

---

# ARTIFICIAL INTELLIGENCE

---



---

# Informed Search

1. Generate-and-test
  2. Hill climbing
  3. Best-first search
  4. Problem reduction ( $A^*/AO^*$ )
  5. Constraint satisfaction
  6. Means-ends analysis
-

---

# Heuristic Search

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.
-

$$h(n) \leq h^*(n)$$

- 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.
- The value of the heuristic function is always positive.
- Can be solved using three methods :
  1. Euclidian distance
  2. Manhattan distance
  3. No. of misplaced tiles

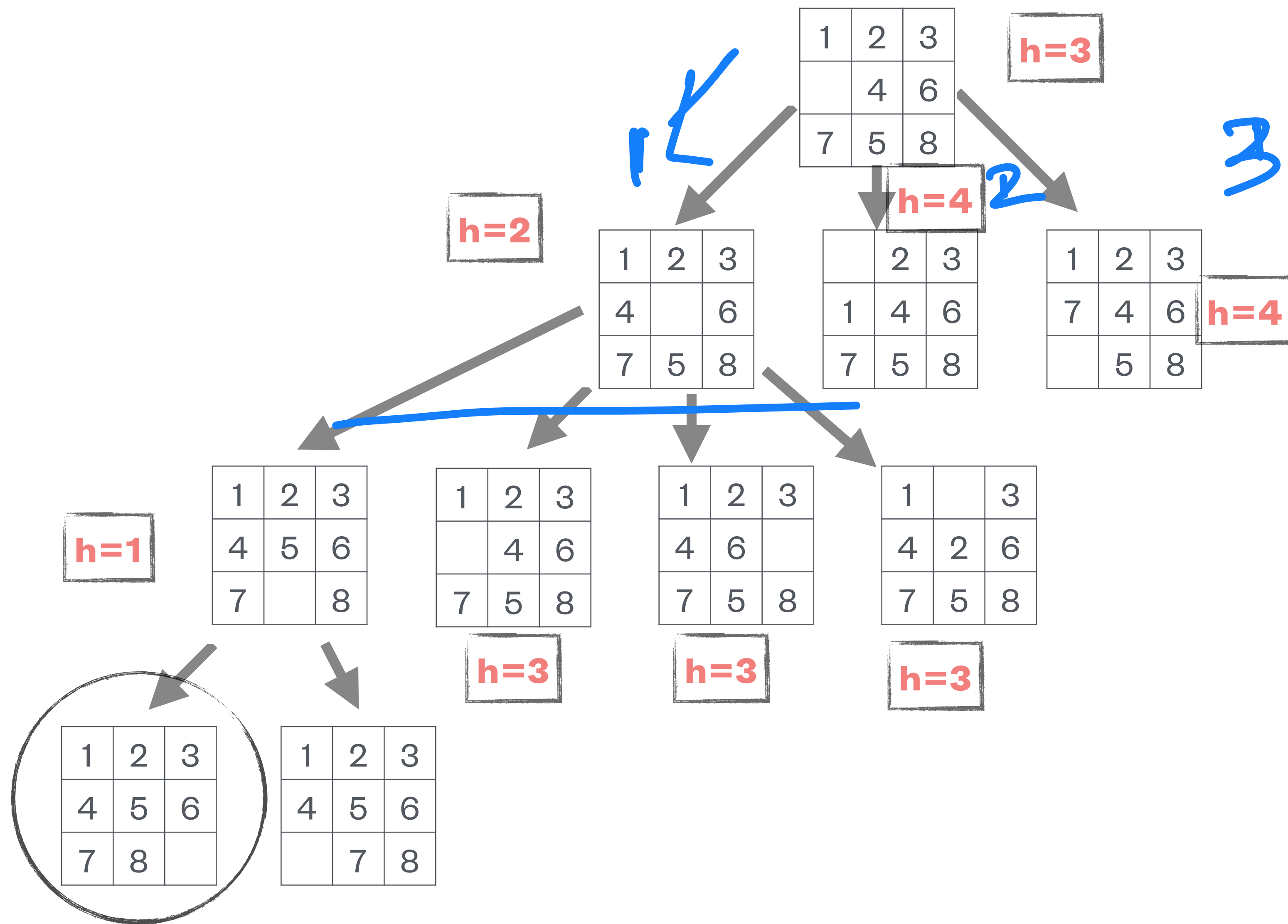
St. Dist

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

---

# 8-puzzle problem with heuristic

---



---

# Generate and Test

## **Algorithm:**

1. Generate a possible solution.
2. Test to see if this is actually a solution.
3. Quit if a solution has been found. Otherwise, return to step 1.

## **Advantages**

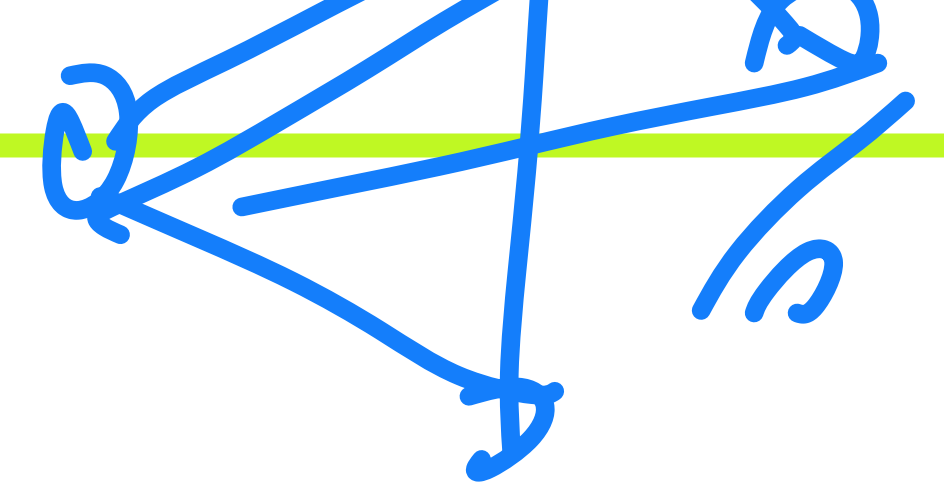
- Complete
- Non Redundant

## **Disadvantage**

- Acceptable for simple problems.
  - Inefficient for problems with large space.
-

---

# Hill Climbing



- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbour has a higher value.
  - Hill climbing algorithm is a technique which is used for optimising the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimise the distance traveled by the salesman.
  - It is also called greedy local search as it only looks to its good immediate neighbour state and not beyond that.
  - A node of hill climbing algorithm has two components which are state and value.
  - Hill Climbing is mostly used when a good heuristic is available.
  - In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.
-



---

# Features:

Following are some main features of Hill Climbing Algorithm:

1. **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
  2. **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimises the cost.
  3. **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.
-

---

# Types :

1. Simple Hill Climbing
2. Steepest-Ascent hill-climbing

The **closest node** is chosen in a **simple hill climbing** algorithm while the node **closest to the solution** is identified and chosen in a **steepest ascent hill climbing algorithm**

---

---

# Simple Hill climbing


- Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimises current cost and set it as a current state.** It only checks its one successor state, and if it finds better than the current state, then move else be in the same state.
- Algorithm for Simple Hill Climbing:

**Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.

**Step 2:** Loop Until a solution is found or there is no new operator left to apply.

**Step 3:** Select and apply an operator to the current state.

**Step 4:** Check new state:

- If it is goal state, then return success and quit. 
- Else if it is better than the current state then assign new state as a current state.
- Else if not better than the current state, then return to step2.

**Step 5:** Exit.

---



---

# Steepest-Ascent Algorithm for Hill Climbing

- Evaluate the initial state. If it is the goal state then return and quit. Otherwise continue with initial state as current state.
  - Loop until a solution is found or until there are no new operators left to be applied to the current state:
    - Select operator that has not been applied to the current state and apply it to produce the new state.
    - Evaluate the new state
  - Key Points :
    - (i) If it is the goal state, then return 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.
-

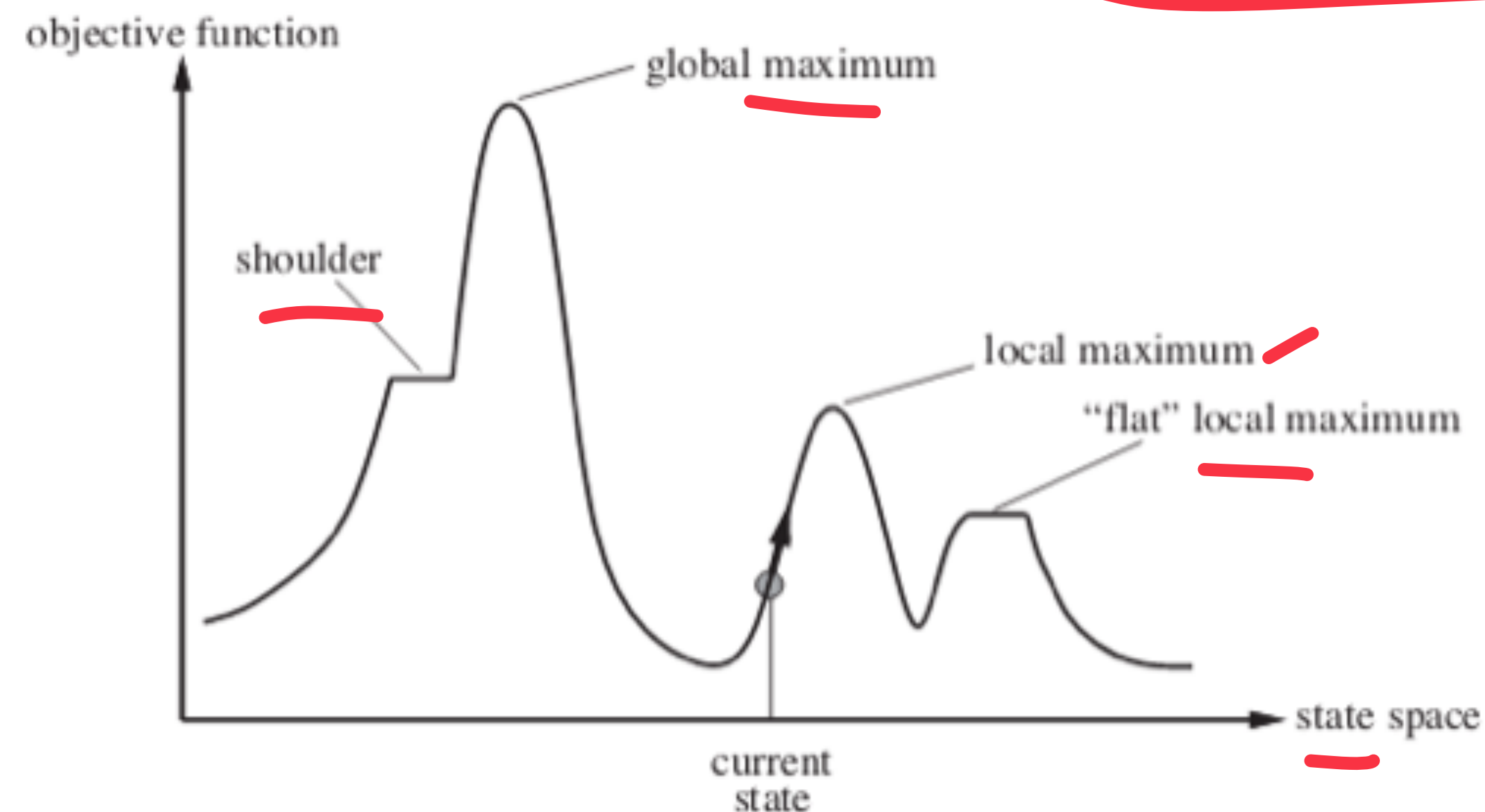
# State Space diagram for Hill Climbing

- State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximise).

**X-axis :** denotes the state space ie states or configuration our algorithm may reach.

**Y-axis :** denotes the values of objective function corresponding to a particular state.

The best solution will be that state space where objective function has maximum value(global maximum)



- 
- **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.
  - **Global maximum :** It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
  - **Plateau/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
  - **Ridge :** It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
  - **Current state :** The region of state space diagram where we are currently present during the search.
  - **Shoulder :** It is a plateau that has an uphill edge.
-

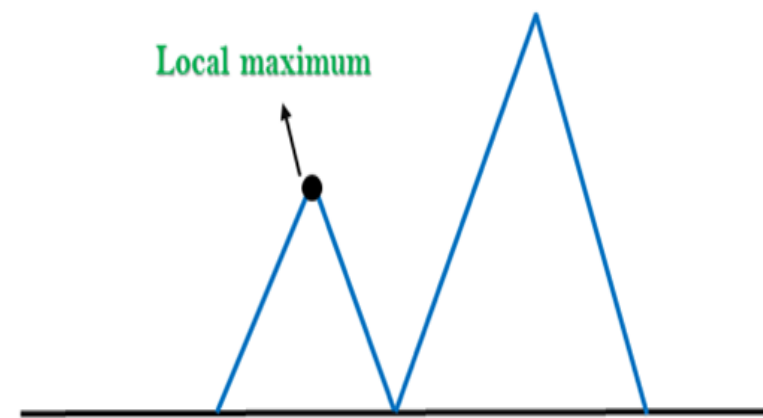


---

# Problems:

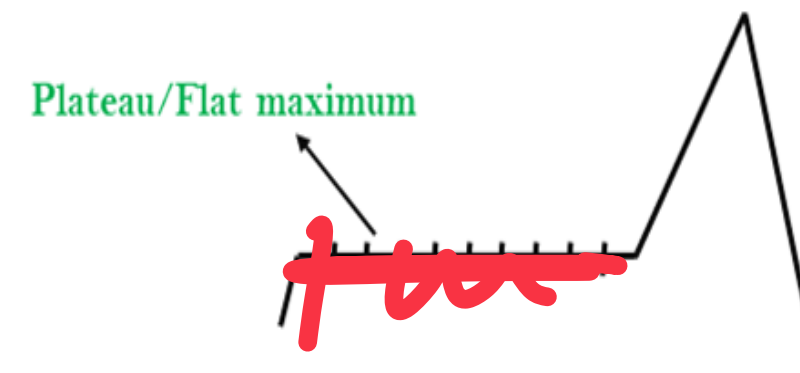
1. Local Maximum: A state that is better than all its neighbours but is not better than some other states farther away.

**To overcome the local maximum problem:** Utilise the **backtracking technique**. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.



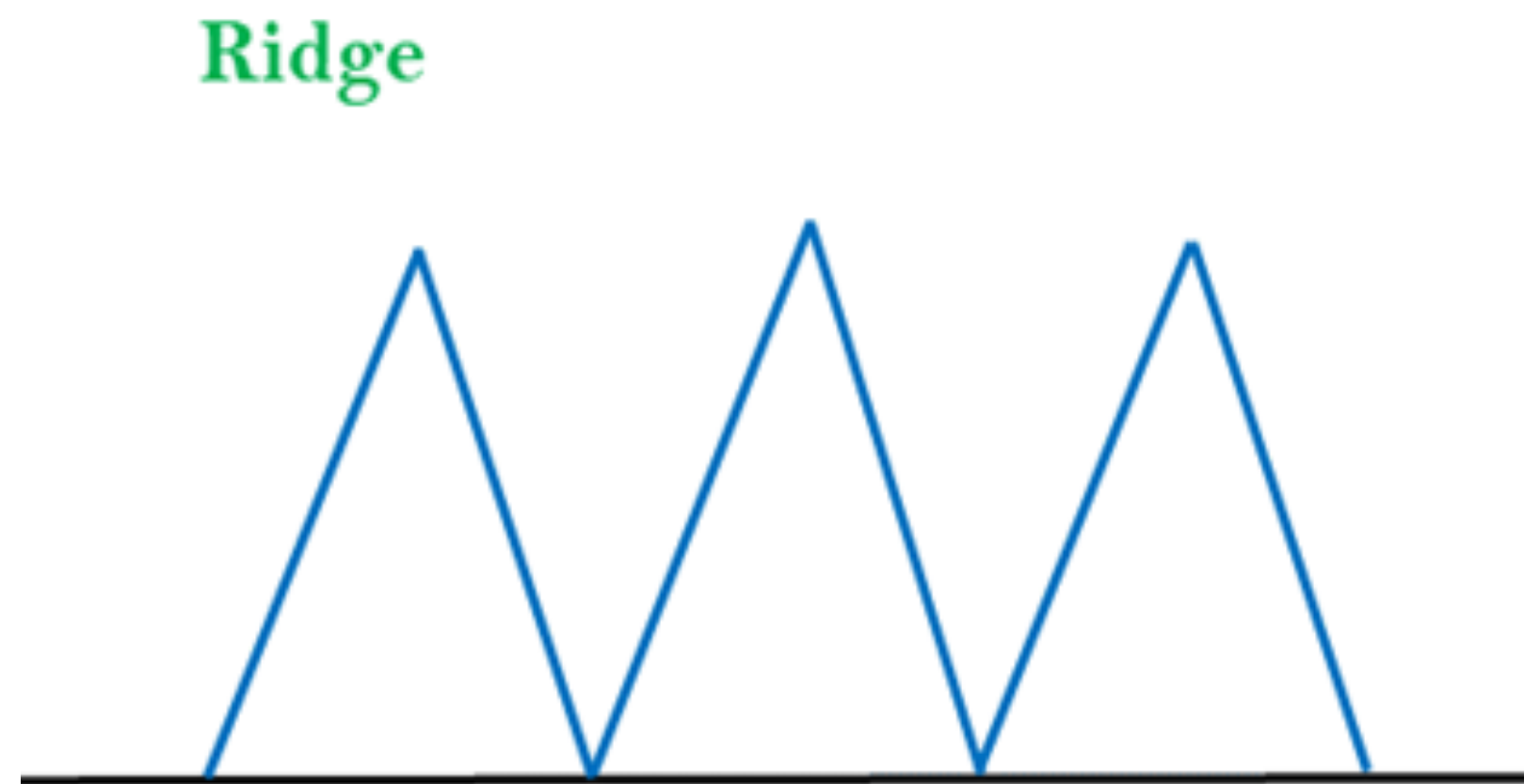
2. Plateau: Is a flat area of the search space in which a whole set of neighbouring states have the same value.

**To overcome plateaus:** Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region



- 
3. Ridge: A special kind of local maximum. An area of the search space that is higher than surrounding areas and that itself has slope .

**To overcome Ridge:** You could use two or more rules before testing. It implies moving in several directions at once.



# Example :

start

4	1	2
3		5
6	7	8

1

2

4		2
3	1	5
6	7	8

2

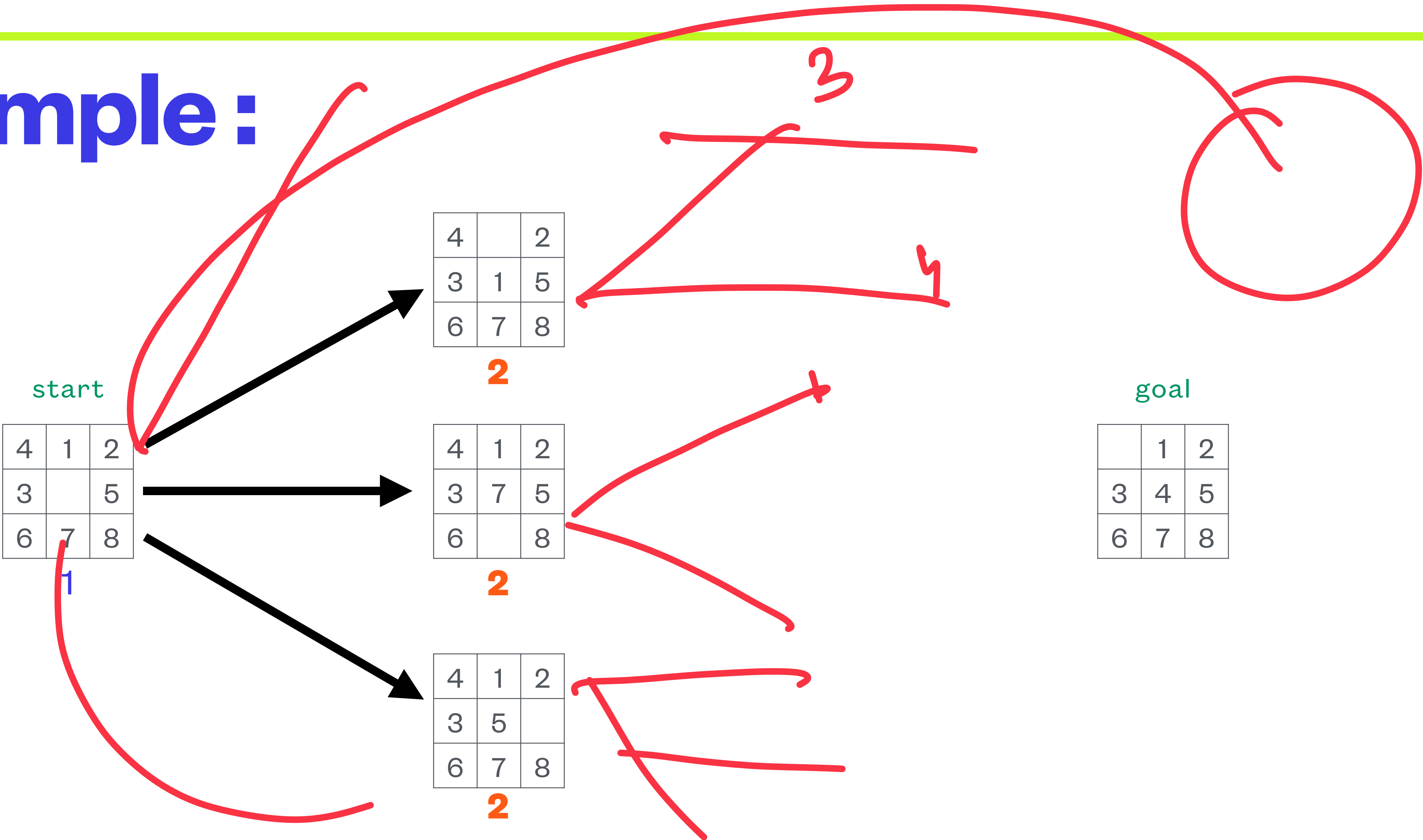
4	1	2
3	7	5
6		8

2

4	1	2
3	5	
6	7	8

goal

	1	2
3	4	5
6	7	8





---

# Global search techniques:

1. Best First Search(OR graph) : Where not only the current branch of the search space but all the so far explored nodes/states in the search space are considered in determining the next best state/node.
  2. A\* Algorithm : Which is improvised version of Best first Search.
  3. Problem Reduction and And-Or Graphs : AO\* Algorithm.
  4. Constraint Satisfaction Problem (CSP) : Graph Colouring Problem and Crypt Arithmetic Problems.
  5. Mean End Analysis (MEA)
-

---

# Best-first 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.
- With the help of best-first search, at each step, we can choose the most promising node.
- The greedy best first algorithm is implemented by the priority queue.
- 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.

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

*main focus*

$h(n)$  = estimated cost from node  $n$  to the goal

---

---

# Difference from hill climbing

- In hill climbing at each step one node is selected and all others are rejected and never considered again. While in Best-first search one node is selected and all others are kept around so that they may be revisited again.
  - Best available state is selected even if it may have a value lower than the currently expanded state.
  - Implementation of the best first search requires the following :
    1. OPEN- all those nodes that have been generated & have had heuristic function applied to them but have not yet been examined.
    2. CLOSED- contains all nodes that have already been examined.
-



---

# Algorithm :

**Step 1:** Place the starting node into the OPEN list.

**Step 2:** If the OPEN list is empty, Stop and return failure.

**Step 3:** Remove the node  $n$ , from the OPEN list which has the lowest value of  $h(n)$ , and places it in the CLOSED list.

**Step 4:** Expand the node  $n$ , and generate the successors of node  $n$ .

**Step 5:** Check each successor of node  $n$ , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

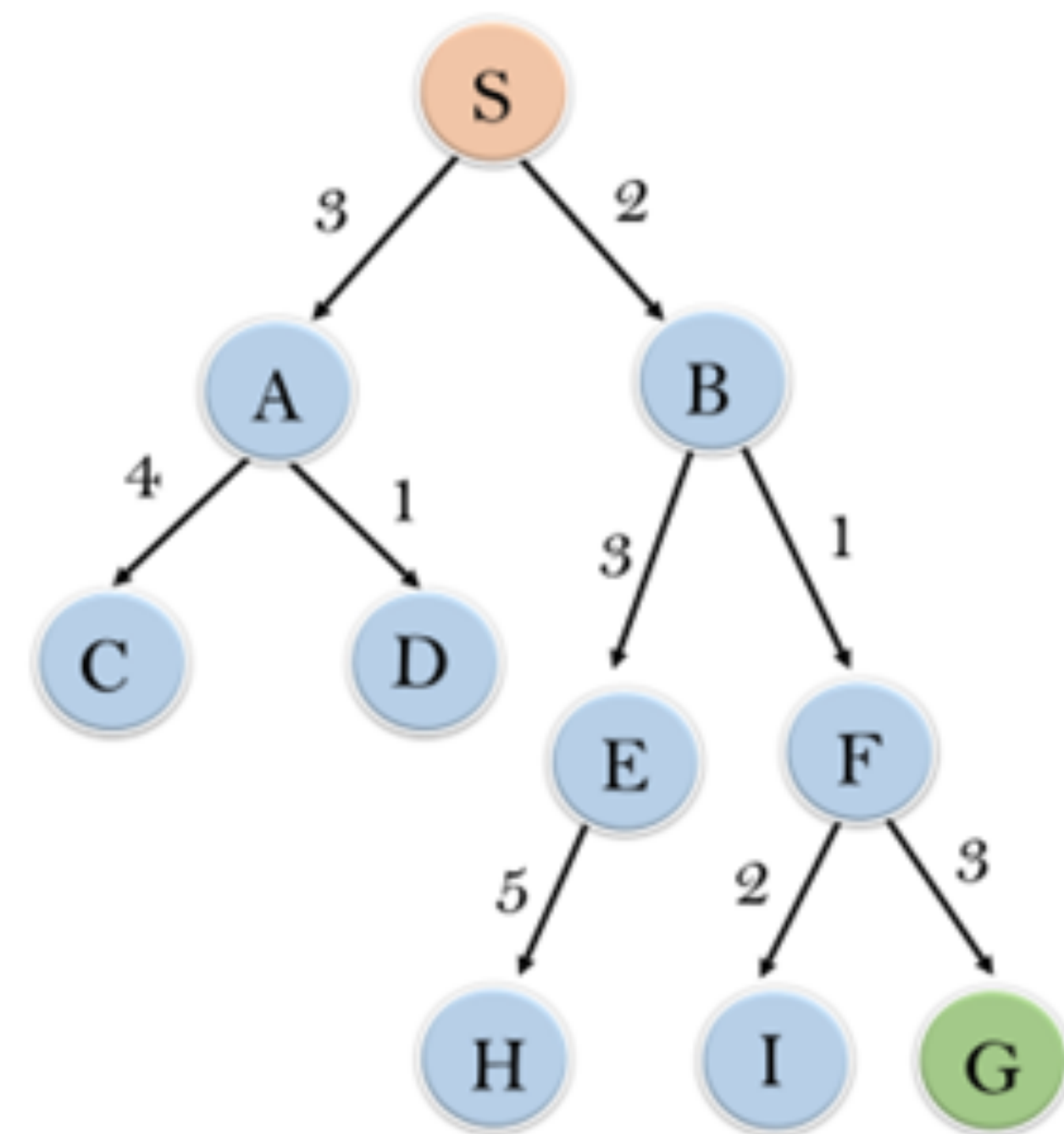
**Step 6:** For each successor node, algorithm checks for evaluation function  $f(n)$ , and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

**Step 7:** Return to Step 2.

---

# Example :

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function  $f(n)=h(n)$  a



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

already given

- In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.
- Expand the nodes of S and put in the CLOSED list

**Initialization:** Open [A, B], Closed [S]

**Iteration 1:** Open [A], Closed [S, B]

**Iteration 2:** Open [E, F, A], Closed [S, B]

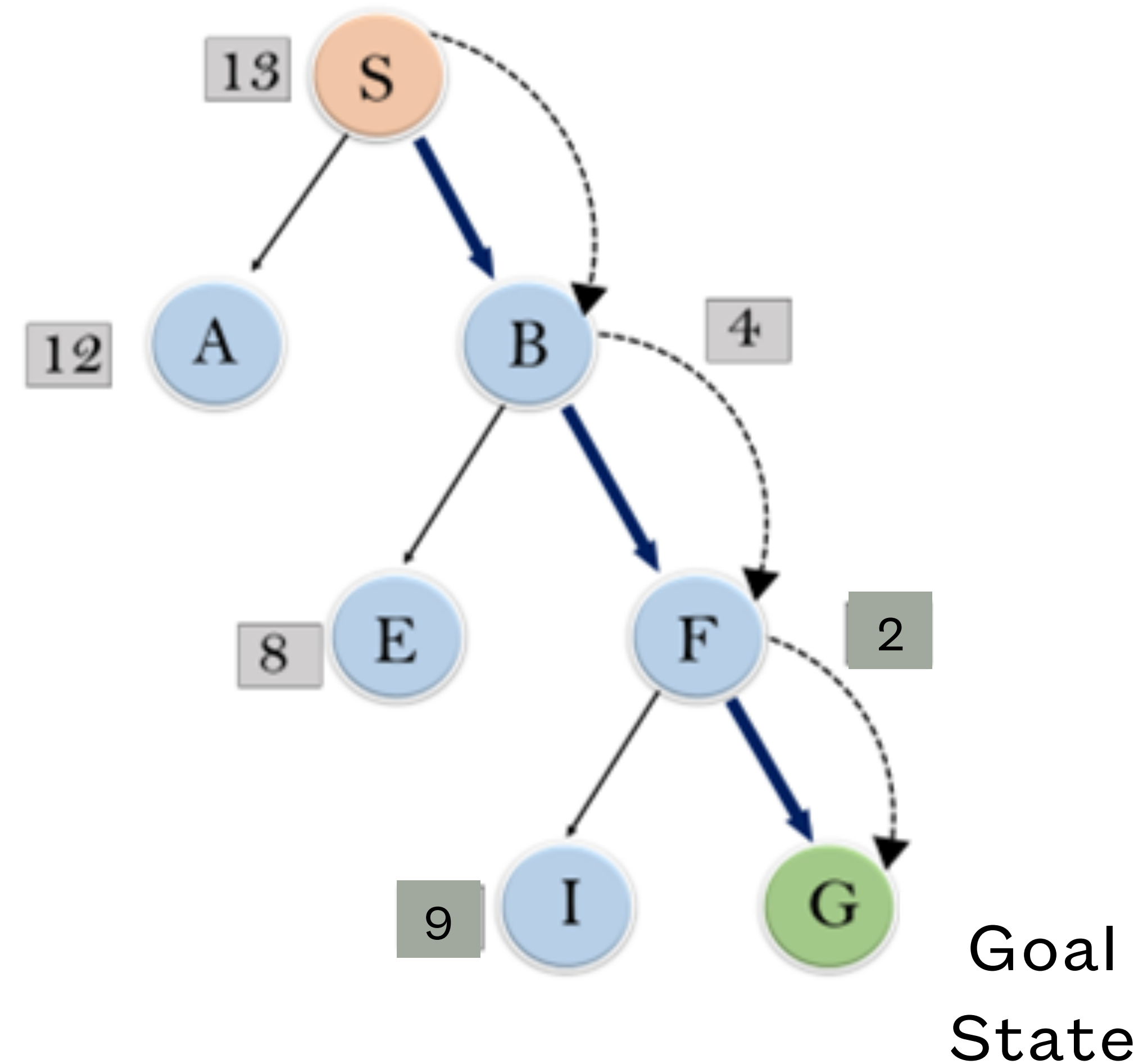
: Open [E, A], Closed [S, B, F]

**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**



---

# Properties :

- **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,  $m$  is the maximum depth of the search space.
  - **Complete:** Greedy best-first search is also incomplete, even if the given state space is finite.
  - **Optimal:** Greedy best first search algorithm is not optimal.
-



---

- **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.
-

---

# A\* search algorithm

- A\* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently.
- It is essentially a best first search algorithm.
- A\* Algorithm extends the path that minimises the following function-

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

'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

---

---

# Working:

- 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.
-

---

# Algorithm :

- **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 :

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.

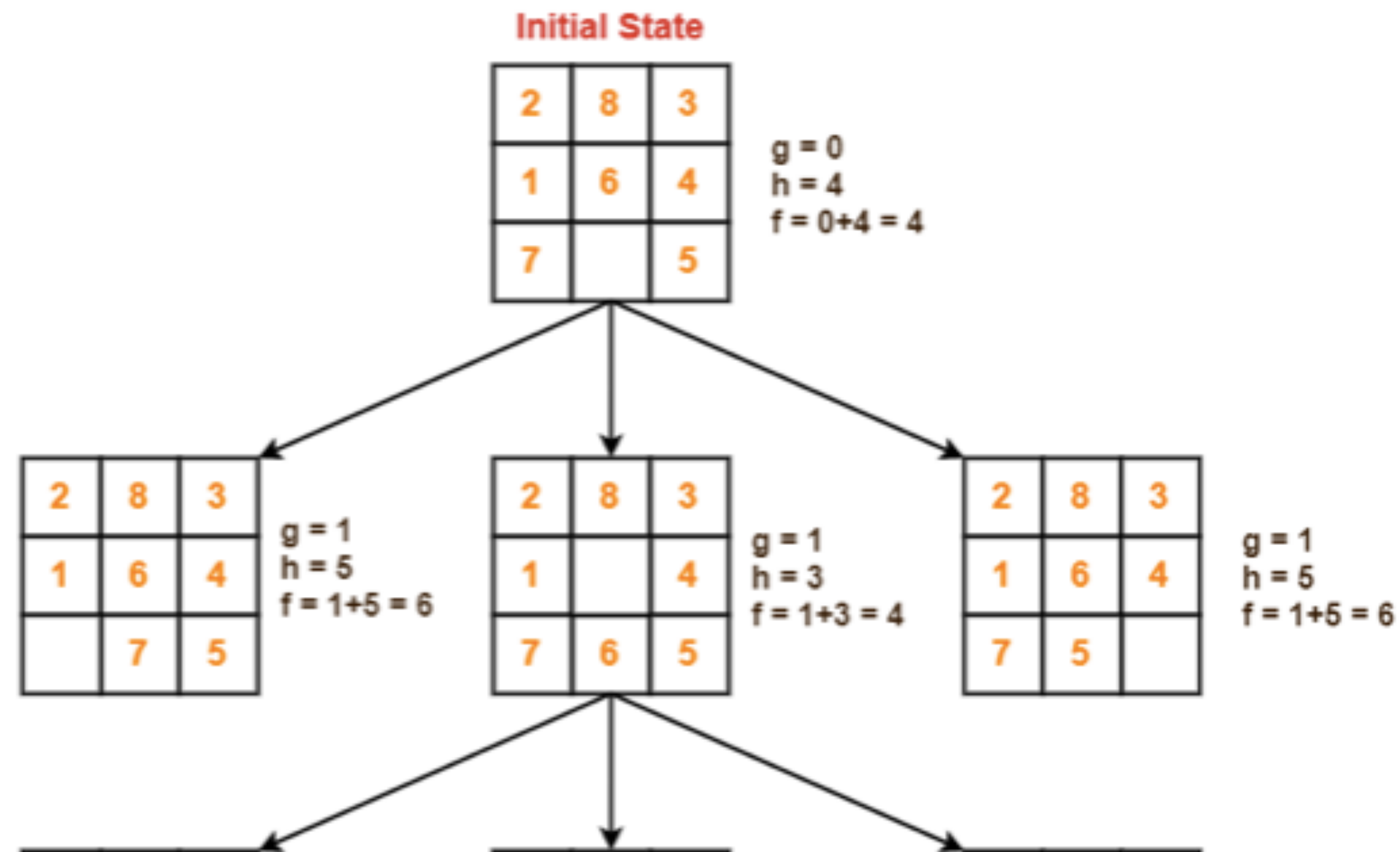
Consider  $g(n)$  = Depth of node and  $h(n)$  = Number of misplaced tiles.

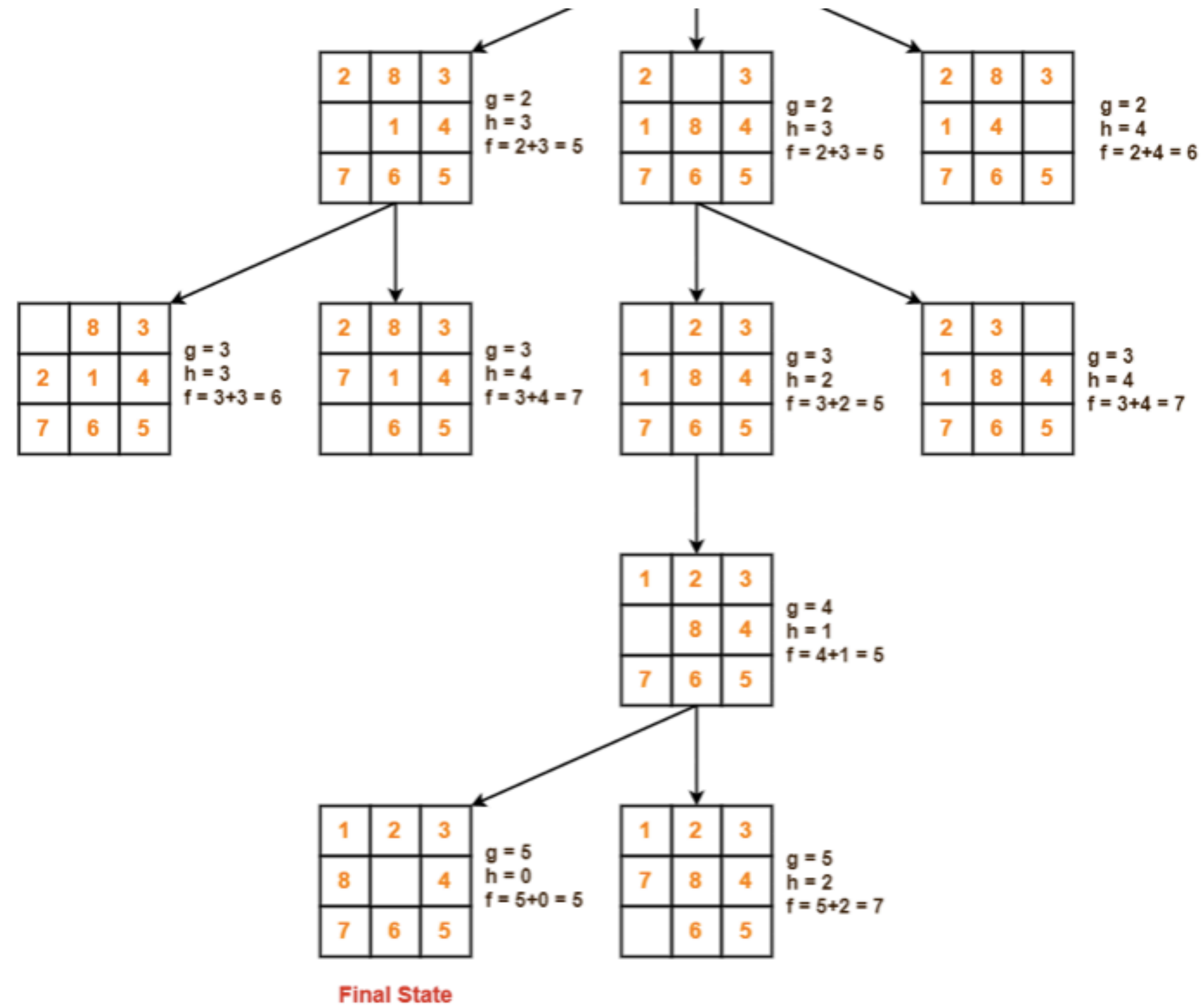
---



# Solution :

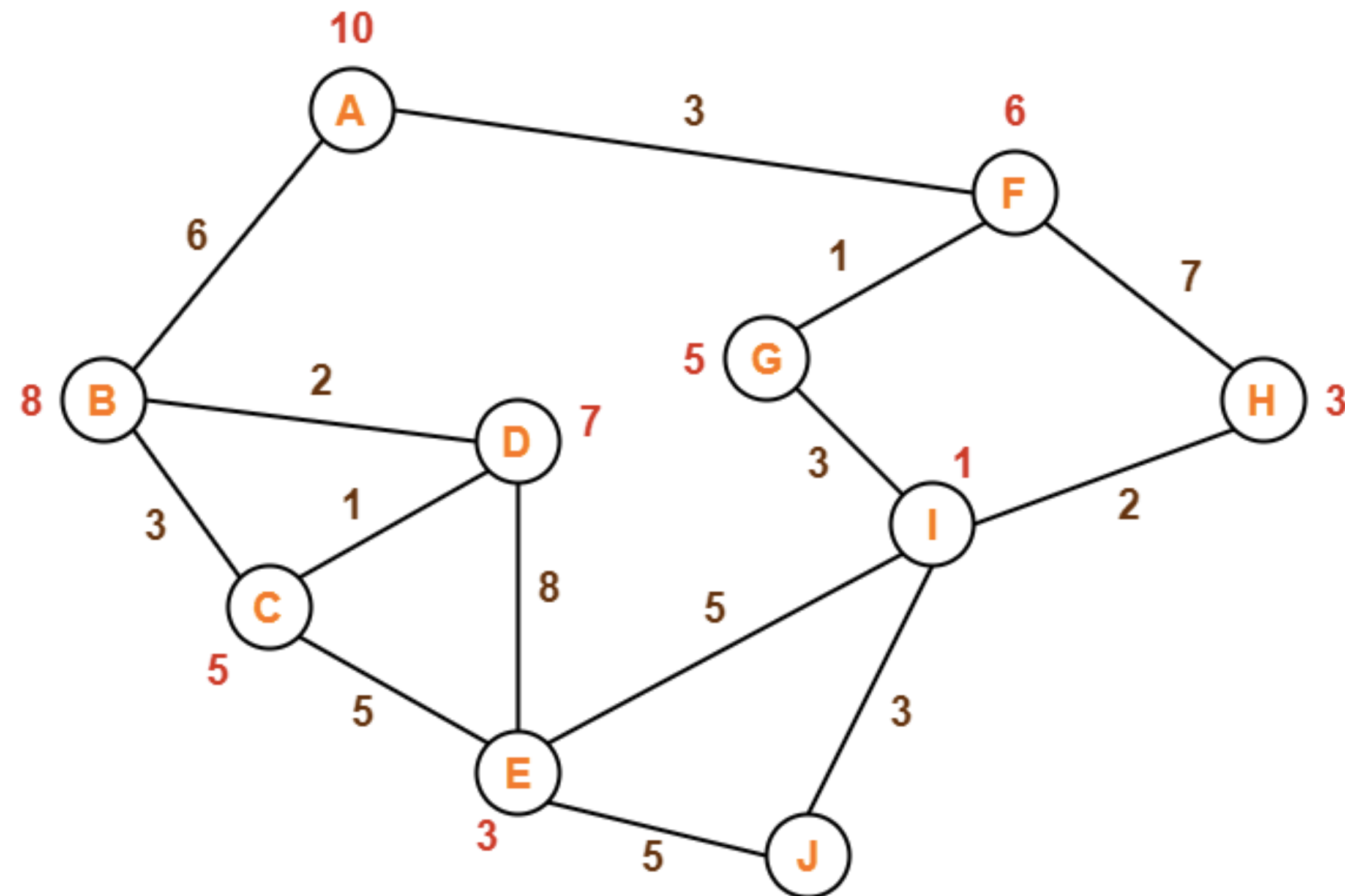
- A\* Algorithm maintains a tree of paths originating at the initial state.
- It extends those paths one edge at a time.
- It continues until final state is reached.





# Example 2:

- Consider the following graph-



- The numbers written on edges represent the distance between the nodes.
- The numbers written on nodes represent the heuristic value.
- Find the most cost-effective path to reach from start state A to final state J using A\* Algorithm.
- Solution-

### **Step-01:**

- We start with node A.
- Node B and Node F can be reached from node A.

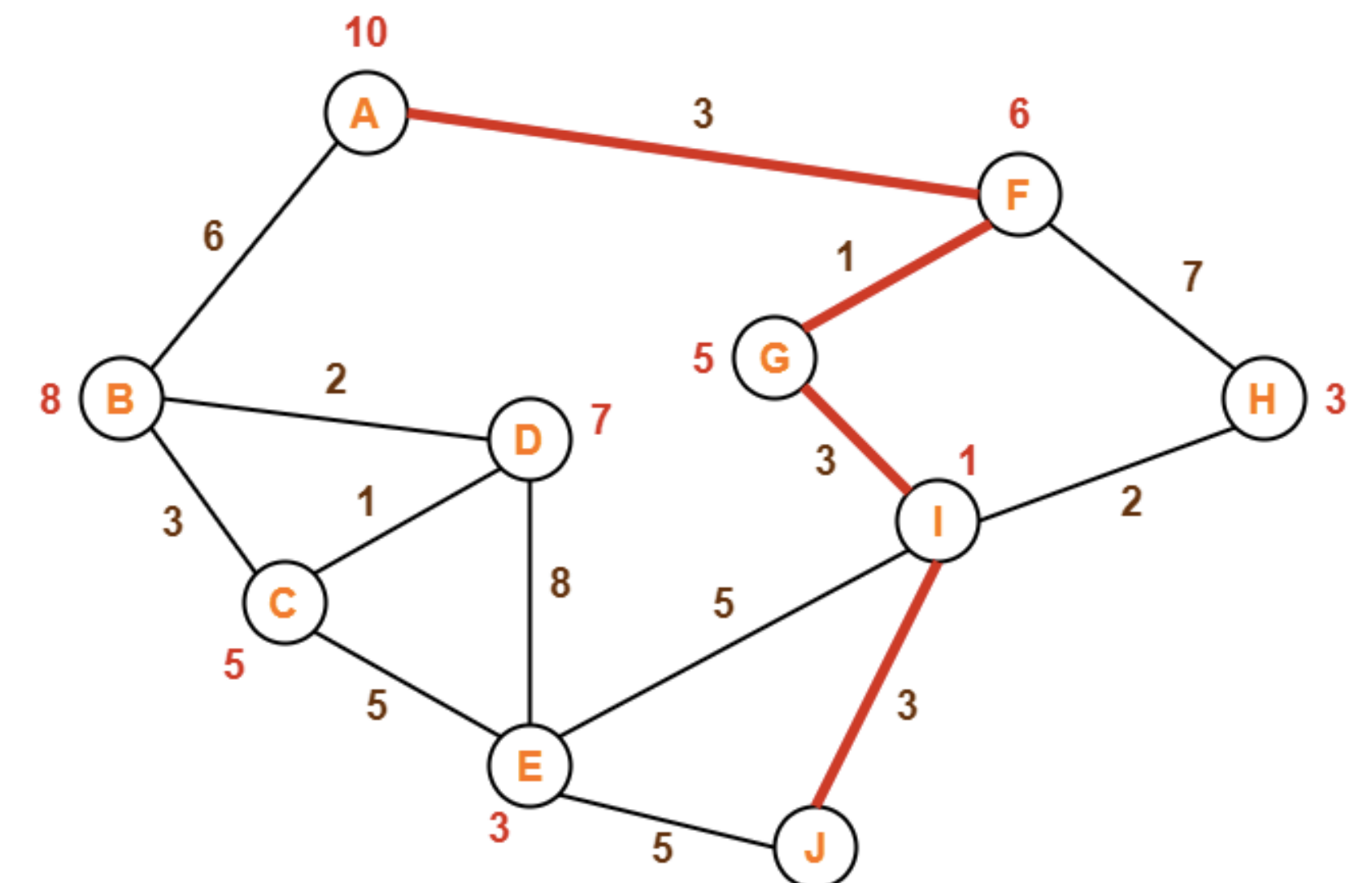
A\* Algorithm calculates  $f(B)$  and  $f(F)$ .

- $f(B) = 6 + 8 = 14$
- $f(F) = 3 + 6 = 9$

Since  $f(F) < f(B)$ , so it decides to go to node F.

**Path- A → F**

This is the required shortest path from node A to node J.



- **Step-02:**

Node G and Node H can be reached from node F.

A\* Algorithm calculates  $f(G)$  and  $f(H)$ .

- $f(G) = (3+1) + 5 = 9$
- $f(H) = (3+7) + 3 = 13$

Since  $f(G) < f(H)$ , so it decides to go to node G.

**Path-  $A \rightarrow F \rightarrow G$**

- **Step-03:**

Node I can be reached from node G.

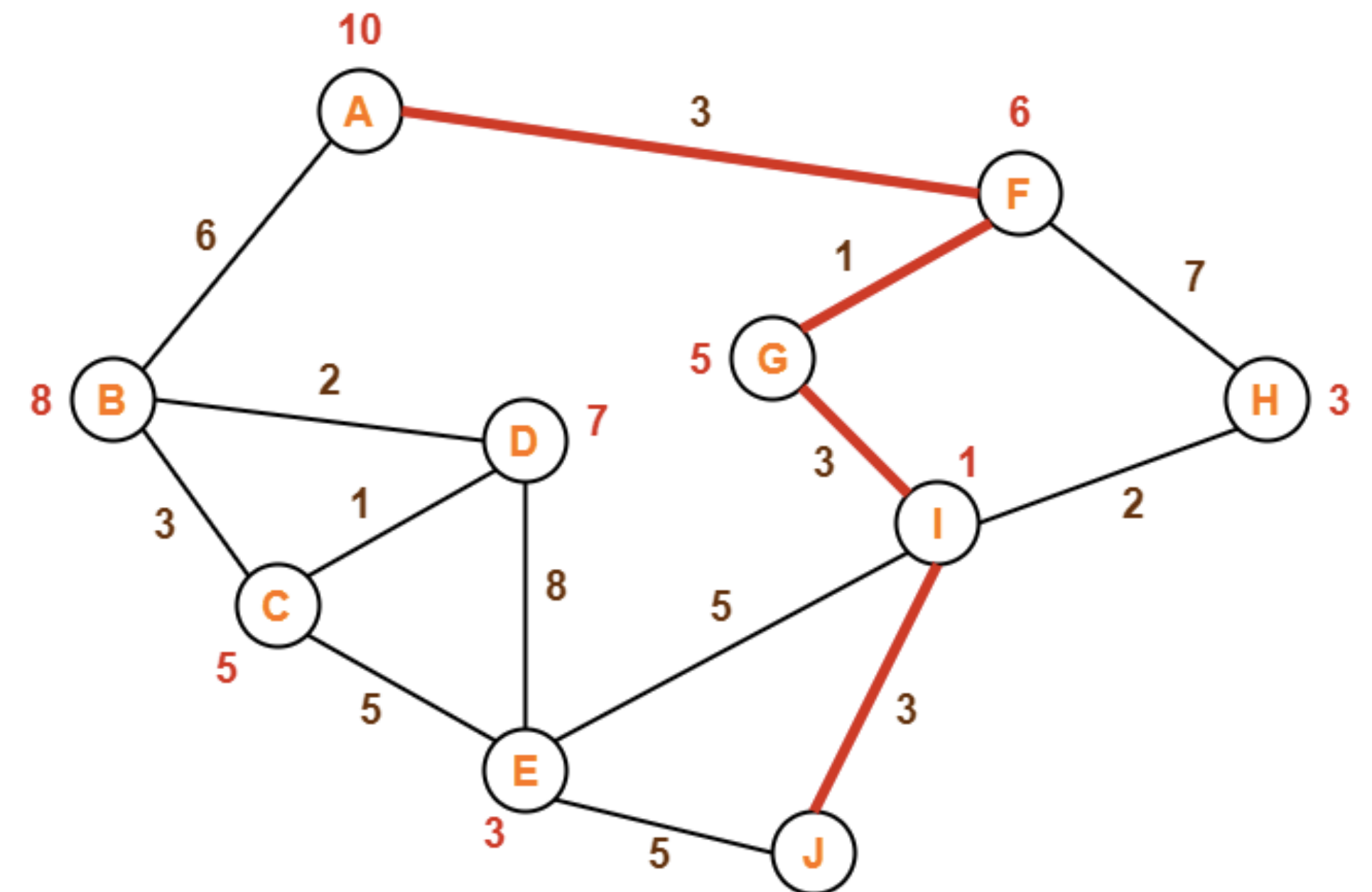
A\* Algorithm calculates  $f(I)$ .

$$f(I) = (3+1+3) + 1 = 8$$

It decides to go to node I.

**Path-  $A \rightarrow F \rightarrow G \rightarrow I$**

This is the required shortest path from node A to node J.





- Step-04:

Node E, Node H and Node J can be reached from node I.

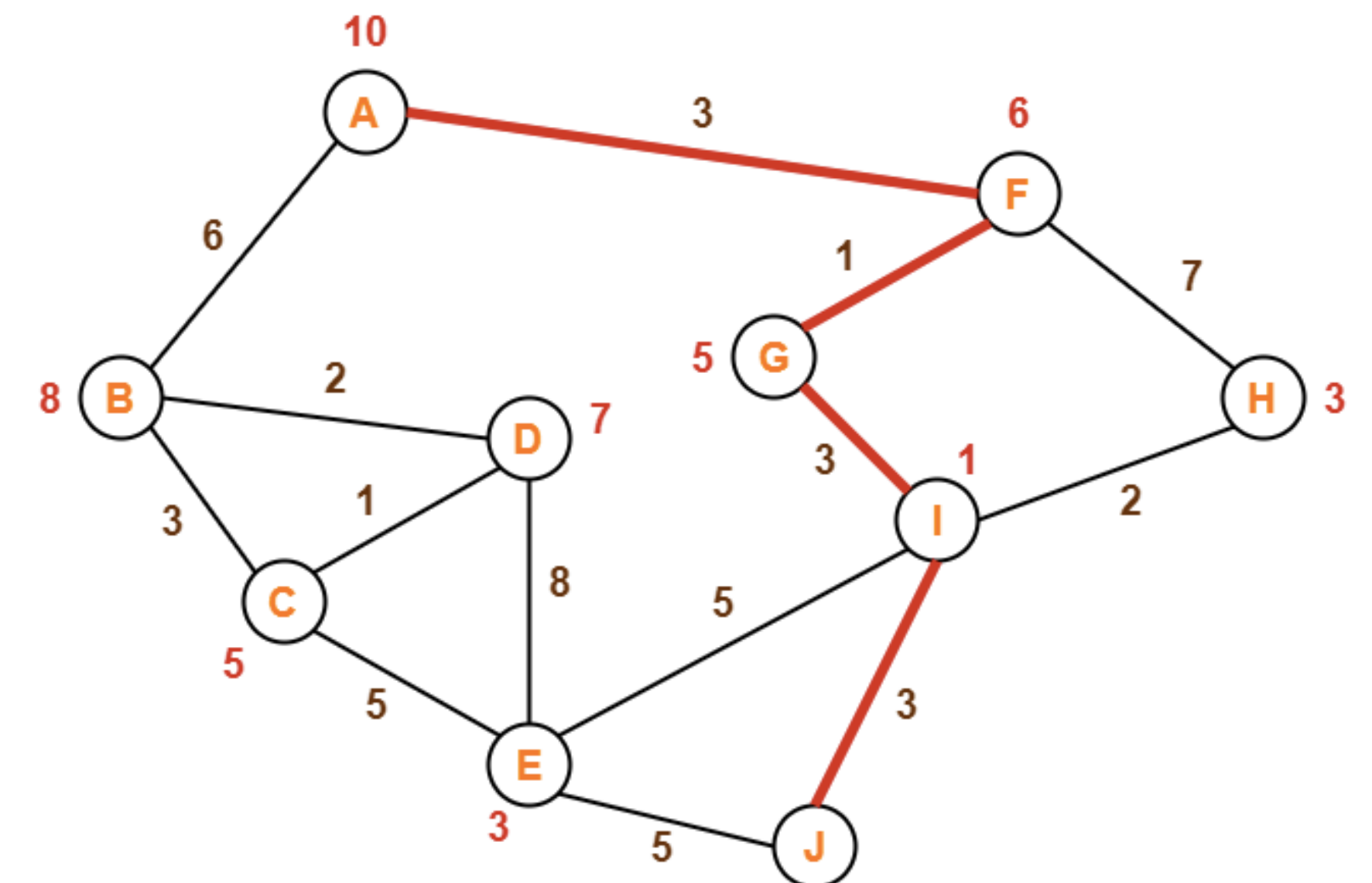
A\* Algorithm calculates  $f(E)$ ,  $f(H)$  and  $f(J)$ .

- $f(E) = (3+1+3+5) + 3 = 15$
- $f(H) = (3+1+3+2) + 3 = 12$
- $f(J) = (3+1+3+3) + 0 = 10$

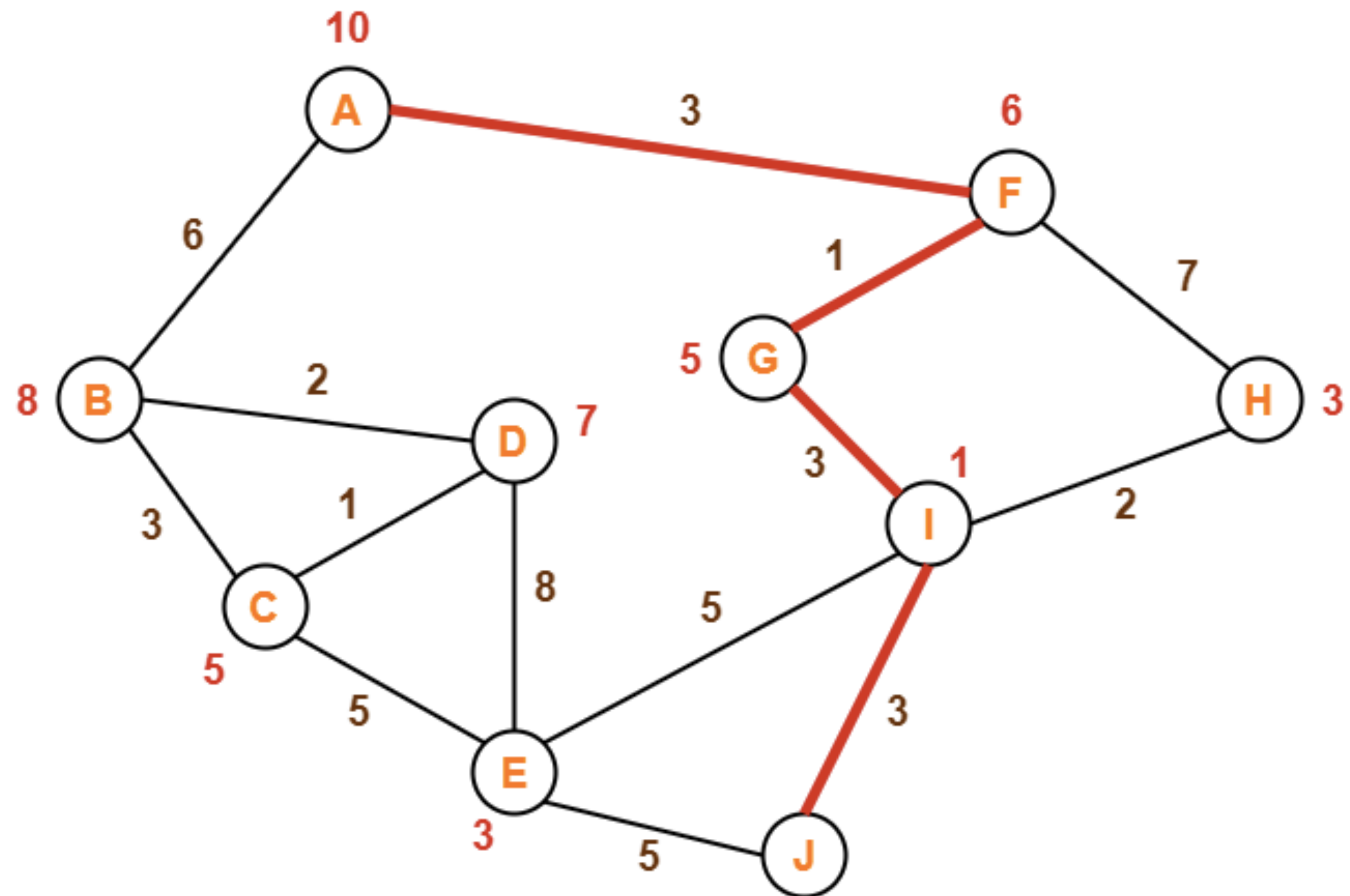
Since  $f(J)$  is least, so it decides to go to node J.

**Path-  $A \rightarrow F \rightarrow G \rightarrow I \rightarrow J$**

This is the required shortest path from node A to node J.



This is the required shortest path from node A to node J.



---

# Properties :

- **Complete:** A\* algorithm is complete as long as:
    - Branching factor is finite.
    - Cost at every action is fixed.
  - **Optimal:** A\* search algorithm is optimal if it follows below two conditions:
    - **Admissible:** the first condition requires for optimality is that  $h(n)$  should be an admissible heuristic for A\* tree search. An admissible heuristic is optimistic in nature.
    - **Consistency:** Second required condition is consistency for only A\* graph-search.  
  
(If the heuristic function is admissible, then A\* tree search will always find the least cost path)
  - **Time Complexity:** The time complexity of A\* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution  $d$ . So the time complexity is  $O(b^d)$ , where  $b$  is the branching factor.
  - **Space Complexity:** The space complexity of A\* search algorithm is  **$O(b^d)$**
-

---

*Admissible*

- **Advantages:**

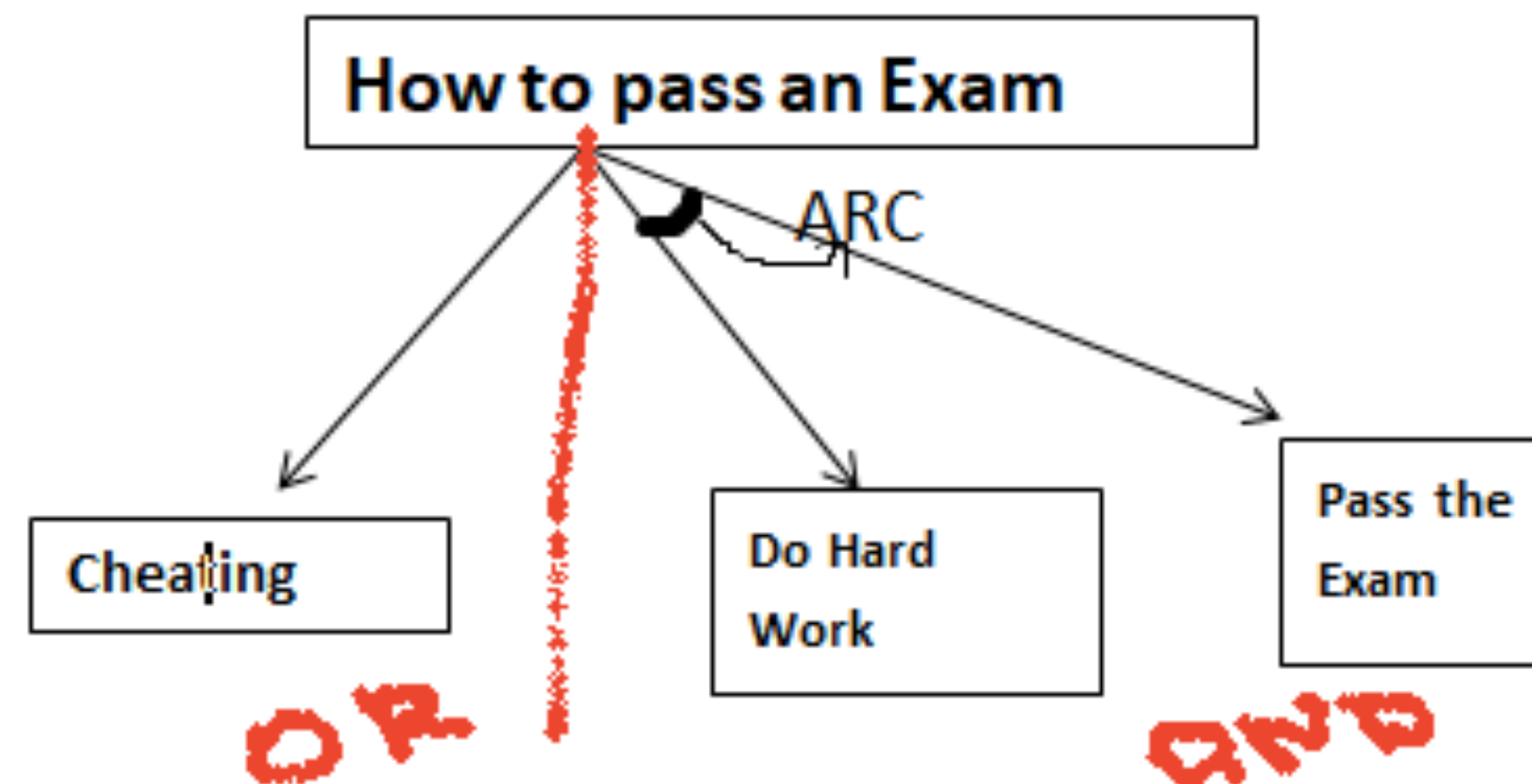
- A\* search algorithm is the best algorithm than other search algorithms.
- A\* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

- **Disadvantages:**

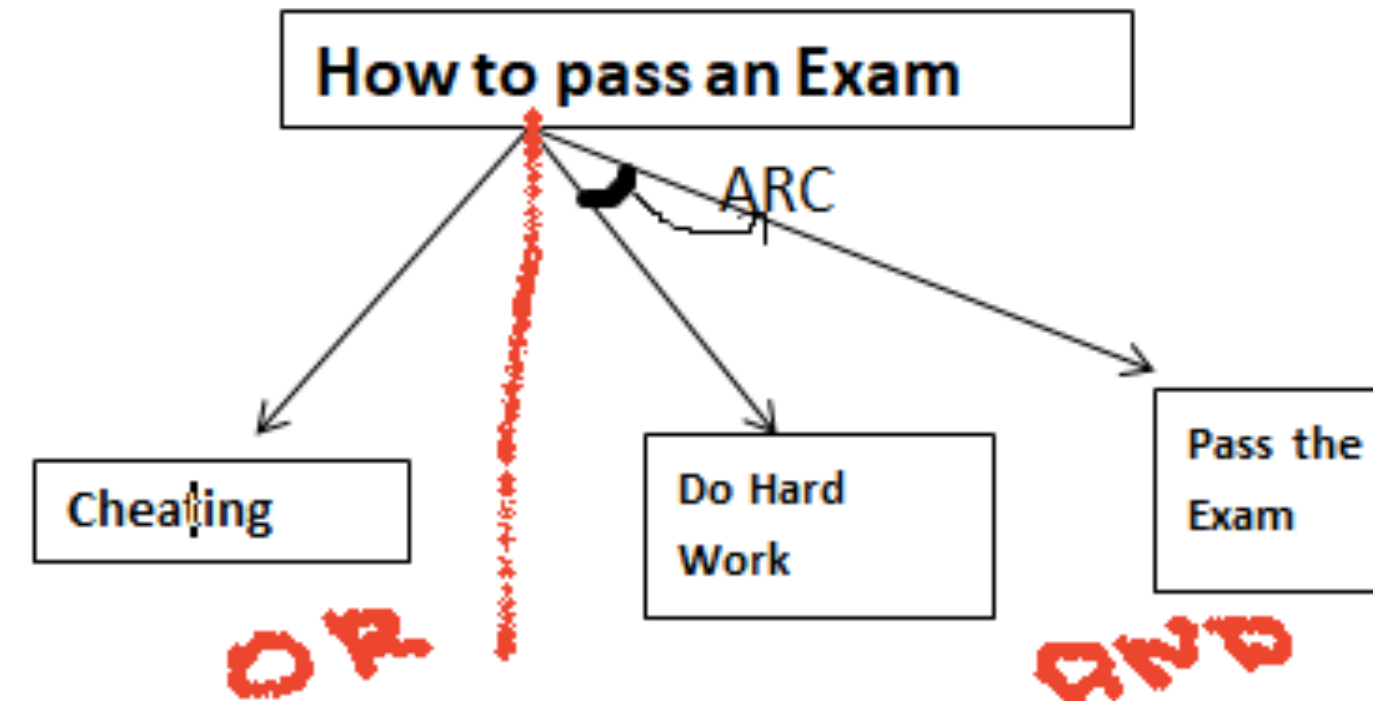
- It does not always produce the shortest path as it mostly based on heuristics and approximation.
  - A\* search algorithm has some complexity issues.
  - The main drawback of A\* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.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

# AO\*

- AO\* Algorithm basically based on problem decomposition (Breakdown problem into small pieces)
- When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, **AND-OR graphs** or **AND - OR trees** are used for representing the solution.
- The decomposition of the problem or problem reduction generates AND arcs.
- AND-OR Graph



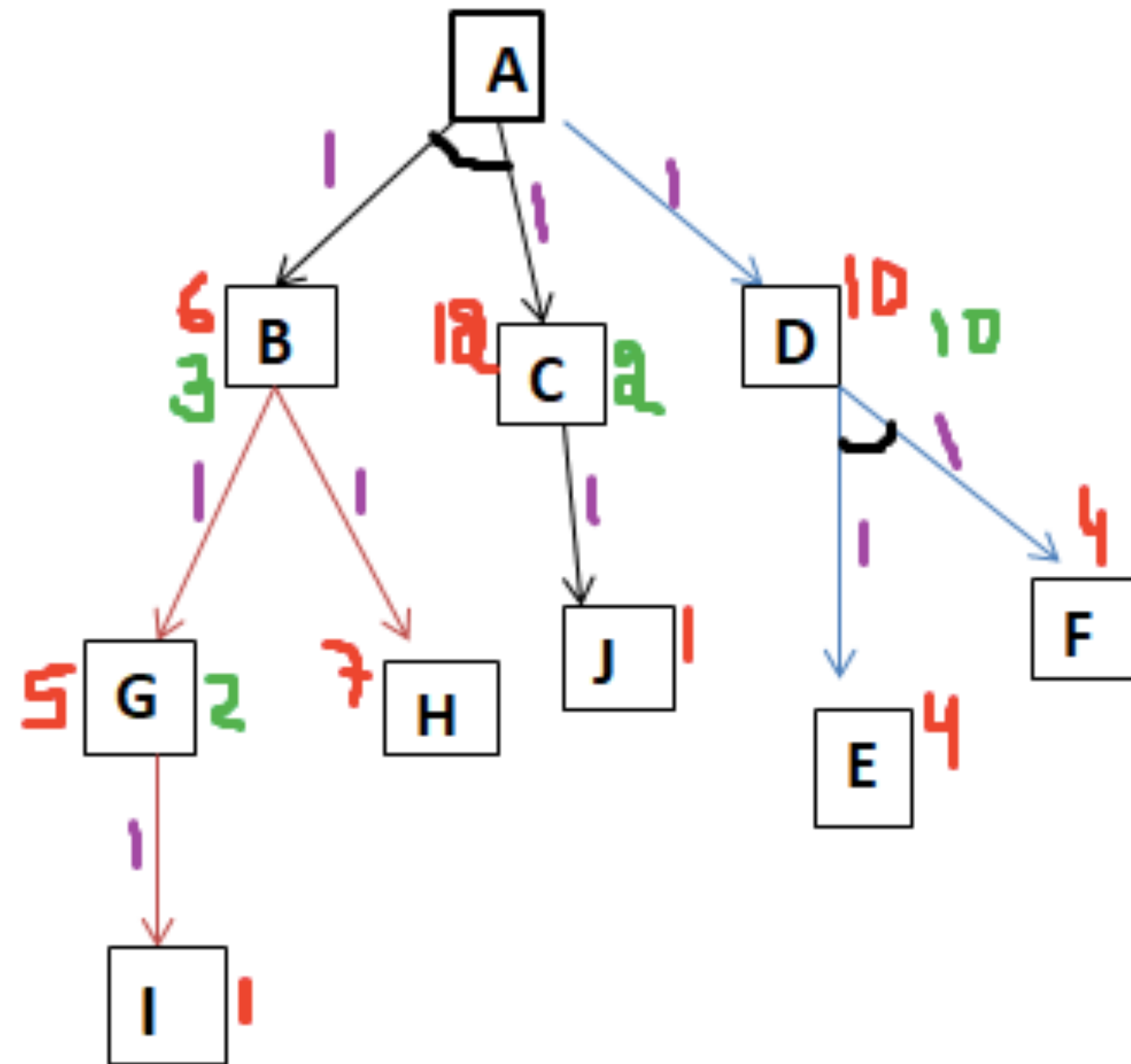




- The figure shows an AND-OR graph
  1. To pass any exam, we have two options, either cheating or hard work.
  2. In this graph we are given two choices, first do cheating **or (The red line)** work hard and **(The arc)** pass.
  3. When we have more than one choice and we have to pick one, we apply **OR condition** to choose one.(That's what we did here).
  4. Basically the **ARC** here denote **AND condition**.
  5. Here we have replicated the arc between the work hard and the pass because by doing the hard work possibility of passing an exam is more than cheating.

# Working:

- Let's try to understand it with the following diagram



- 
- The algorithm always moves towards a **lower cost value**.
  - Basically, We will calculate the **cost function** here ( **$F(n) = G(n) + H(n)$** )
  - **H: heuristic/ estimated** value of the nodes. and **G:** actual cost or edge value (here unit value).
  - Here we have taken the **edges value 1** , meaning we have to focus solely on the **heuristic value**.
1. The Purple colour values are edge values (here all are same that is one).
  2. The Red colour values are Heuristic values for nodes.
  3. The Green colour values are New Heuristic values for nodes.
-

---

# Procedure :

1. In the above diagram we have two ways from **A to D** or **A to B-C** (because of and condition). calculate cost to select a path
  2.  **$F(A-D) = 1+10 = 11$**  and  **$F(A-BC) = 1 + 1 + 6 + 12 = 20$**
  3. As we can see  **$F(A-D)$**  is less than  **$F(A-BC)$**  then the algorithm choose the path  **$F(A-D)$** .
  4. Form D we have one choice that is  **$F-E$** .
  5.  **$F(A-D-FE) = 1+1+ 4 +4 =10$**
  6. Basically **10** is the cost of reaching **FE from D**. And **Heuristic value of node D** also denote the cost of reaching **FE from D**. So, the new Heuristic value of D is 10.
  7. And the Cost from A-D remain same that is **11**.
-

---

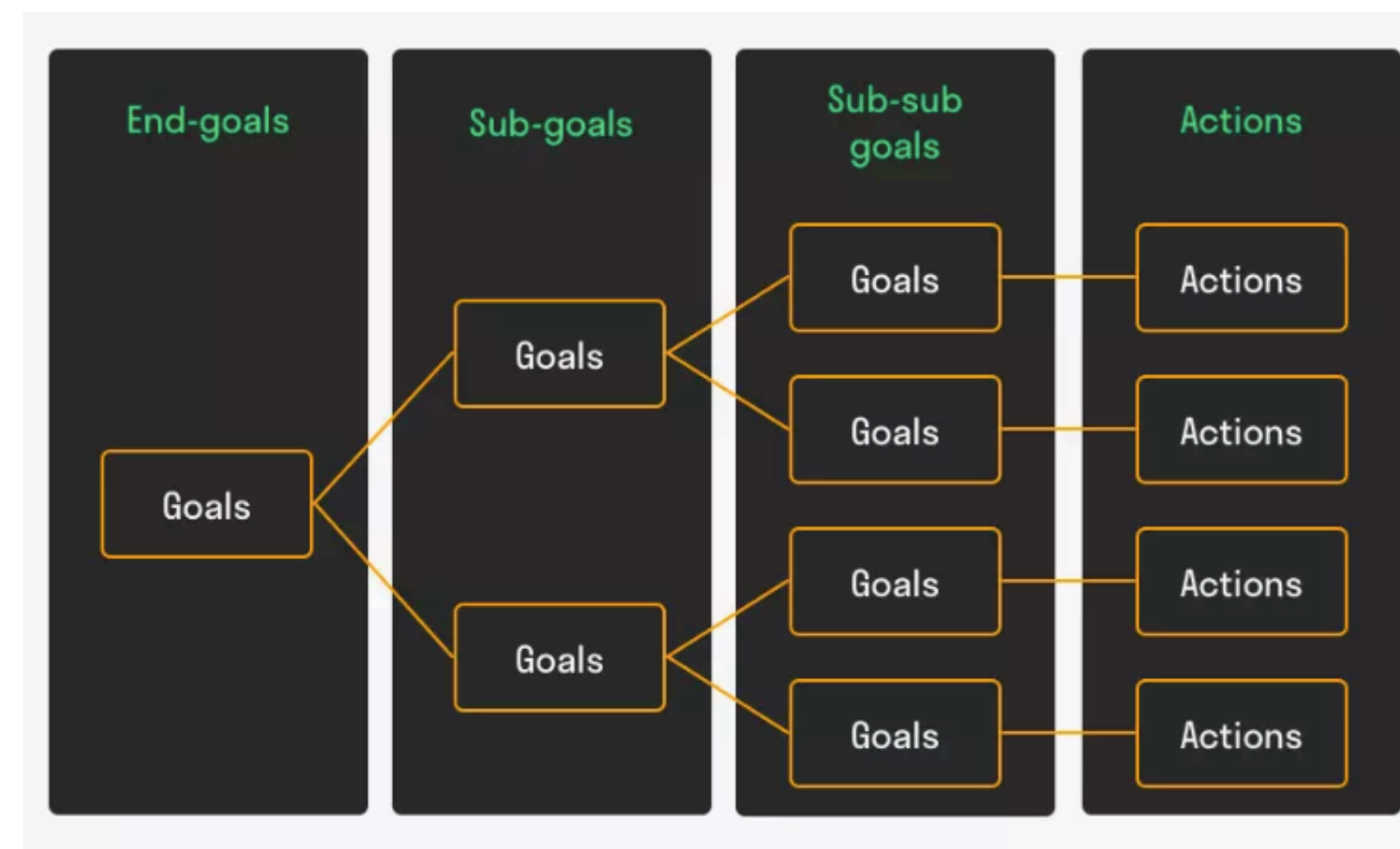
# A\* vs AO\*

- Both are part of informed search technique and use heuristic values to solve the problem.
  - The solution is guaranteed in both algorithm.
  - A\* **always** gives an **optimal solution** (shortest path with low cost) But It is not guaranteed to that **AO\*** always provide **an optimal solutions**.
  - **Reason:** Because AO\* does not explore all the solution path once it got solution.
-



# Mean End Analysis

- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.



---

- **How means-ends analysis Works:**

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

1. First, evaluate the difference between Initial State and final State.
2. Select the various operators which can be applied for each difference.
3. Apply the operator at each difference, which reduces the difference between the current state and goal state.

- **Operator Subgoalings:**

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. So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoalings**.

---

---

# Algorithm :

**Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.

**Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.

b. Attempt to apply operator O to CURRENT. Make a description of two states.

i) O-Start, a state in which O's preconditions are satisfied.

ii) O-Result, the state that would result if O were applied in O-start.

c. If

(First-Part  $\leftarrow$  MEA (CURRENT, O-START)

And

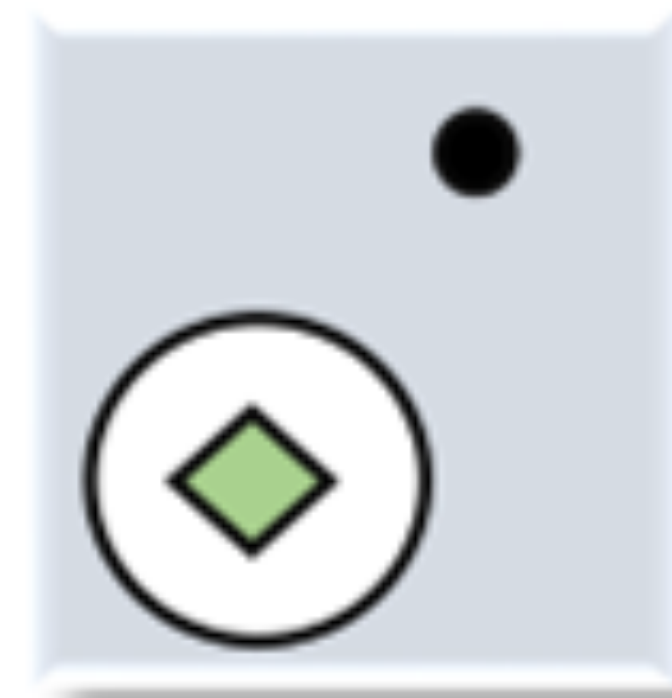
**(LAST-Part  $\leftarrow$  MEA (O-Result, GOAL)**, are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

---

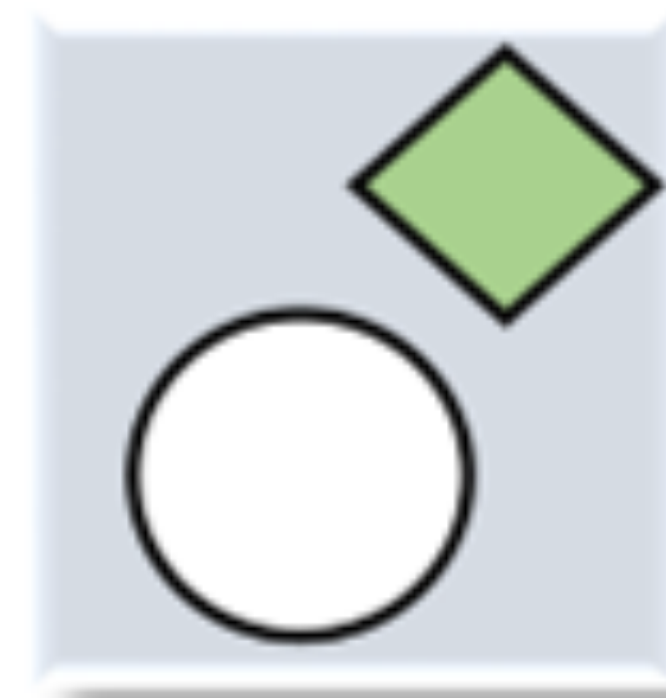
---

# Example :

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



**Initial State**

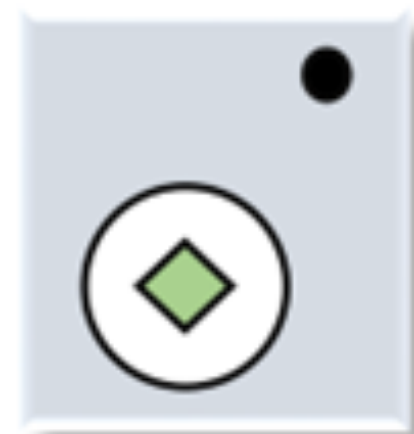


**Goal State**

---

# Solution :

- To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:
  1. Move
  2. Delete
  3. Expand
- A. **Evaluating the initial state:** In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.

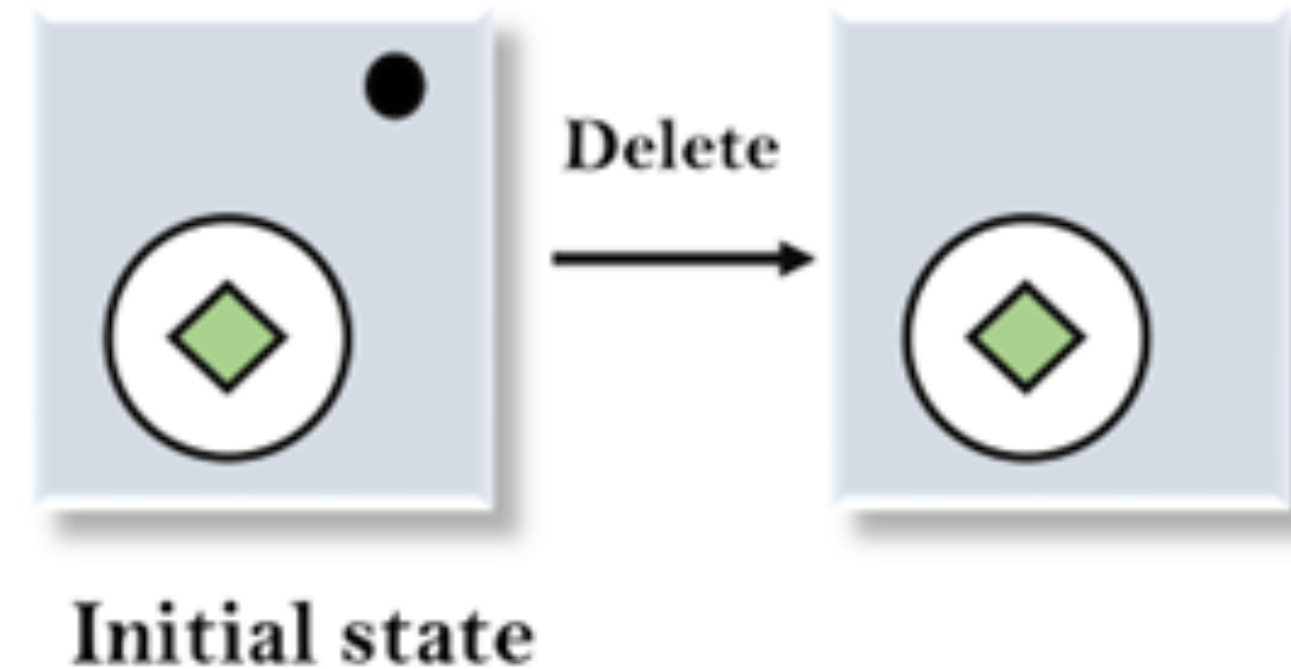


Initial state

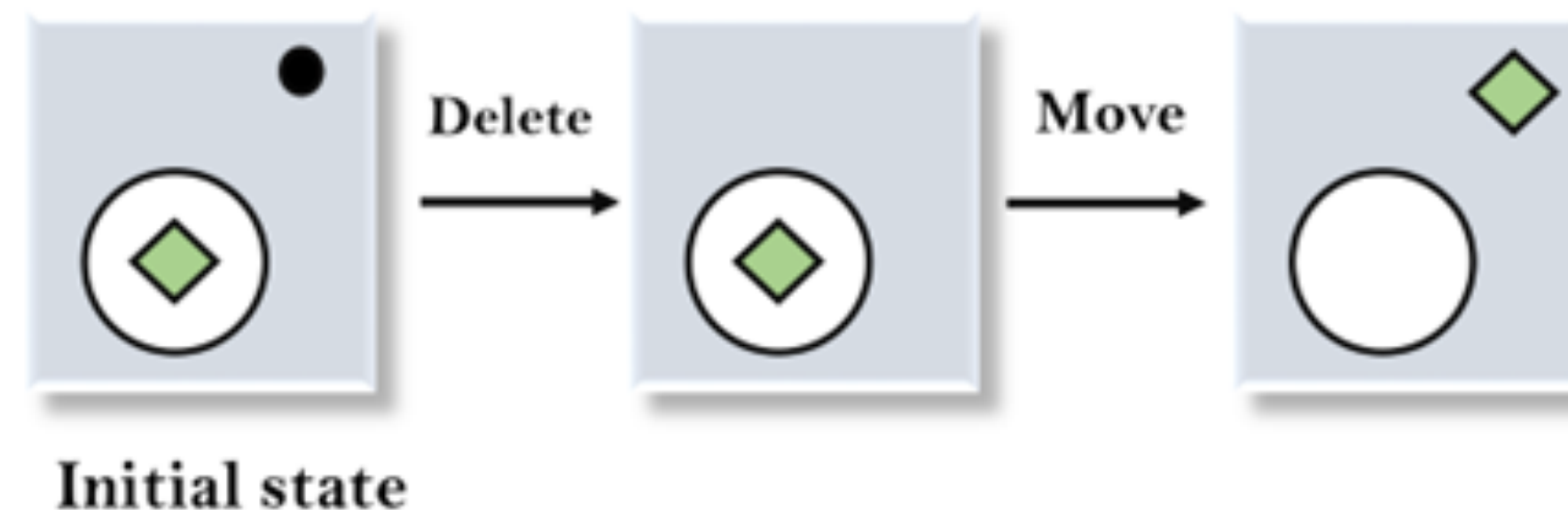
---



B. **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.



C. **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**.



D. **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.

