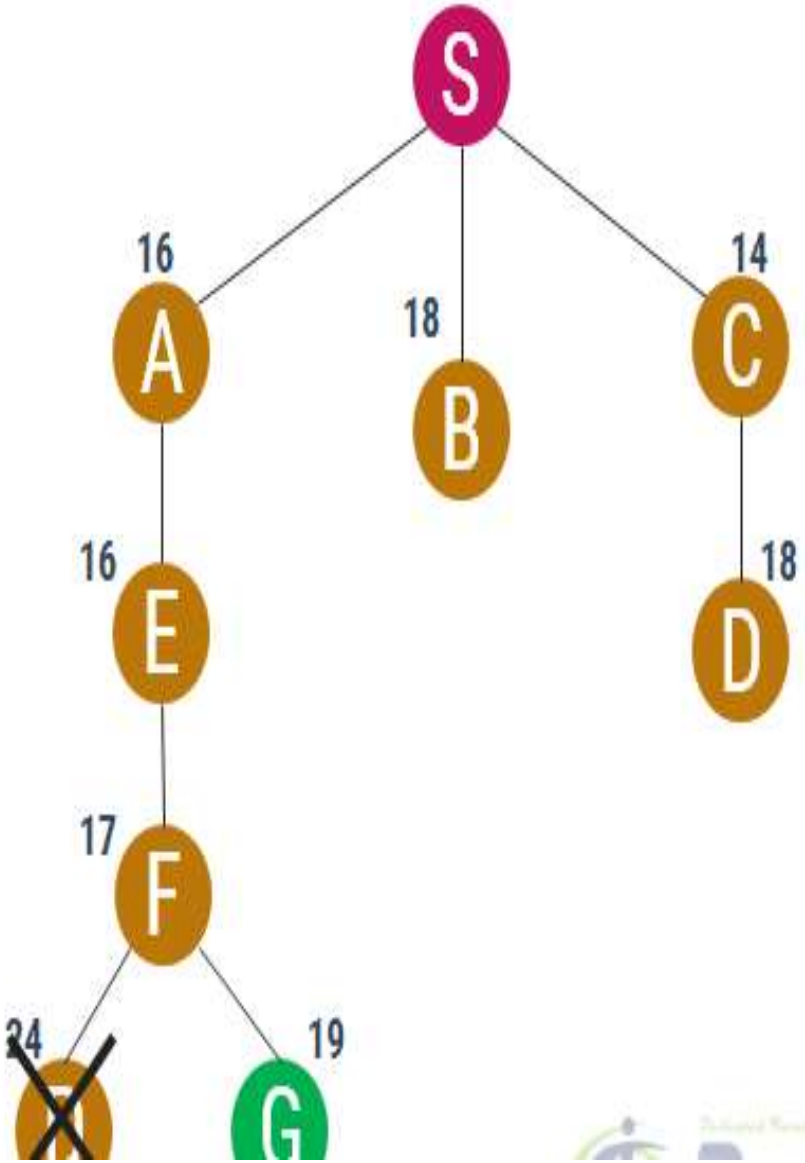
Example 2

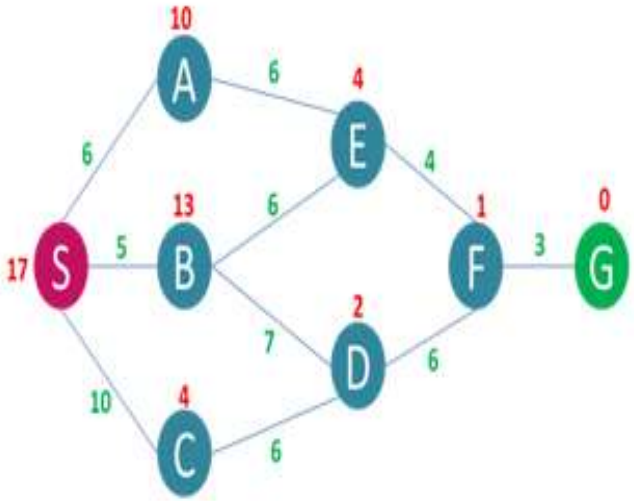


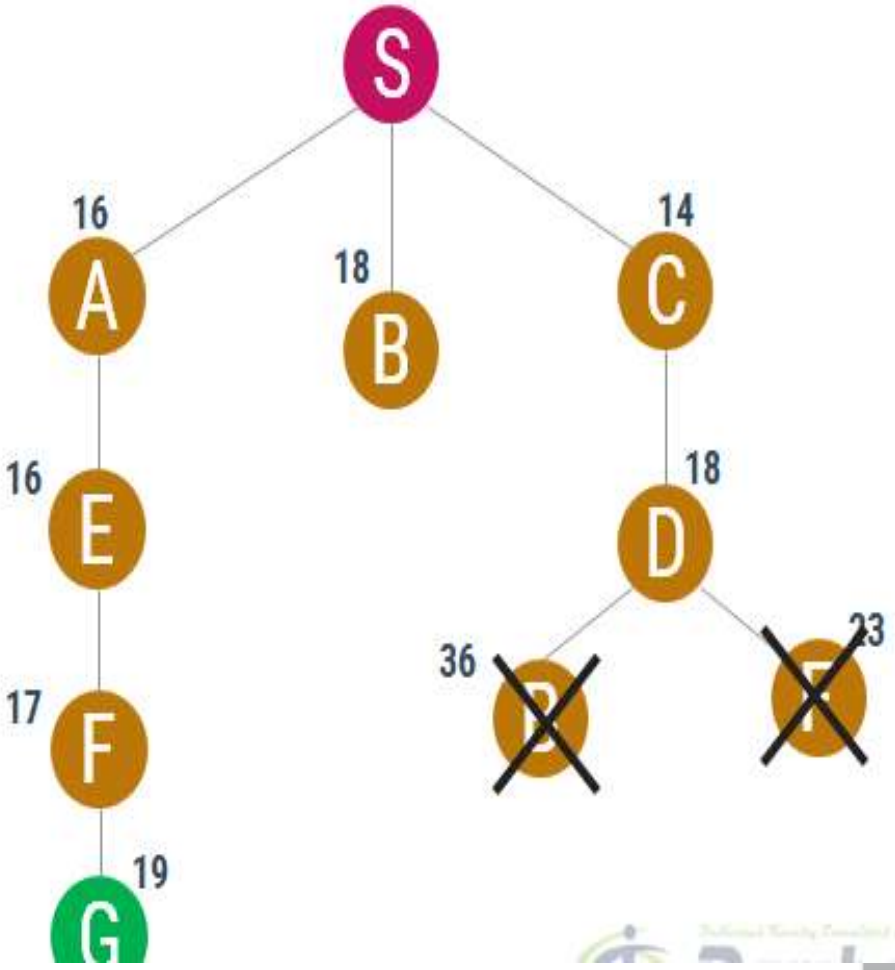

10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

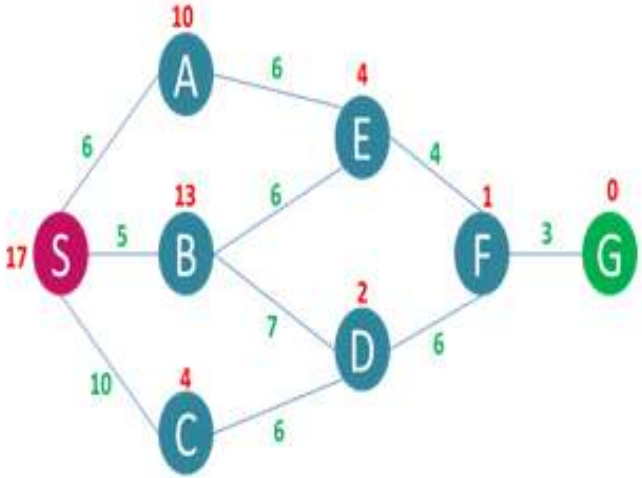

S
16
A
18
B
14
C
16
E
18
D
31
B
17
F

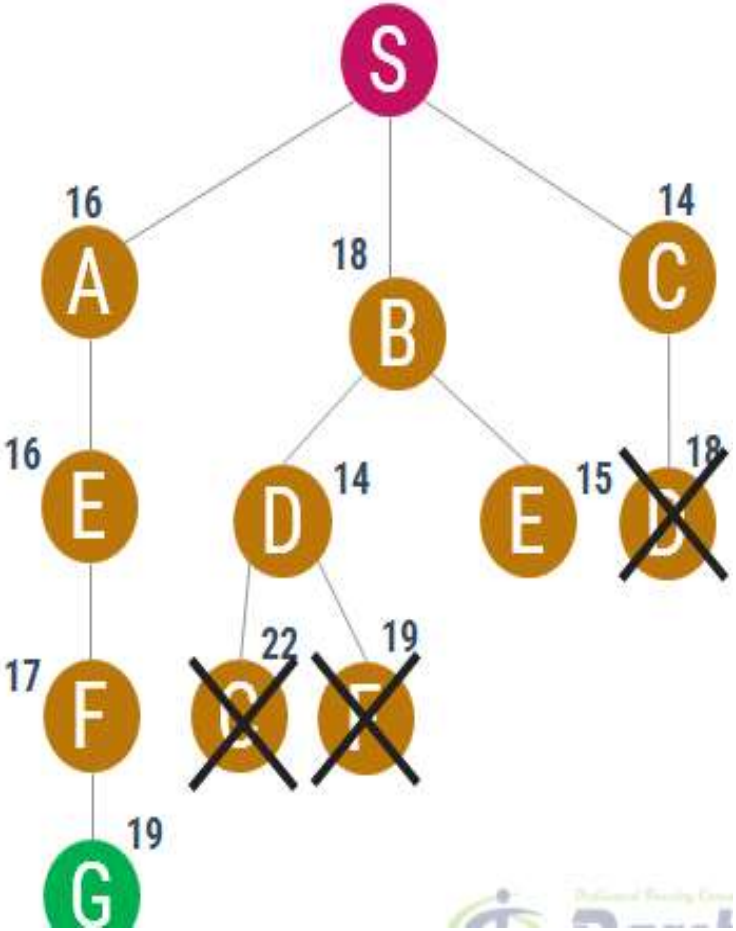

10
A
6
4
E
6
4
6
13
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

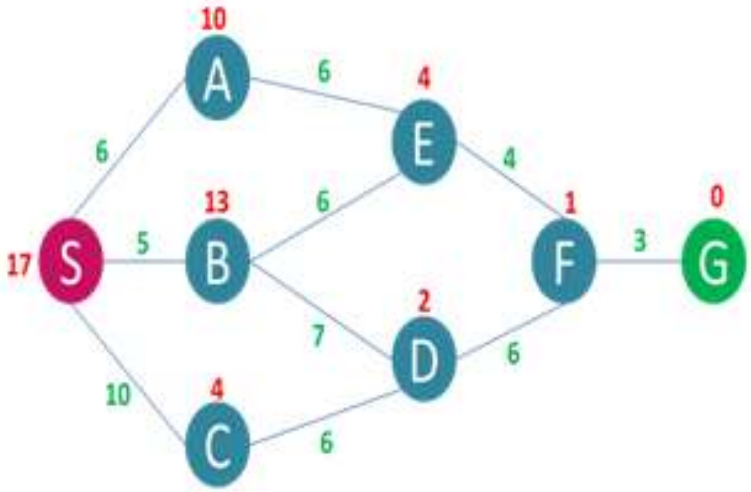

S
16
18
14
A
B
C
16
18
E
D
17
F
24
19
G

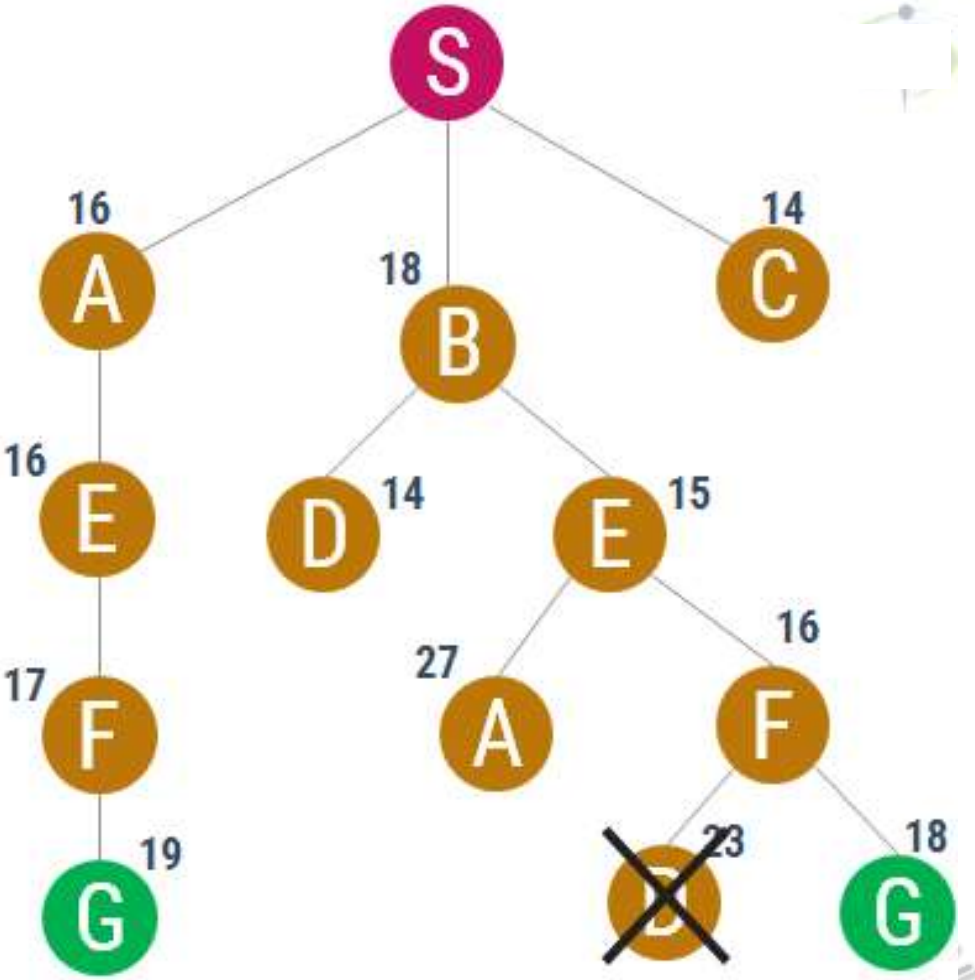

10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

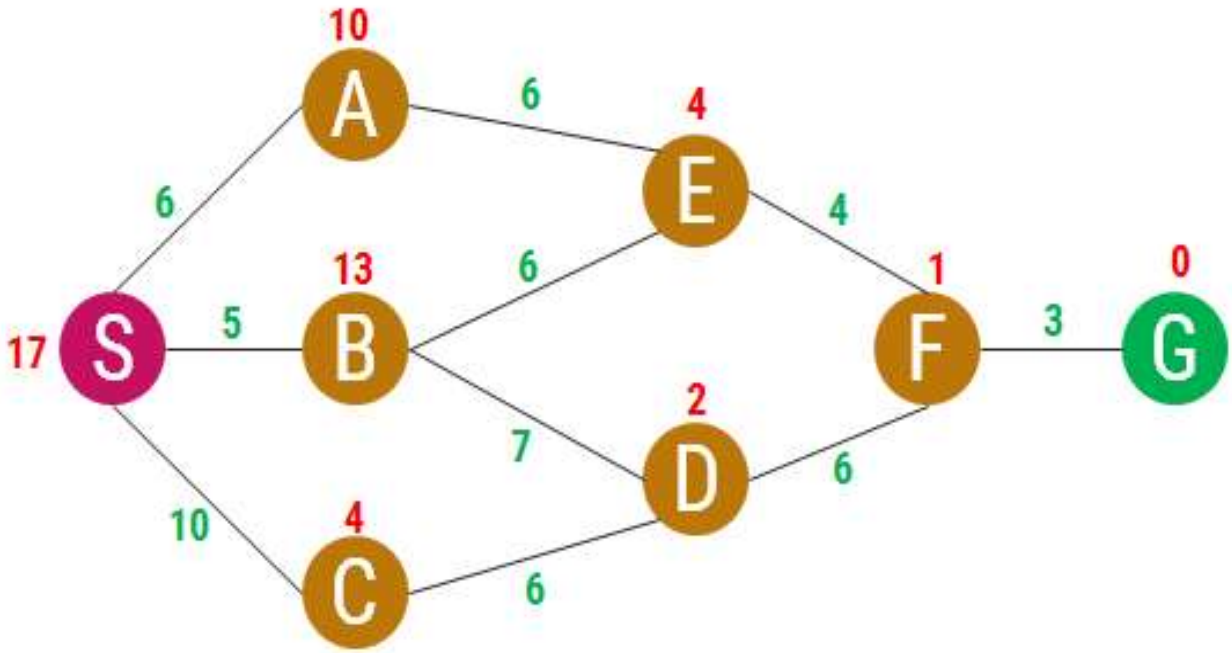

S
16
A
18
B
14
C
16
E
18
D
36
23
17
F
19
G


10
A
6
4
E
6
4
6
13
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6


S
16
A
18
B
14
C
16
E
D
14
E
15
18
22
19
17
F
19
G


10
A
6
4
E
6
4
6
13
B
1
F
3
0
G
17
S
5
2
7
D
6
10
4
C
6


S
16
A
18
B
14
C
16
E
14
D
15
E
17
F
27
A
16
F
19
G
23
D
18
G



$S \rightarrow B \rightarrow E \rightarrow F \rightarrow G$

Total Cost = 18

Example 3

Given an initial state of a 8-puzzle problem and final state to be reached-

2	8	3
1	6	4
7		5

Initial State

1	2	3
8		4
7	6	5

Final State

- Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

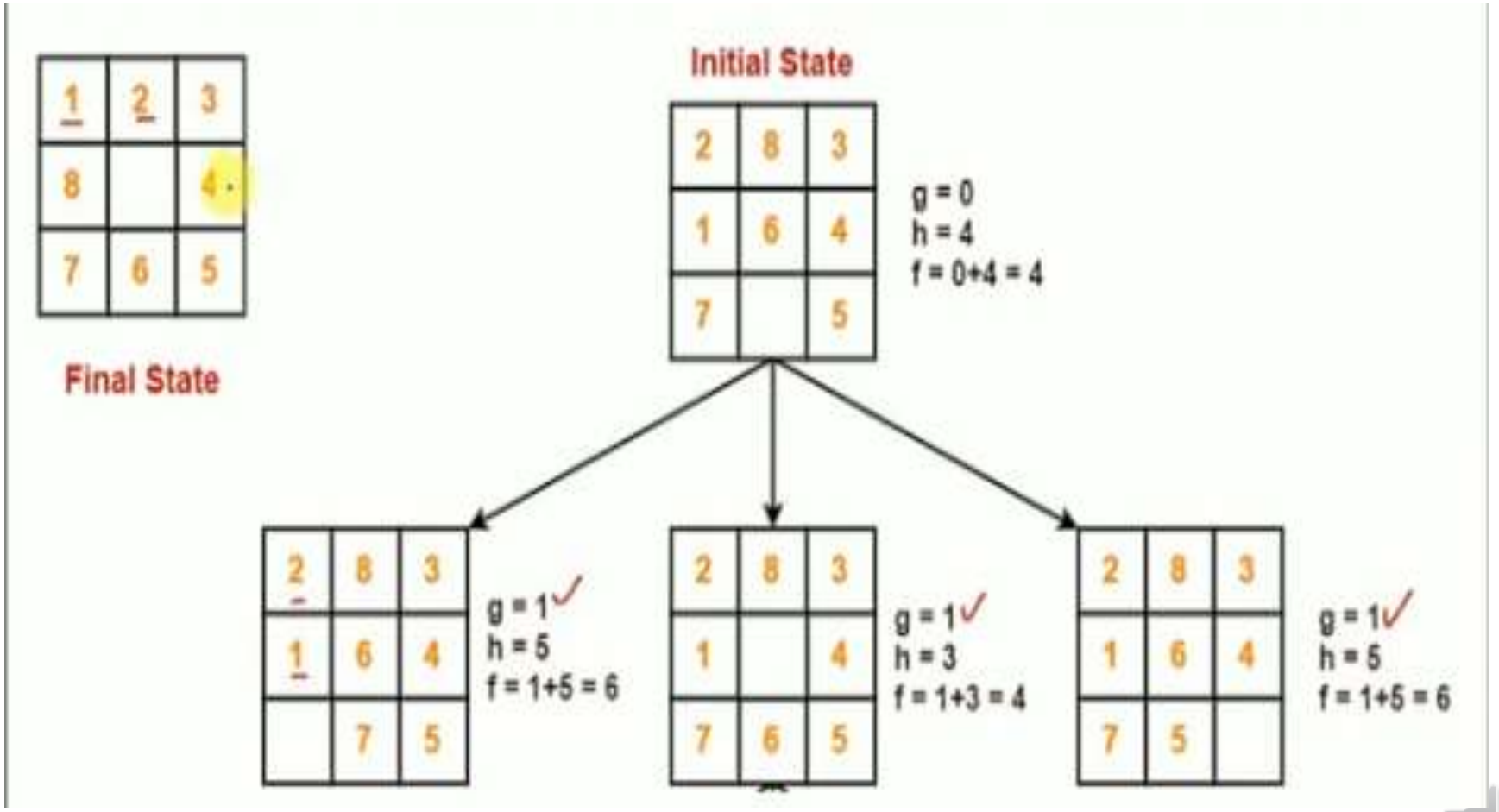
$$f(n) = g(n) + h(n)$$

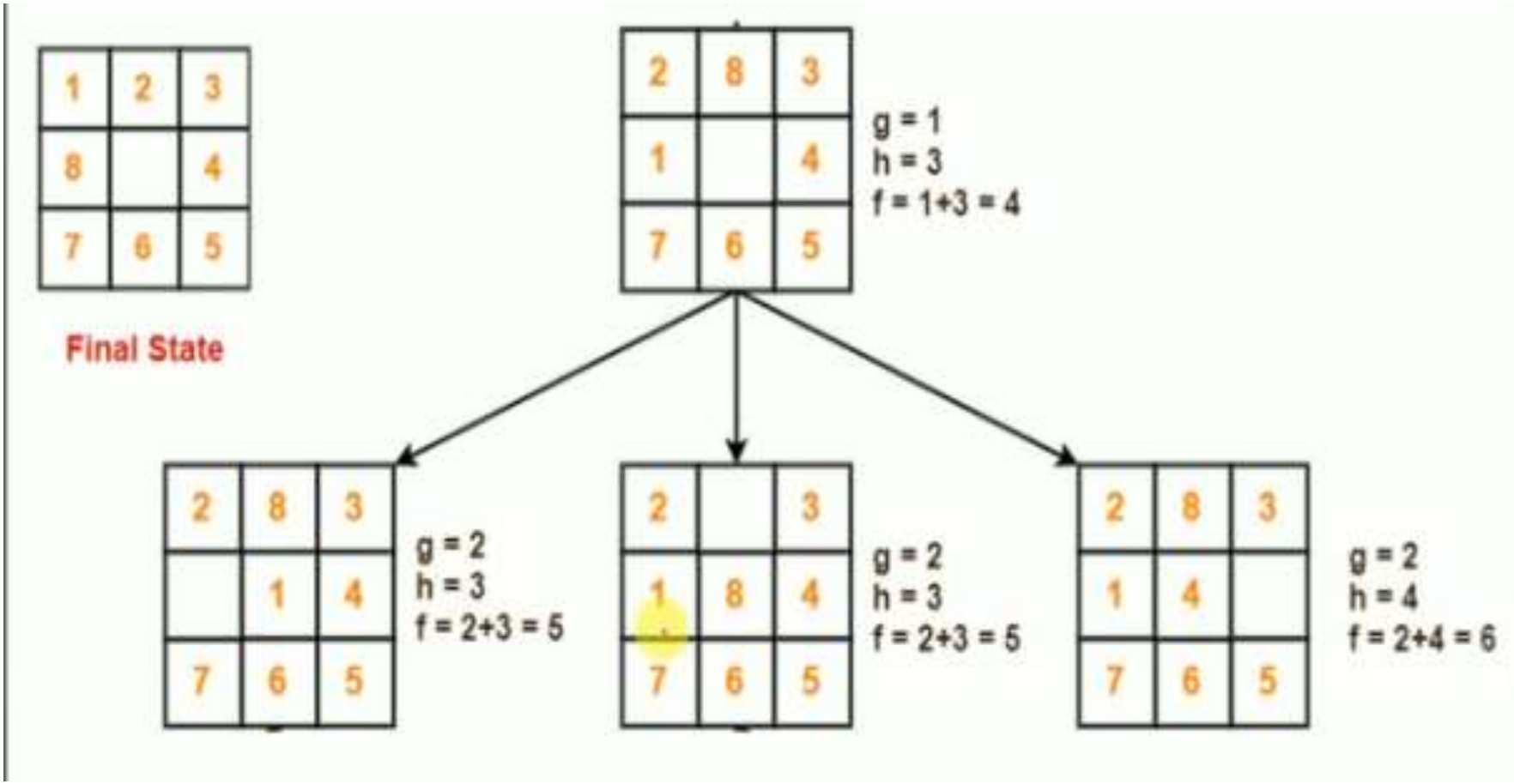
- Consider **g(n) = Depth of node** and **h(n) = Number of misplaced tiles**.

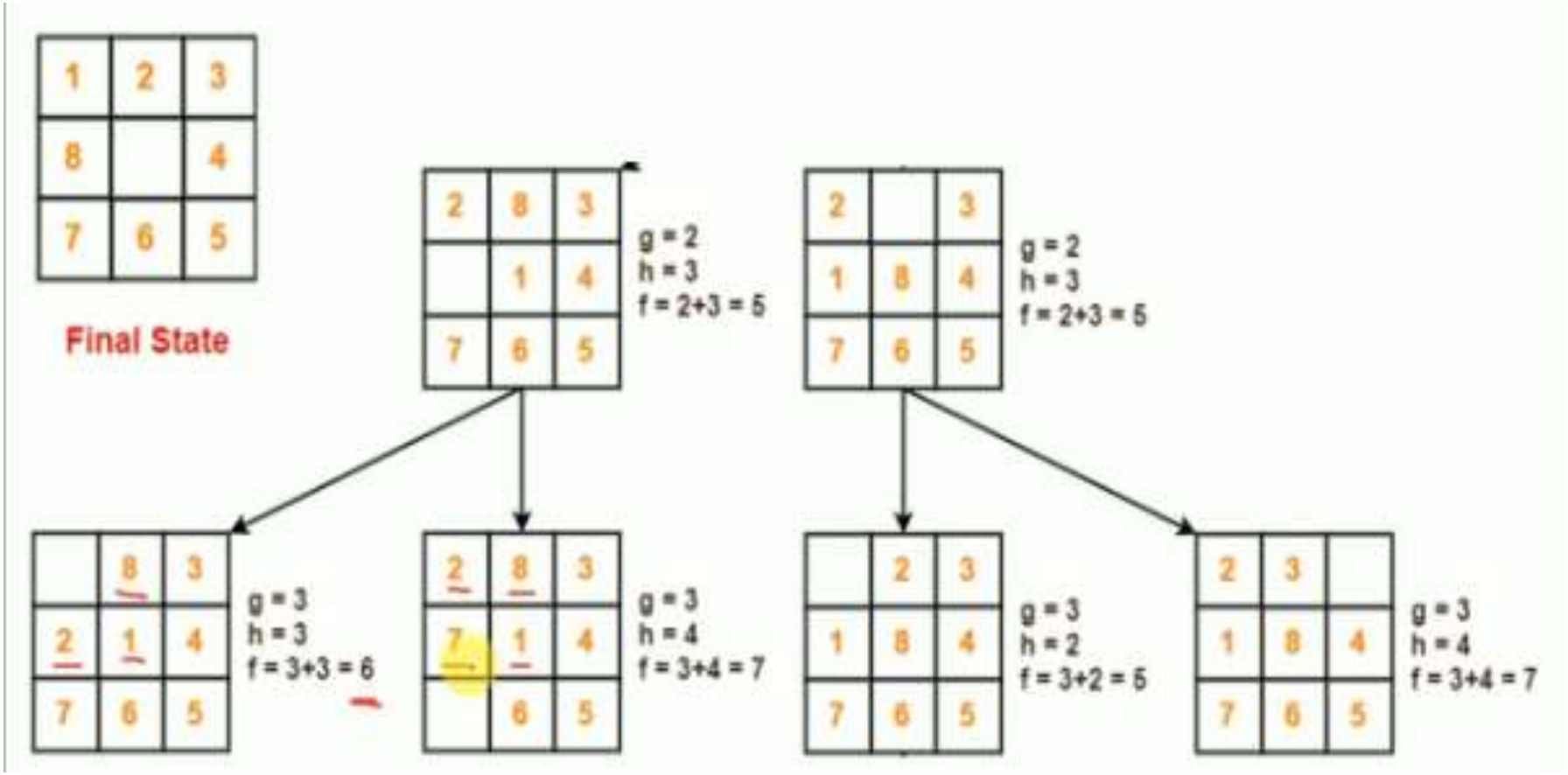
Initial State

2	8	3
1	6	4
7		5

g = 0 ✓
h = 4 ✓
f = 0+4 = 4


Final State
1 2 3
8 4
7 6 5
Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6


1 2 3
8 4
7 6 5
Final State
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6


1 2 3
8 4
7 6 5
Final State
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
8 3
2 1 4
7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3
7 1 4
6 5
g = 3
h = 4
f = 3+4 = 7
2 3
1 8 4
7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3
1 8 4
7 6 5
g = 3
h = 4
f = 3+4 = 7

1	2	3
8		4
7	6	5

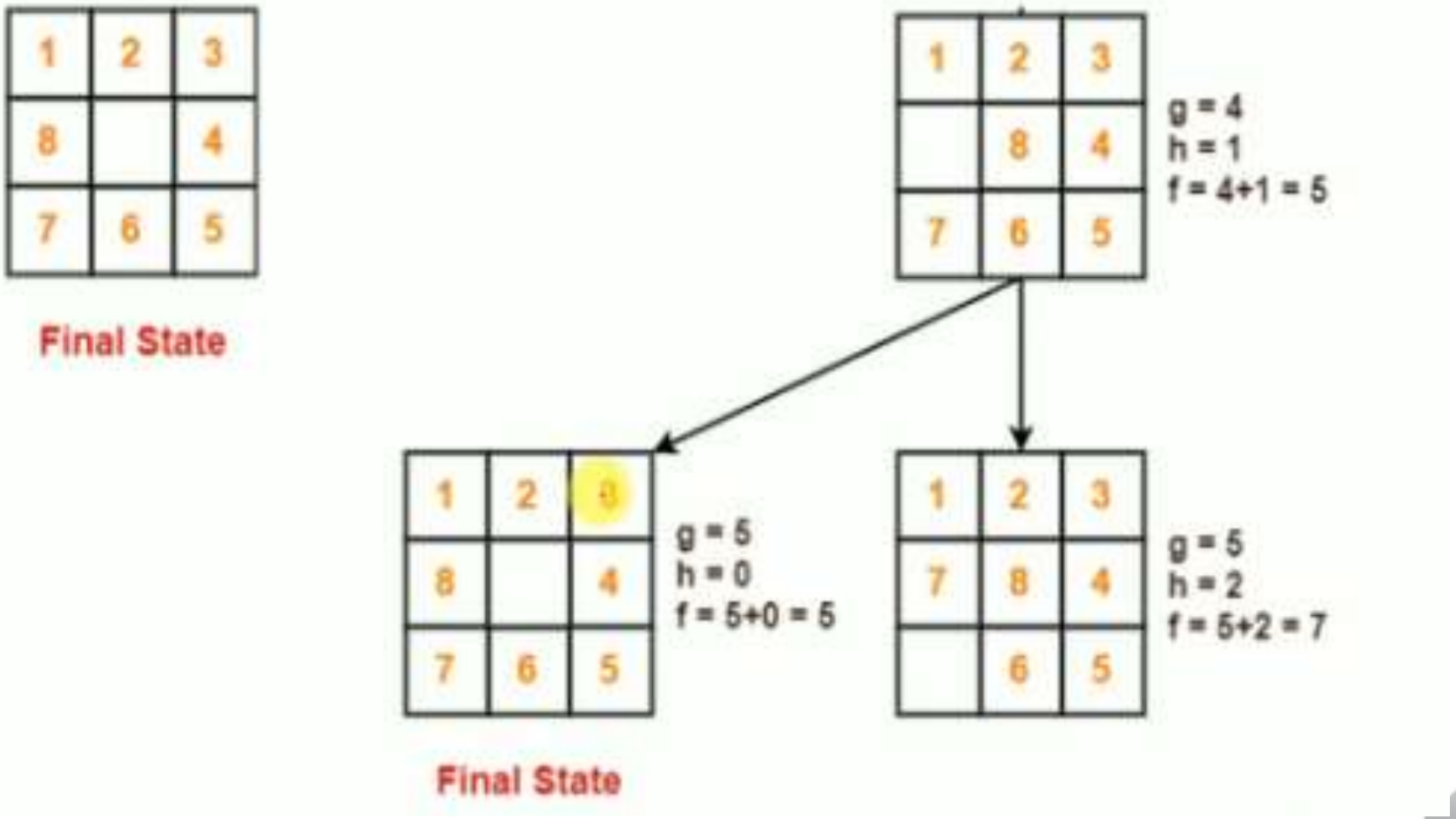
Final State

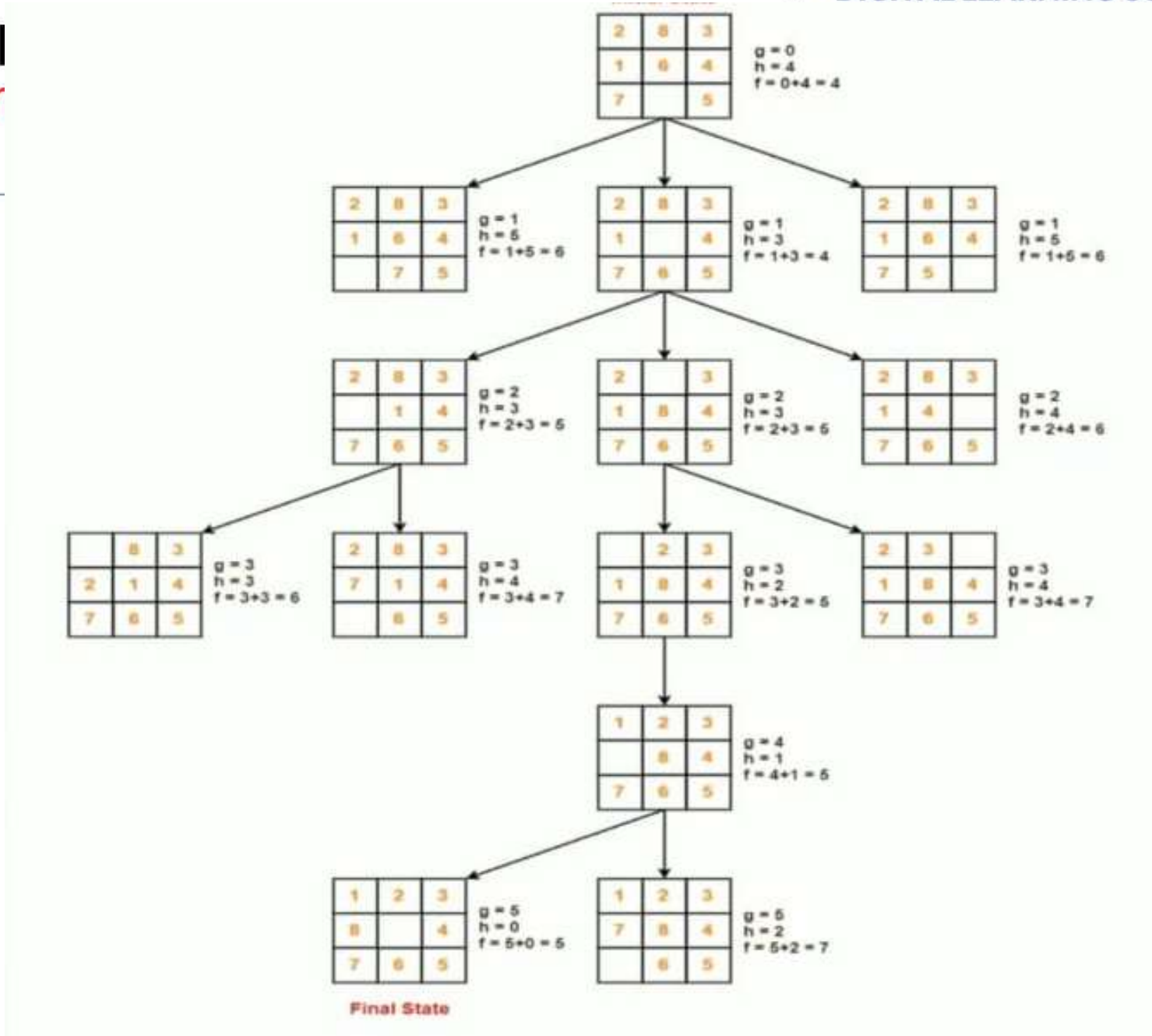
	2	3
1	8	4
7	6	5

g = 3
h = 2
f = 3+2 = 5

1.	2	3
	8	4
7	6	5

g = 4
h = 1
f = 4+1 = 5


1 2 3
8 4
7 6 5
Final State
g = 4
h = 1
f = 4+1 = 5
g = 5
h = 0
f = 5+0 = 5
Final State
g = 5
h = 2
f = 5+2 = 7


g = 0
h = 4
f = 0+4 = 4
g = 1
h = 5
f = 1+5 = 6
g = 1
h = 3
f = 1+3 = 4
g = 1
h = 5
f = 1+5 = 6
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 4
f = 2+4 = 6
g = 3
h = 3
f = 3+3 = 6
g = 3
h = 4
f = 3+4 = 7
g = 3
h = 2
f = 3+2 = 5
g = 3
h = 4
f = 3+4 = 7
g = 4
h = 1
f = 4+1 = 5
g = 5
h = 0
f = 5+0 = 5
g = 5
h = 2
f = 5+2 = 7
Final State

A* Algorithm

- This algorithm uses following functions:
  1. **f':** Heuristic function that estimates the merits of each node we generate. f' represents an estimate of the cost of getting from the initial state to a goal state along with the path that generated the current node. f' = g + h'
  2. **g:** The function g is a measure of the cost of getting from initial state to the current node.
  3. **h':** The function h' is an estimate of the additional cost of getting from the current node to a goal state.

- The algorithm also uses the lists: OPEN and CLOSED
  - **OPEN** - nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined. OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
  - **CLOSED** - nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree, since whenever a new node is generated; we need to check whether it has been generated before.

A* Algorithm

Step 1:

- Start with OPEN containing only initial node.
- Set that node's g value to 0, its h' value to whatever it is, and its f' value to h'+0 or h'.
- Set CLOSED to empty list.

Step 2: Until a goal node is found, repeat the following procedure:

- If there are no nodes on OPEN, report failure.
- Otherwise select the node on OPEN with the lowest f' value.
- Call it BESTNODE. Remove it from OPEN. Place it in CLOSED.
- See if the BESTNODE is a goal state. If so exit and report a solution.
- Otherwise, generate the successors of BESTNODE but do not set the BESTNODE to point to them yet.

A* Algorithm

Step 3: For each of the SUCCESSOR, do the following:

a. Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.

b. Compute g(SUCCESSOR) = g(BESTNODE) + the cost of getting from BESTNODE to SUCCESSOR

c. See if SUCCESSOR is the same as any node on OPEN. If so call the node OLD.

   i. Check whether it is cheaper to get to OLD via its current parent or to SUCESSOR via BESTNODE by comparing their g values.

   ii. If OLD is cheaper, then do nothing. If SUCCESSOR is cheaper then reset OLD's parent link to point to BESTNODE.

   iii. Record the new cheaper path in g(OLD) and update f'(OLD).

   iv. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors.

   v. If SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors. Compute f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR)

Advantages of A* Algorithm

- Optimality: A* guarantees finding the shortest path between a given start and goal node, provided that certain conditions are met
- Efficiency: A* is typically more efficient than brute-force search algorithms like breadth-first search or depth-first search, especially in large graphs.
- Flexibility: A* can be applied to a wide range of problems, including pathfinding in grids, maps, or networks
- Speed: A* is often faster than other uninformed search algorithms because it intelligently prioritizes the exploration of nodes.

Disadvantages of A* Algorithm

- Heuristic Accuracy: The performance of A* heavily relies on the accuracy of the heuristic function used
- Memory Requirements: A* needs to keep track of the nodes in the open and closed sets during the search process.
- Computational Complexity: Although A* is generally efficient, its worst-case time complexity is exponential. In graphs with many possible paths or when using a poor heuristic, A* can degrade in performance. In such cases, alternative algorithms like Dijkstra's algorithm (which guarantees optimality but lacks the heuristic-guided search) might be more suitable.
- Path Reoptimization: A* assumes that the environment and costs remain static throughout the search process

Parul® University NAAC A++ ACCREDITED UNIVERSITY

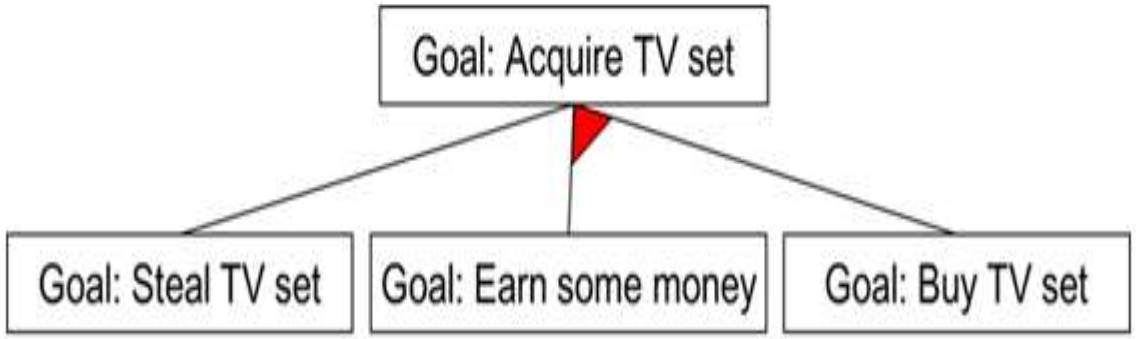
DIGITAL LEARNING CONTENT

Problem Reduction

- AND-OR graph (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
- This decomposition or reduction generates arcs that we call AND arcs.
- One AND arc may point to any numbers of successor nodes. All of which must then be solved in order for the arc to point solution.
- In order to find solution in an AND-OR graph we need an algorithm similar to best –first search but with the ability to handle the AND arcs appropriately.
- We define FUTILITY, if the estimated cost of solution becomes greater than the value of FUTILITY then we abandon the search, FUTILITY should be chosen to correspond to a threshold.

AND-OR Graph

- In following figure AND arcs are indicated with a line connecting all the components.



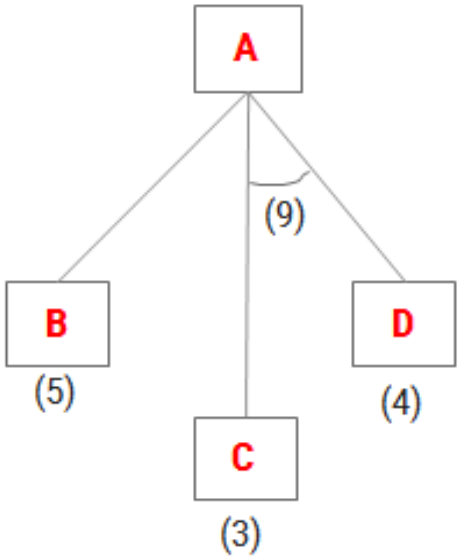
Parul® University

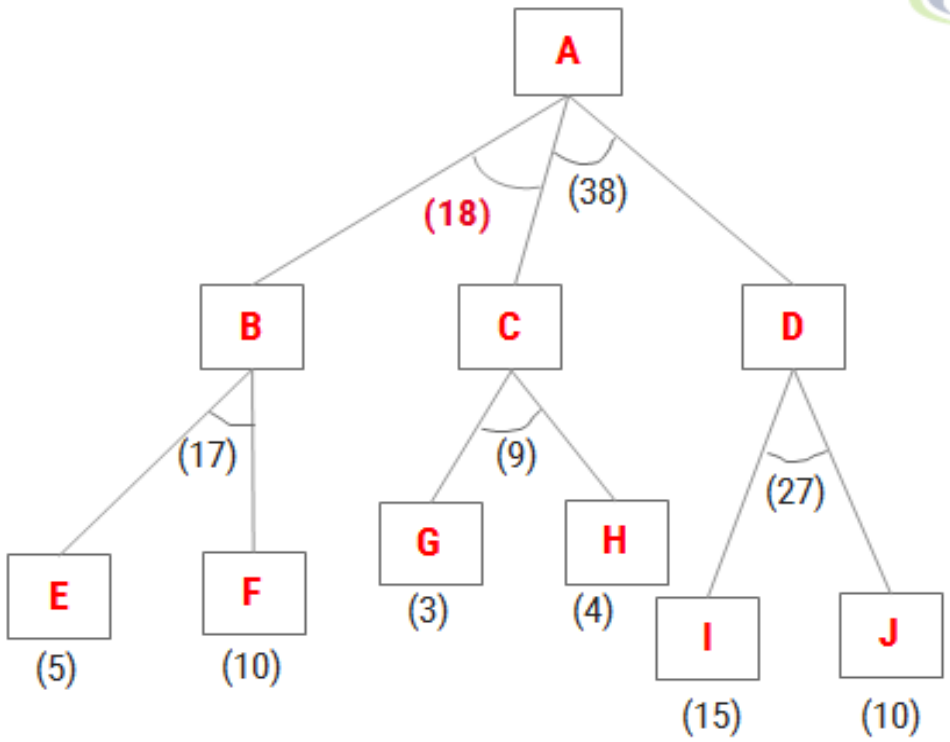
NAAC A++ ACCREDITED UNIVERSITY

Difference Between A* And AO*

	A*	**AO***
1	Optimal Solution	Not guarantees for optimal solution
2	Explore all the paths	Stops exploring paths, once found the solution
3	Works for simple problem	Works for complex problem
4	No arcs are used	AND arcs are used

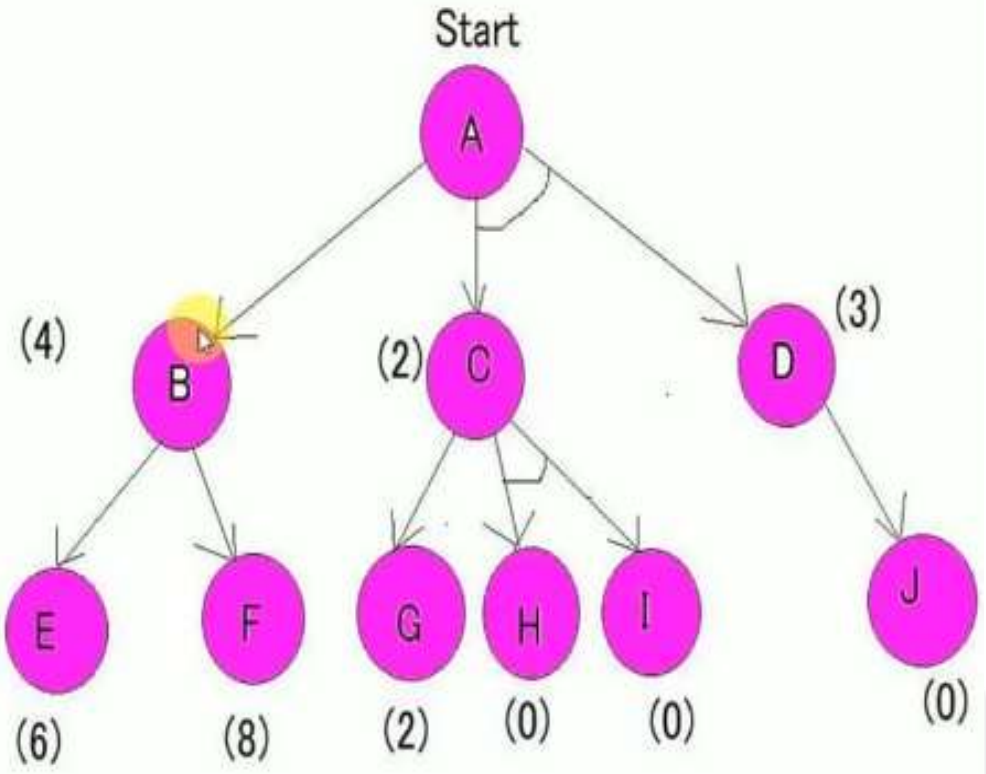
AO* Algorithm- Example 1





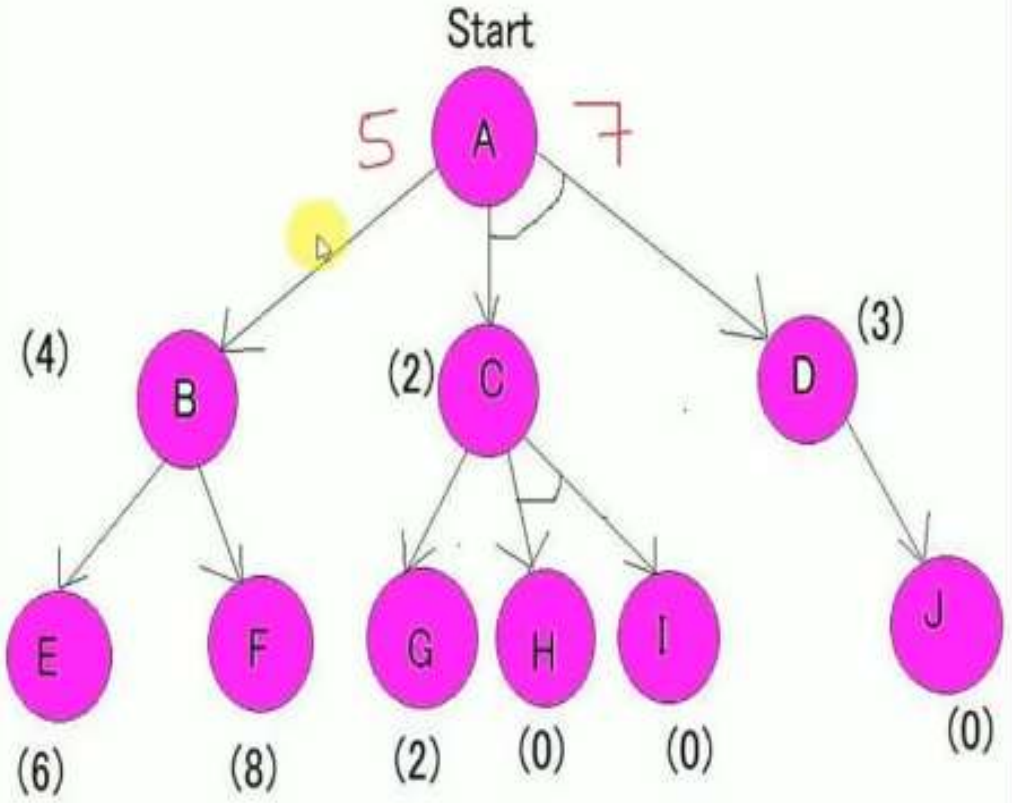
AO* Algorithm- Example 2

- Here, in the above example all numbers in brackets are the heuristic value i.e h(n).
- Each edge is considered to have a value of 1 by default.



Step-1

- Starting from node A, we first calculate the best path.
- **f(A-B) = g(B) + h(B) = 1+4= 5** , where 1 is the default cost value of travelling from A to B and 4 is the estimated cost from B to Goal state.
- **f(A-C-D) = g(C) + h(C) + g(D) + h(D) = 1+2+1+3 = 7**, here we are calculating the path cost as both C and D because they have the AND-Arc.
- The default cost value of travelling from A-C is 1, and from A-D is 1, but the heuristic value given for C and D are 2 and 3 respectively hence making the cost as 7.



Step-2

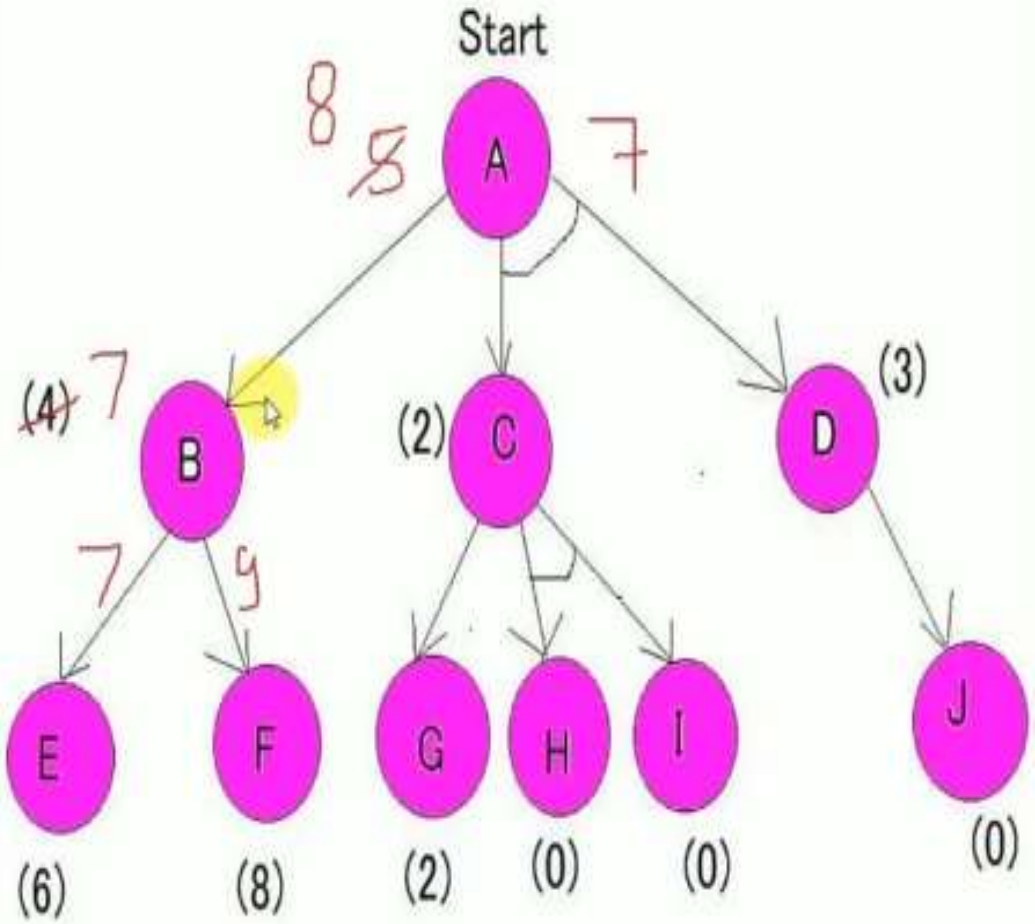
- From the B node,

  f(B-E) = 1 + 6 = 7

  f(B-F) = 1 + 8 = 9

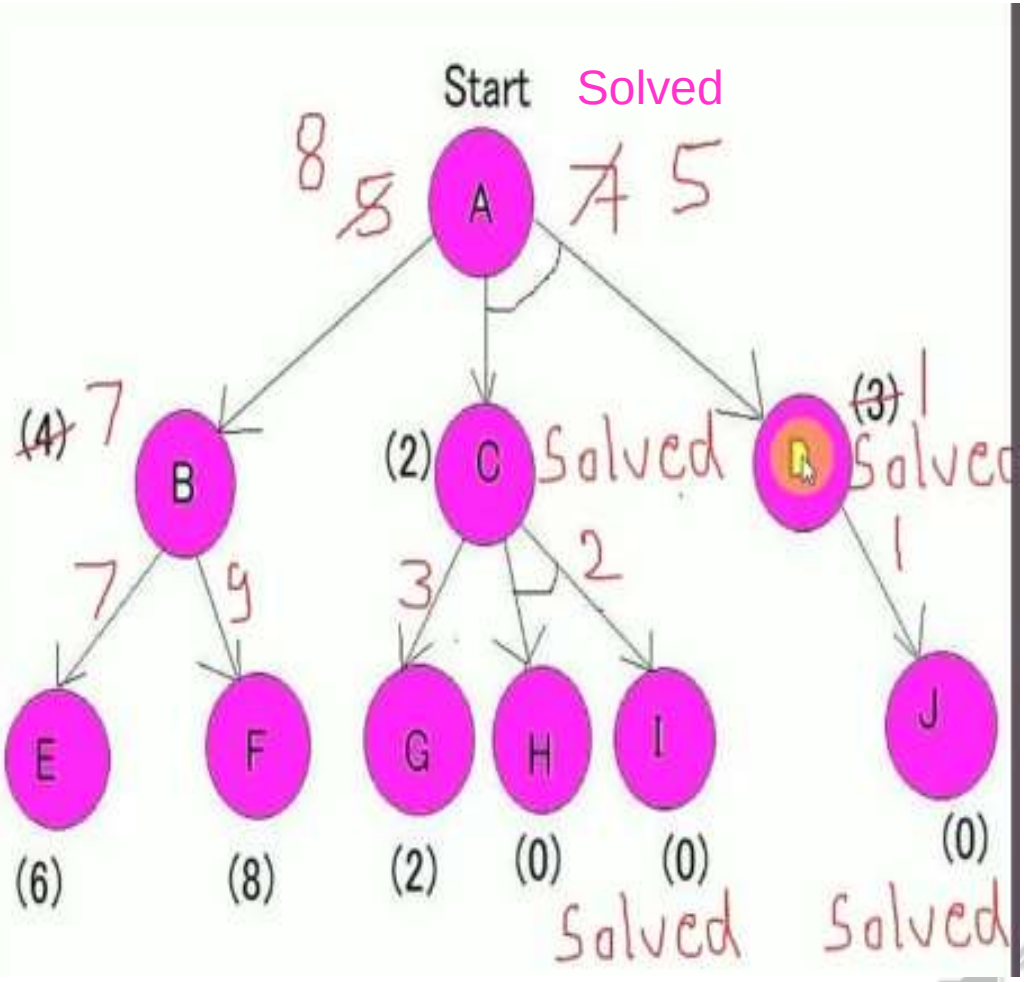
- Hence, the B-E path has lesser cost. Now the heuristics have to be updated since there is a difference between actual and heuristic value of B.
- The minimum cost path is chosen and is updated as the heuristic , in our case the value is 7.
- And because of change in heuristic of B there is also change in heuristic of A which is to be calculated again.

  f(A-B) = g(B) + updated((h(B)) = 1+7=8

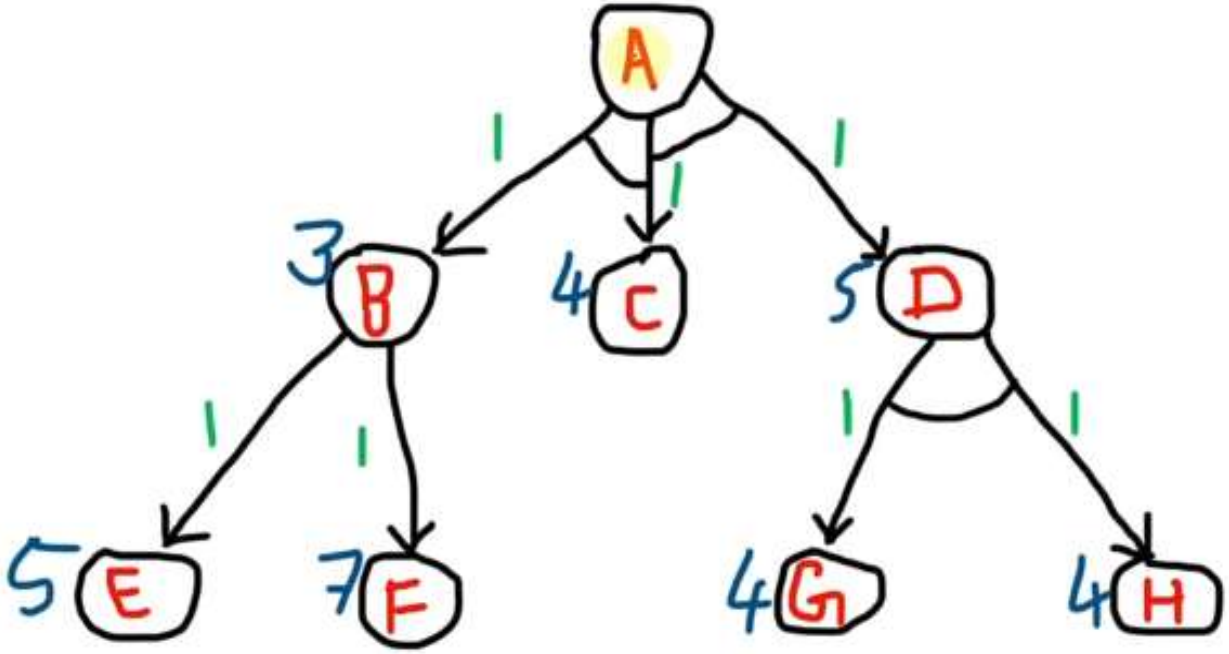


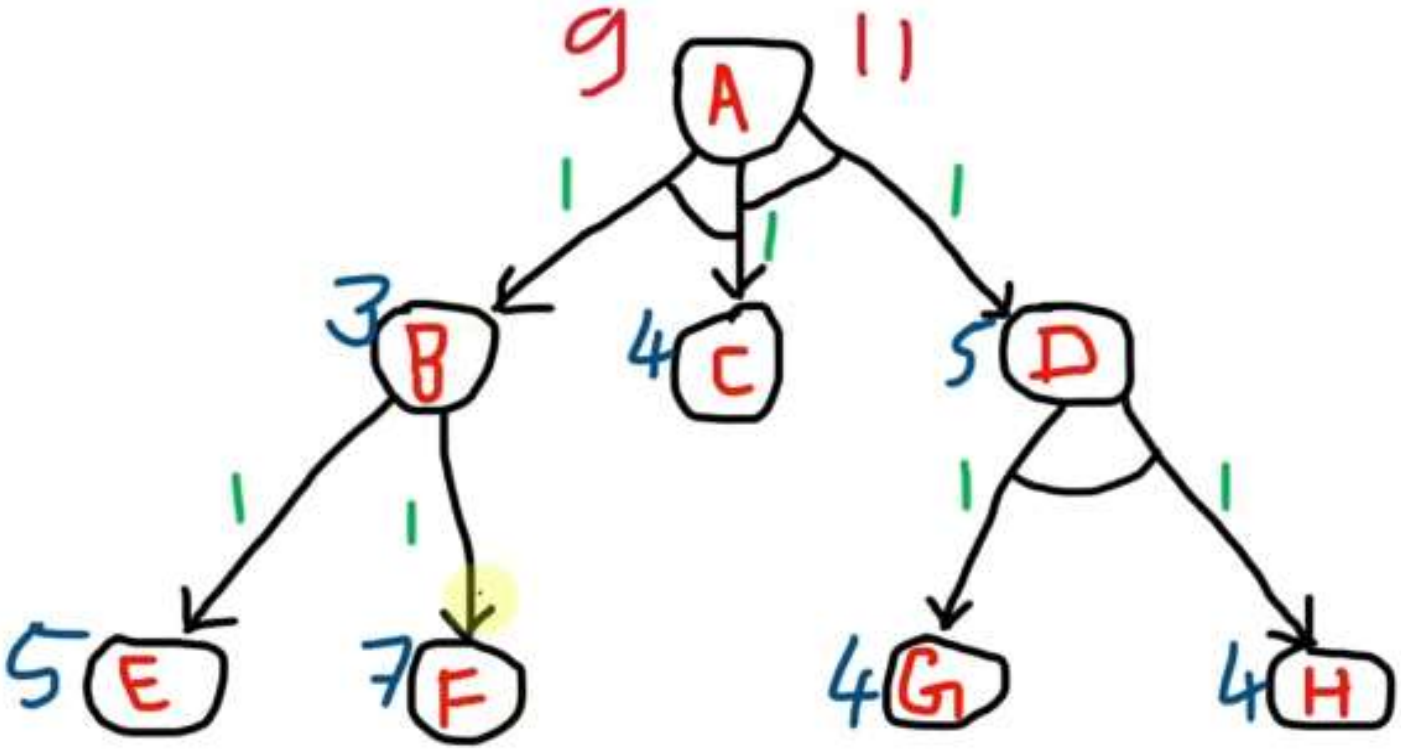
Step-3

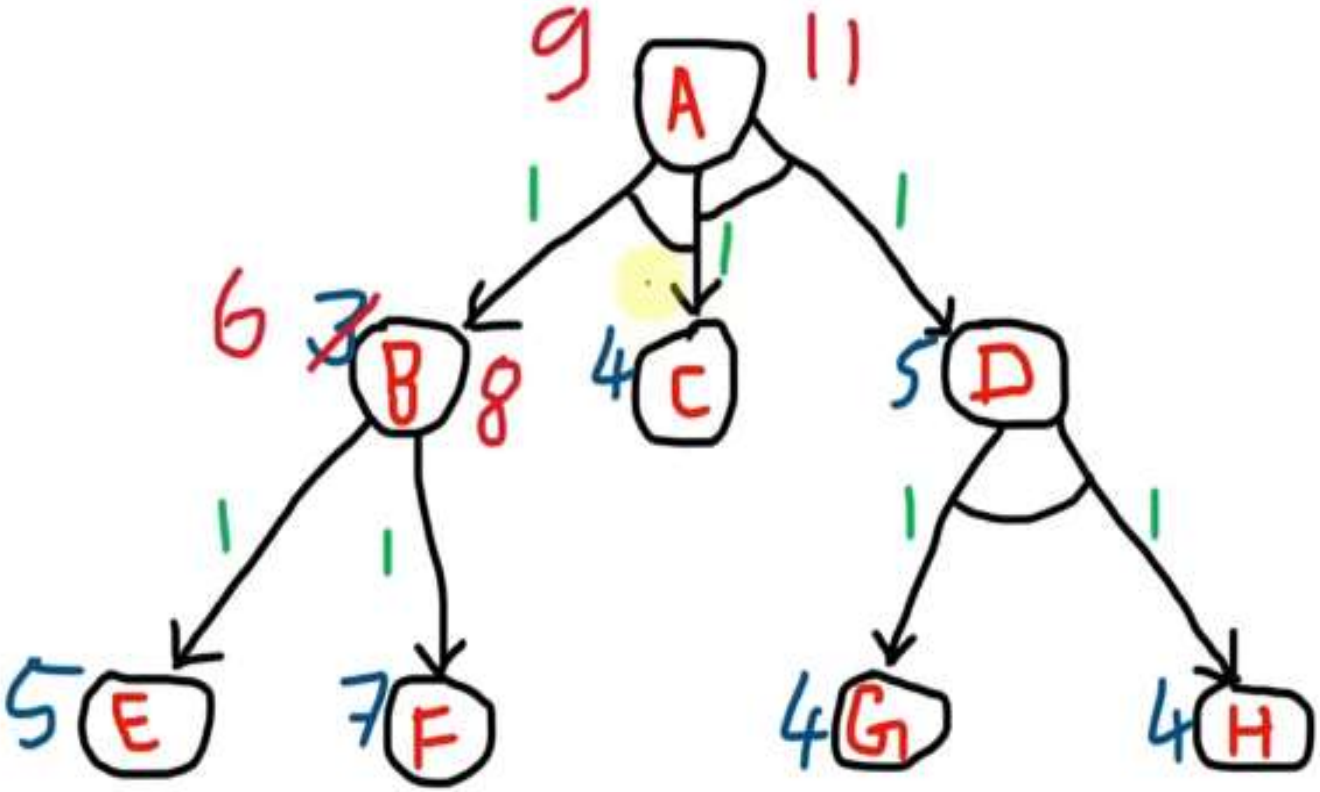
- Now the current node becomes C node and the cost of the path is calculated,
  f(C-G) = 1+2 = 3
  f(C-H-I) = 1+0+1+0 = 2
  f(C-H-I) is chosen as minimum cost path.
- Heuristic of path of H and I are 0 and hence they are solved,
- But Path A-D also needs to be calculated , since it has an AND-arc.
- **f(D-J) = 1+0 = 1**, hence heuristic of D needs to be updated to 1.
- And finally the f(A-C-D) needs to be updated.
  f(A-C-D) = g(C) + h(C) + g(D) + updated((h(D))
  = 1+2+1+1 =5.

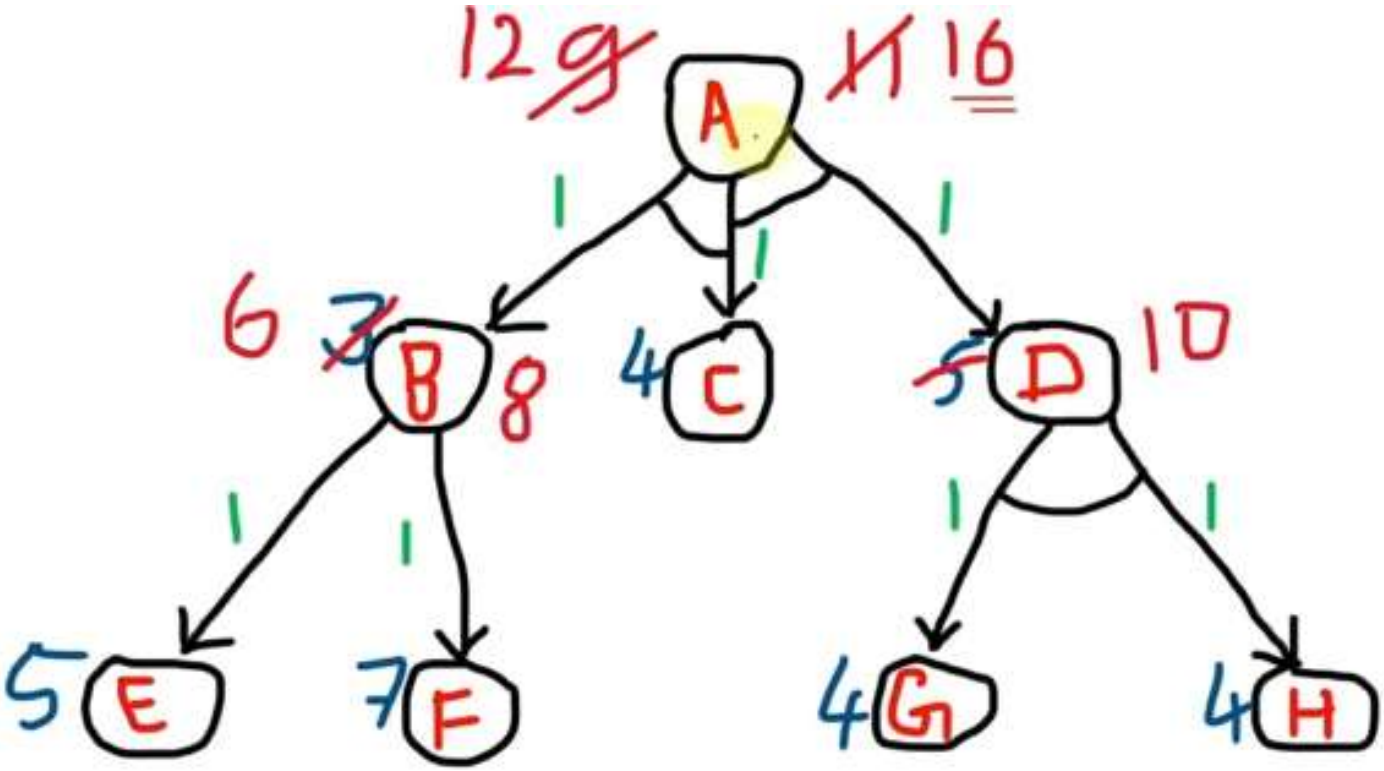


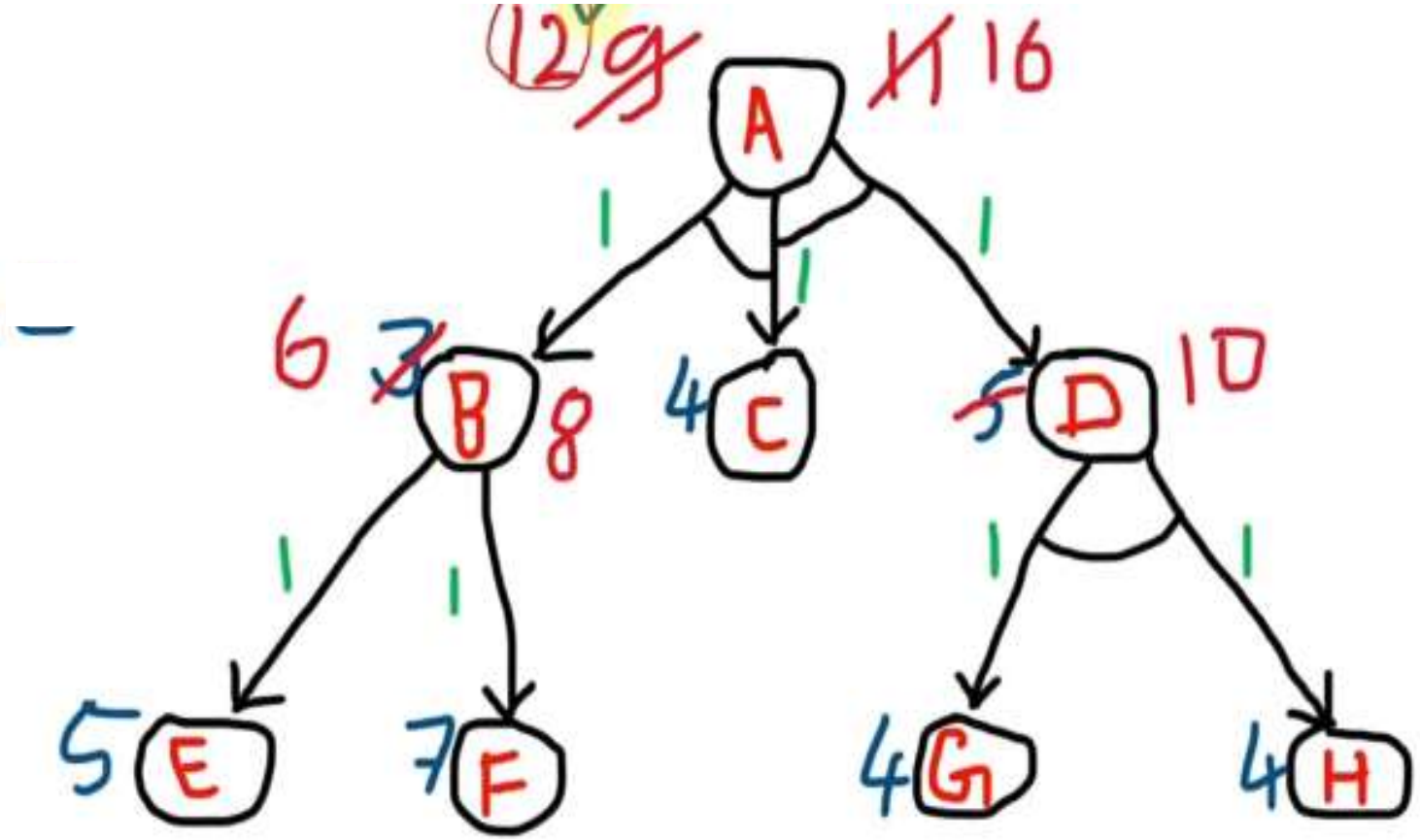
Example-3




9
A
11
1
1
1
3
B
4
C
5
D
1
1
1
1
5
E
7
F
4
G
4
H


9
A
11
1
1
1
6
3
B
8
4
C
5
D
1
1
1
1
5
E
7
F
4
G
4
H


12 9
A
11 16
1
1
1
6 7
B
8
4
C
5
D
10
1
1
1
1
5
E
7
F
4
G
4
H


12
A
16
1
1
1
6
B
8
4
C
5
D
10
1
1
1
1
5
E
7
F
4
G
4
H

Heuristic Search techniques for AO*

- Traverse the graph starting at the initial node and following the current best path, and accumulate the set of nodes that are on the path and have not yet been expanded.
- Pick one of these unexpanded nodes and expand it. Add its successors to the graph and computer f'(cost of the remaining distance) for each of them.
- Change the f' estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which is the current best path.
- The propagation of revised cost estimation backward is in the tree is not necessary in A* algorithm. This is because in AO* algorithm expanded nodes are re-examined so that the current best path can be selected.

Parul® University NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Constraint Satisfaction Problem

Constraint Satisfaction Problem (CSP)

- CSP consists of three components V, D, C
- V is set of variables $\{V_1\ V_2 \ldots V_n\}$
- D is set of domains $\{D_1\ D_2\ D_3 \ldots D_n\}$ one for each variable
- C is set of Constraints that specify allowable combination of values.

$C_i = (\text{Scope}, \text{rel})$

$\{C_1\ C_2\ C_3\}$

- Where Scope is set of variables that participate in Constraint
  - Rel is relation that define the values that variable can take

$V_1\ V_2$

$A\ \ B$

$C_1 = \left((V_1, V_2), (V_1 \neq V_2)\right)$

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Constraint Satisfaction Problem

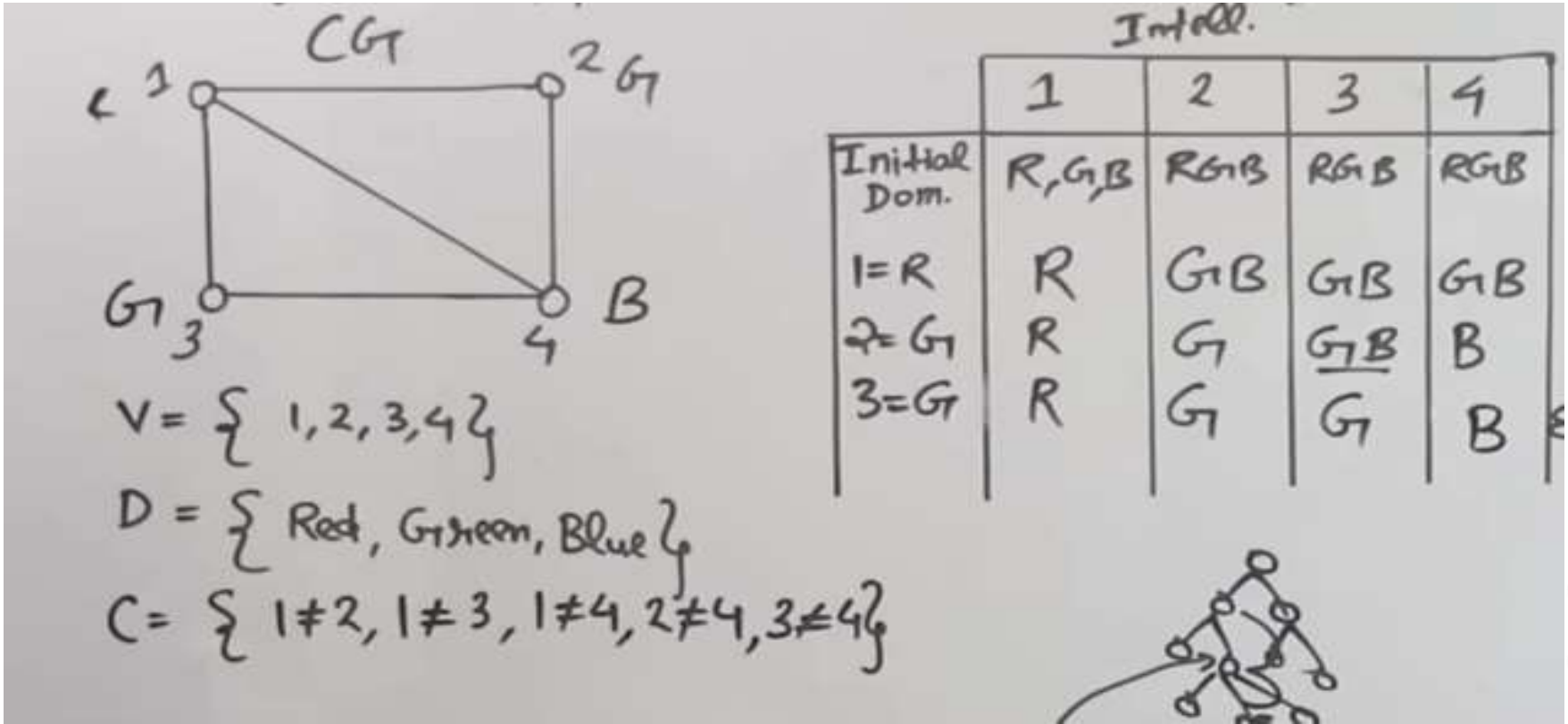
- CSPs are a key topic in computer science, especially in artificial intelligence (AI).
- **Definition**: A CSP consists of a set of variables, a domain for each variable, and a set of constraints that specify allowable combinations of values.
- **Variables**: Elements that have to be assigned values from their domains. For example, in a scheduling problem, variables could be different time slots.
- **Domains**: The range of possible values that each variable can take. For example, in a coloring problem, the domain might be a set of colors.
- **Constraints**: Rules that define which combinations of values are valid or invalid. They restrict the ways in which variables can be assigned values.
- **Solutions**: Assignments of values to all variables that do not violate any constraints. A problem can have zero, one, or multiple solutions.
- **Solving Methods**: There are various strategies for solving CSPs, including backtracking, forward checking, and constraint propagation.
- **Applications**: CSPs are widely used in many fields such as scheduling, planning, resource allocation, timetabling, and in games and puzzles like Sudoku.

Parul® University NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Constraint Satisfaction Problem

Coloring Graph Problem



	1	2	3	4
Initial Dom.	R, G, B	RGB	RGB	RGB
1=R	R	GB	GB	GB
2=G	R	G	GB	B
3=G	R	G	G	B

Constraint Satisfaction Problem

- Many AI problems can be viewed as problems of constraint satisfaction.
- For example, Crypt-arithmetic puzzle:

```
  S E N D
+ M O R E
---------
M O N E Y
```

- As compared with a straightforward search procedure, viewing a problem as one of the constraint satisfaction can substantially reduce the amount of search.
- Two-step process:
  1. Constraints are discovered and propagated as far as possible.
  2. If there is still not a solution, then search begins, adding new constraints.
- Initial state contains the original constraints given in the problem.
- A goal state is any state that has been constrained "enough". i.e. The goal is to discover some problem state that satisfies the given set of constraints.

Constraint Satisfaction Problem

Crypt-arithmetic Puzzle

Initial state:

- Assign values between 0 to 9.
- No two letters have the same value.
- The sum of the digits must be as shown.

```
  S E N D
+ M O R E
---------
M O N E Y
```

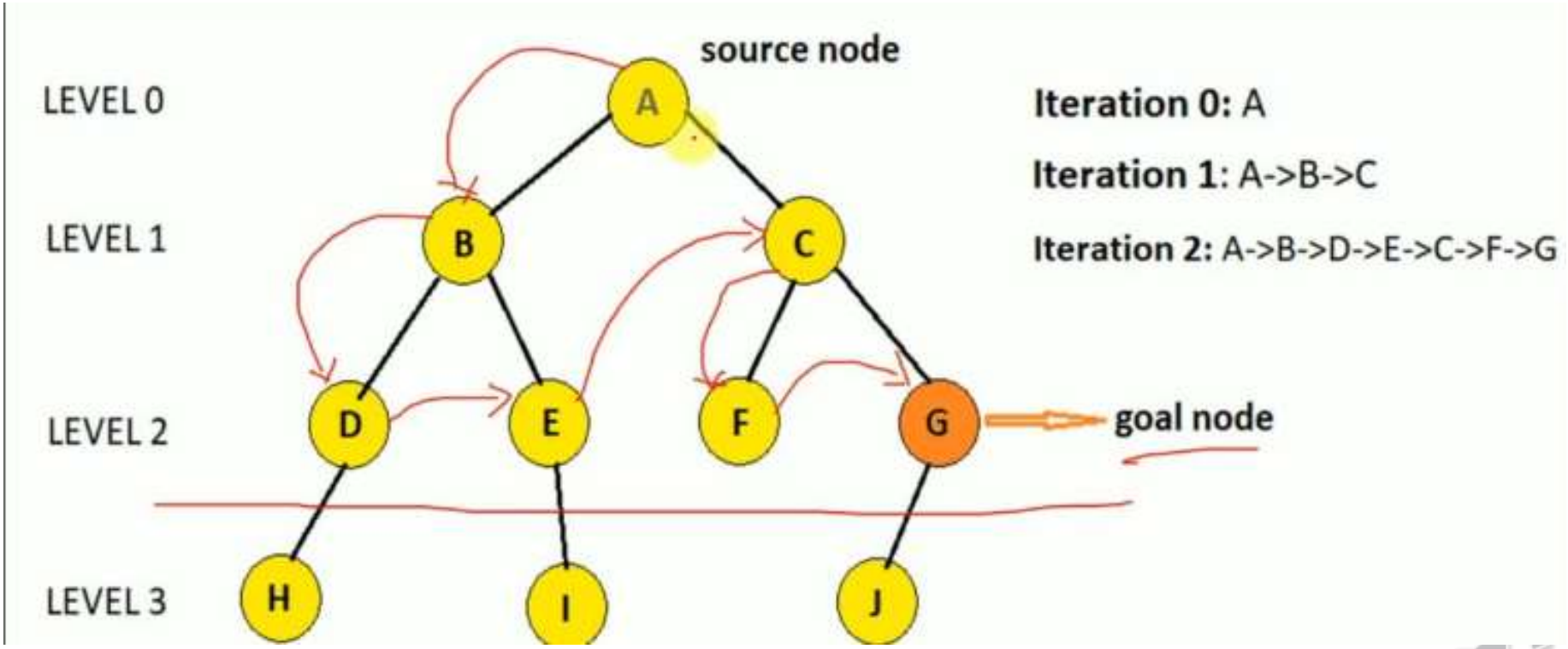
```
      1   1
  9   5   6   7
  1   0   8   5
-------------------
  1   0   6   5   2
```

S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

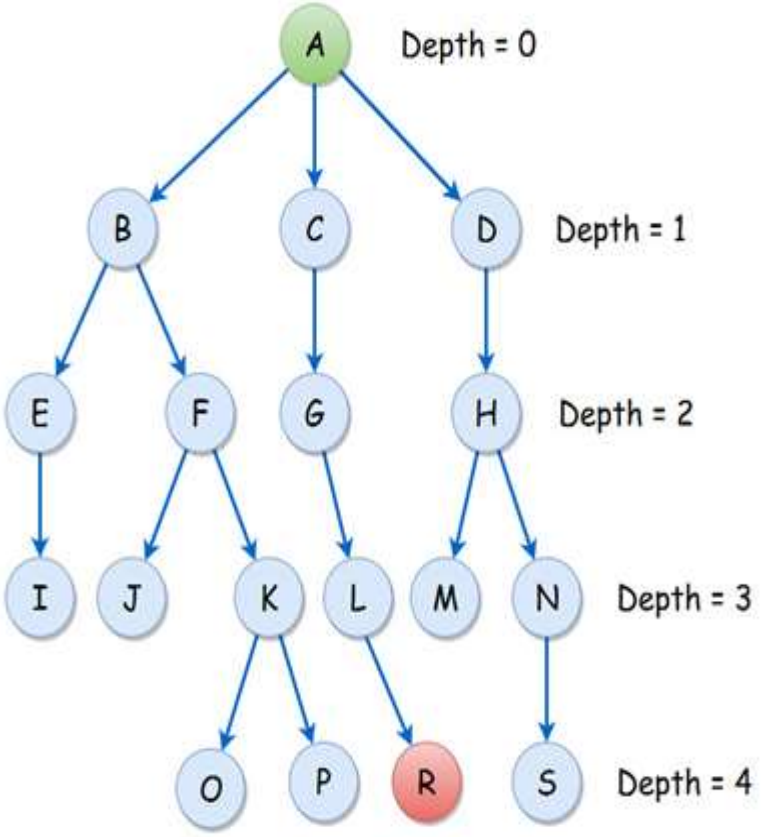
Iterative Deepening Search

- Depth first search is incomplete if there is an infinite branch in the search tree.
- Infinite branches can happen if:
  - paths contain loops
  - infinite number of states and/or operators.
- For problems with infinite state spaces, several variants of depth-first search have been developed: depth limited search, iterative deepening search.
- Depth limited search expands the search tree depth-first up to a maximum depth l.
- The nodes at depth l are treated as if they had no successors.
- Iterative deepening (depth-first) search (IDDS) is a form of depth limited search which progressively increases the bound.
- It first tries $l = 1$, then $l = 2$, then $l = 3$, etc. until a solution is found at $l = d$.

Example 1


source node
LEVEL 0
LEVEL 1
LEVEL 2
LEVEL 3
A
B
C
D
E
F
G
H
I
J
goal node
Iteration 0: A
Iteration 1: A->B->C
Iteration 2: A->B->D->E->C->F->G

Example 2



Depth	DLS Traversal
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

Means Ends Analysis(MEA)

- Collection of strategies presented so far can reason either forward or backward, but for a given problem, one direction or the other must be chosen.
- A mixture of the two directions is appropriate. Such a mixed strategy would make it possible to solve the major parts of a problem first and then go back and solve the small problems that arise in "gluing" the big pieces together.
- The technique of Means-Ends Analysis (MEA) allows us to do that.
- MEA process centers around the detection of differences between the current state and the goal state.
- Once such a difference is isolated, an operator that can reduce the difference must be found.
- If the operator cannot be applied to the current state, we set up a sub-problem of getting to a state in which it can be applied.
- The kind of backward chaining in which operators are selected and then sub-goals are set up to establish the preconditions of the operators is called operator sub-goaling

- The means-ends analysis process can be applied recursively for a problem.
- Following are the main Steps which describes the working of MEA technique for solving a problem.
  - First, evaluate the difference between Initial State and final State.
  - Select the various operators which can be applied for each difference.
  - Apply the operator at each difference, which reduces the difference between the current state and goal state.

- In the MEA process, we detect the differences between the current state and goal state.
- Once these differences occur, then we can apply an operator to reduce the differences.
- But sometimes it is possible that an operator cannot be applied to the current state.

Algorithm- Means Ends Analysis

1. *Compare CURRENT to GOAL. If there are no differences between them then return.*
2. *Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled:*
   a. *Select an as yet untried operator O that is applicable to the current difference. If there are no such operators, then signal failure.*
   b. *Attempt to apply O to CURRENT. Generate descriptions of two states: O-START, a state in which O's preconditions are satisfied and O-RESULT, the state that would result if O were applied in O-START.*
   c. *If*

      (FIRST-PART ← MEA(CURRENT, O-START))

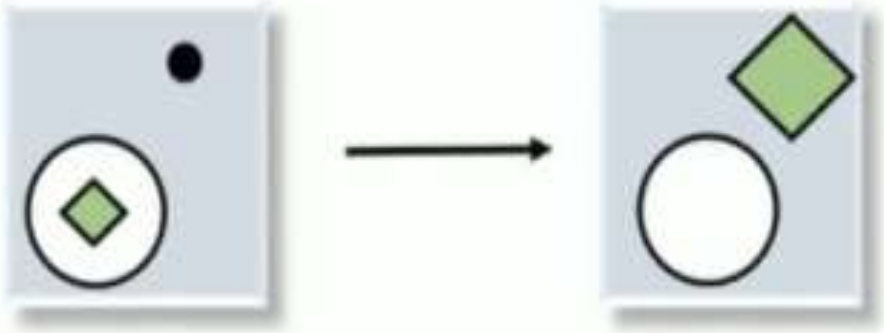
      and

      (LAST-PART ← MEA(O-RESULT, GOAL))

      are successful, then signal success and return the result of concatenating FIRST-PART, O, and LAST-PART.

Example

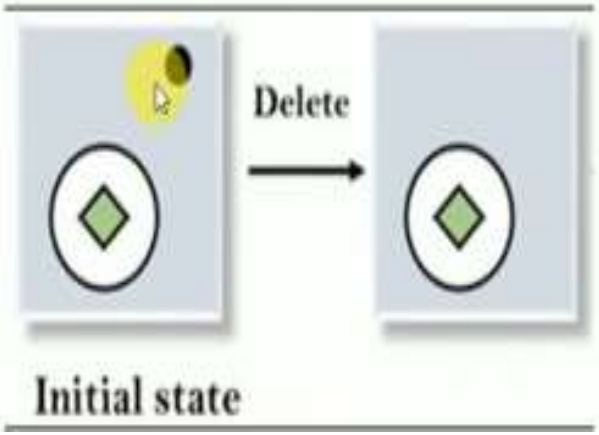
- Let's take an example to understand how MEA algorithm works.
- Lets assume the initial state and goal state as given below.



- To solve the given problem, we need to find the differences between initial state and goal state, and for each of these differences, we will apply an operator to generate a new state.
- The operators we have for this problem are:
  - Move – Move the object diamond outside the circle
  - Delete – Delete the black circle
  - Expand – Expand or increase the size of the diamond

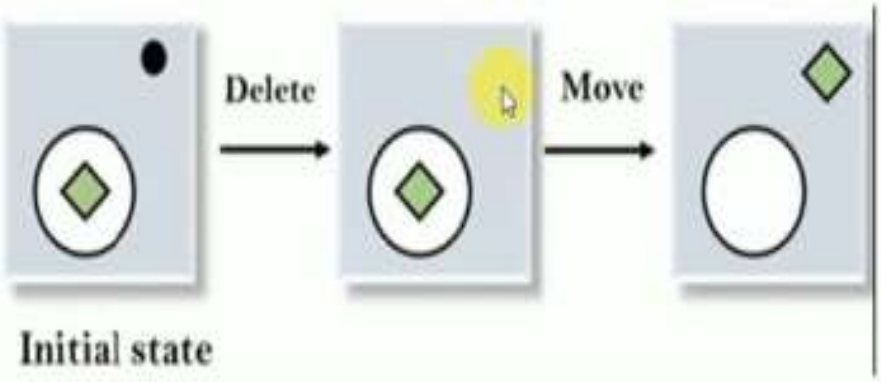
2. Applying Delete operator:

As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the Delete operator to remove this dot.



3. Applying Move Operator:

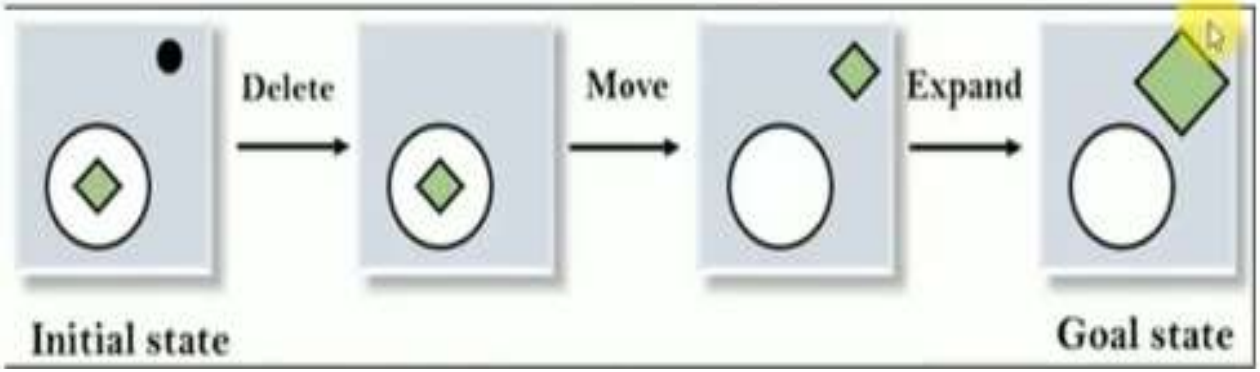
After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the Move Operator.



4. Applying Expand Operator:

Now a new state is generated in the third step, and we will compare this state with the goal state.

After comparing the states there is still one difference which is the size of the square, so, we will apply Expand operator, and finally, it will generate the goal state.



Parul University

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in