



# Problem Solving Methods

Artificial Intelligence

**CSE3007**

# Introduction



- One of the major application of AI is the problem solving. That is how to solve a particular problem.
- During 1970s, the main research in AI happened in the area of problem solving.
- We wanted to solve problems such as tick-tack-toe, 8-Queen problem, chess, GO games using machines or agents. This is called problem solving area.
- In this, state space search is one important concept.

# State-Space Search

- ❑ The number of states in which the problem can go is referred to as state-space search.
- ❑ **Example 1:**  
Final year project
- ❑ Example 2:  
Planning a destination in google maps.
- ❑ To represent the problem **precisely** in machines.
- ❑ Once we represent problem precisely, it is easy to **analyze** the game either for human or for machines.

# State-Space Search

- ❑ It is represented with  $S : \{S, A, \text{Result}(S, a), \text{Cost}(S, a)\}$ .
- ❑ In the above equation,  $S = \text{Start, Goal, Intermediate states}$
- ❑ For example, take 8-puzzle problem.

3X3, consists  
of 8 tiles, one  
empty space

2	3	4
5		1
8	7	6

Start state

1	2	3
8		4
7	6	5

Goal state

- ❑ In state-space search, we will explore various intermediate states of the start state in order to achieve goal state. We will compare each intermediate state with the goal state.

# State-Space Search

## □ A: Actions:

A → All possible actions: Up, down, left, right.

**Legal moves:** Only empty space should move

**Illegal move:** Other tiles should not move.

2	3	4
5		1
8	7	6

Start state

1	2	3
8		4
7	6	5

Cost: 1

Goal state

2	3	4
	5	1
8	7	6

2	3	4
5	1	
8	7	6

2		4
5	3	1
8	7	6

2	3	4
5	7	1
8		6

Resultant states

□ Result(s, a) → resultant states

□ Cost(s, a) → Cost for moving each tile or for reaching next state. In above example it is 1 unit(time, distance etc).

# State-Space Search

- The state-space search can be done in two ways:
  - **Uninformed or Blind search**, which is Exponentially computationally expensive  $O(b^d)$  . Here, b – branching factor, d- depth
  - **Informed search** uses local benefit or local information.

# Uninformed Vs Informed Search



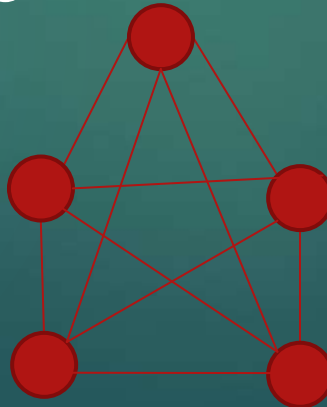
Uninformed or brute-force search	Informed search
Search without information	Search with information
No knowledge	Use Knowledge to find steps to solution.
Time consuming	Quick solution
More Complexity(Time, space)	Less complexity(Time, space)
DFS, BFS etc. are examples	A*, Heuristic Best First Search are examples.

# Uninformed vs Informed

- ❑ In brute-force, we only have start and goal state information and explore all the possibilities without any knowledge or domain information or guide.



- ❑ In Informed search, we use information or **heuristic**
- ❑ Example: Travelling sales man problem.

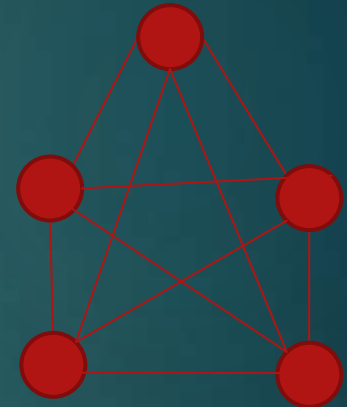




# Uninformed vs Informed

- ❑ Travelling sales man problem (TSP).
  - Start from one city and cover all other cities with minimum distance.

Brute force	Informed
<ul style="list-style-type: none"><li>▪ <math>(n-1)!</math> Possibilities to solve. Ex: <math>(5-1)! = 24</math> searching spaces to get optimal sol.</li><li>▪ Exponential Time complexity. If <math>n=100</math>, 1000 Tc increases.</li><li>▪ It is also called Non-polynomial (NP) problem</li><li>▪ No heuristic or guide or information.</li></ul>	<ul style="list-style-type: none"><li>• Heuristic search. Nearest neighbor distance used as heuristic.</li><li>• Polynomial time problem.</li><li>• In TSP, we can take local guide information to reach destination.</li><li>• Optimal solution may or may not reach.</li></ul>

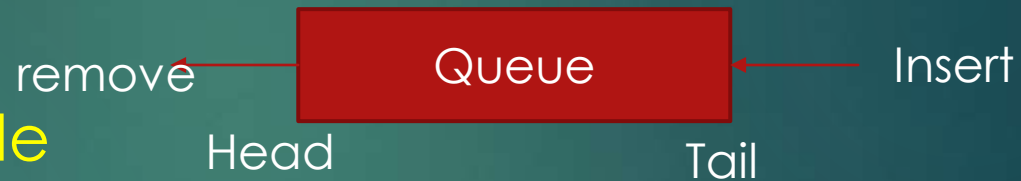




# Uninformed Search

# Breadth First Search (BFS)

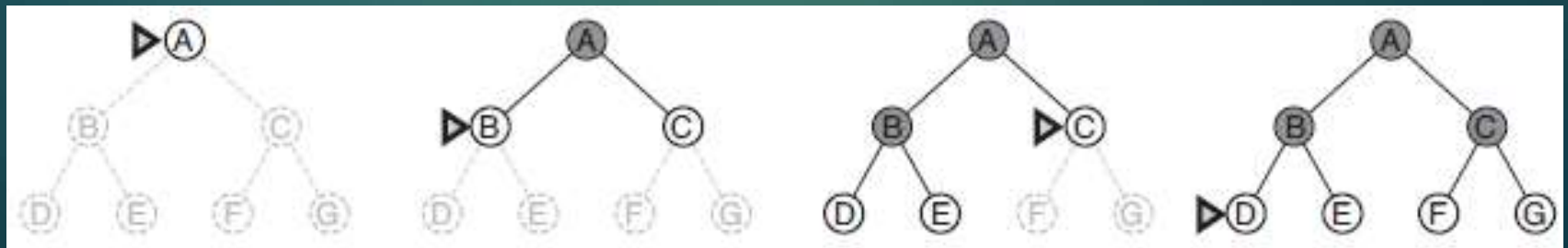
- ❑ It is a **uninformed or brute-force or blind search** in which we don't have any domain knowledge.
- ❑ It is a First In **First Out (FIFO)** method which uses Queue Data structure.



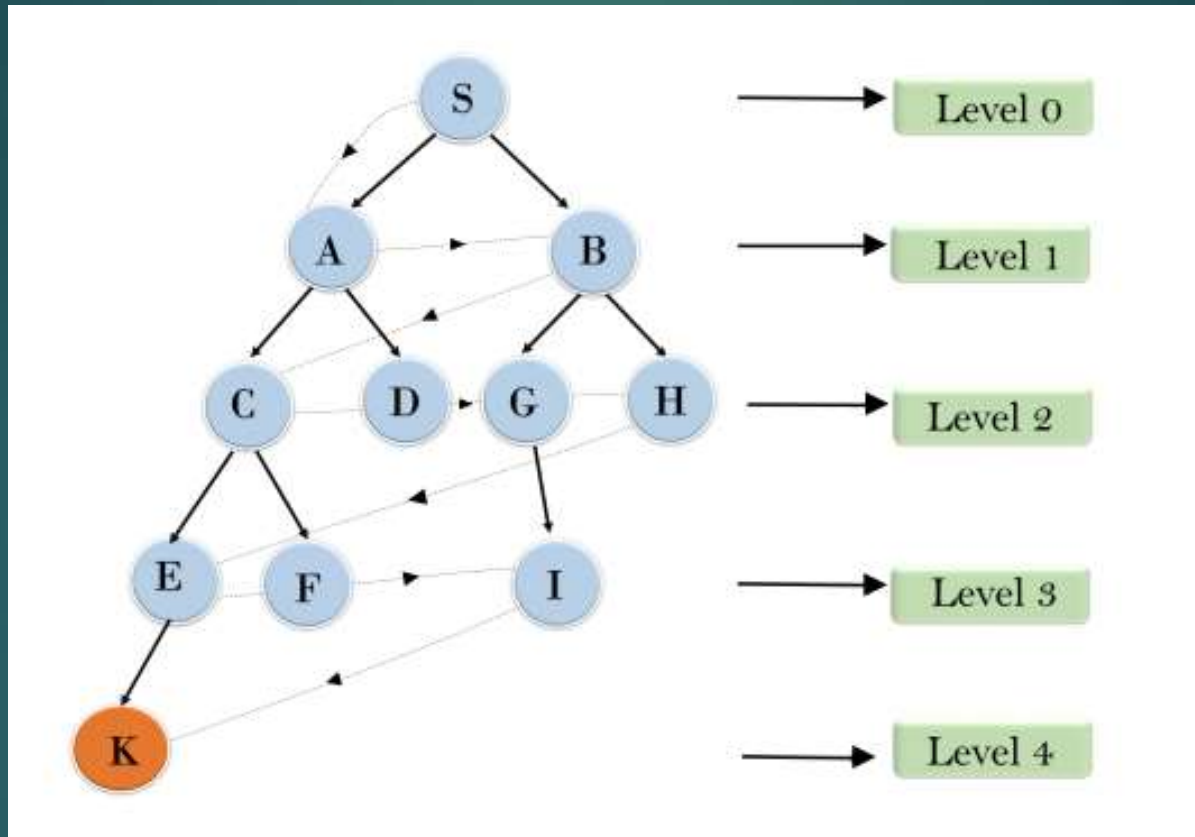
- ❑ **Shallowest Node**
- ❑ **Complete**
- ❑ **Optimal**
- ❑ **Complexity**

# BFS:

- ▶ The root node is expanded first, then all the nodes generated by the root node are expanded next, and then *their* successors, and so on.
- ▶ In general, all the nodes at depth  $d$  in the search tree are expanded before the nodes at depth  $d + 1$ .
- ▶ If there is a solution, breadth-first search is guaranteed to find it, and if there are several solutions, breadth-first search will always find the shallowest goal state first.



# BFS algorithm from the root node **S** to goal node **K**



S---> A--->B--->C--->D--->G--->H--->E--->F--->I--->K

# Breadth-first search Algorithm:

- Insert root node onto a Queue

- While (queue is not empty):

  - a) remove a node

  - b) if it is a goal node

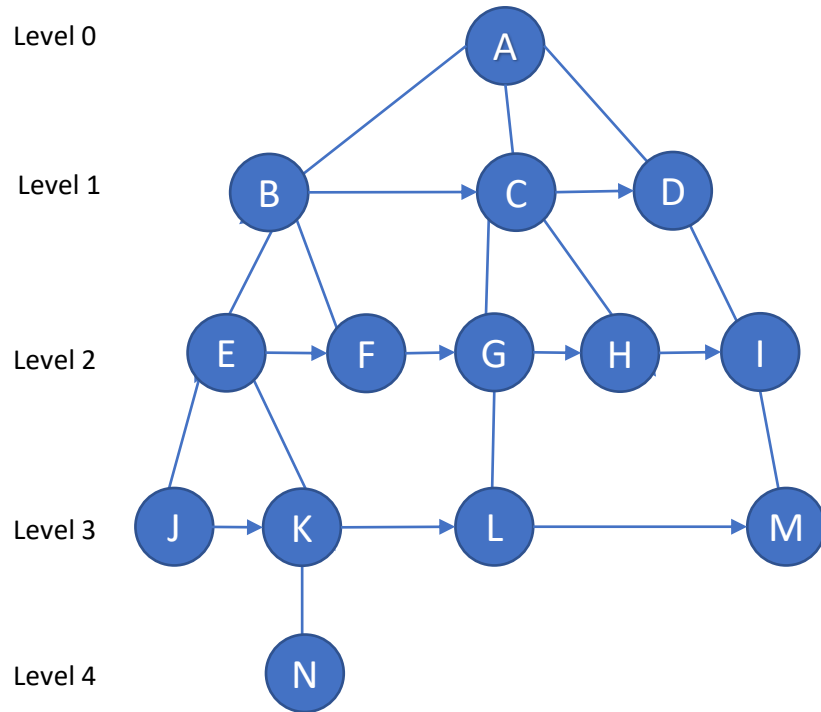
    - Stop Algorithm

  - c) if it is not a goal node

    - Insert all the children nodes into the Queue

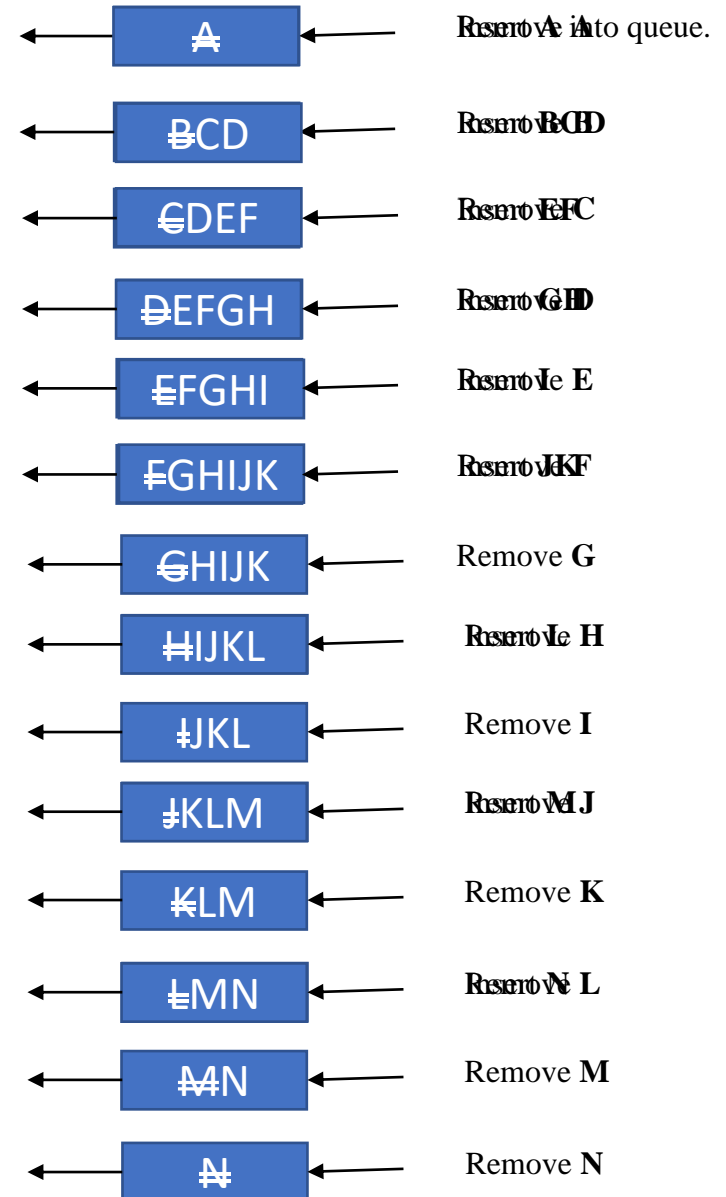
- Return Failure

# Breadth First Search (BFS)



Tree

- Shallowest Node
- Complete
- Optimal
- Time Complexity





❑ BFS is implemented using FIFO queue data structure.

❑ **Advantages:**

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

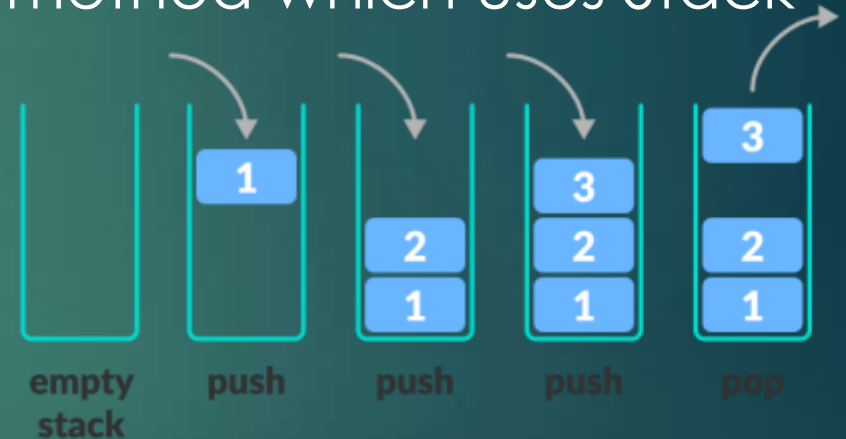
❑ **Disadvantages:**

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.



# Depth First Search (DFS)

- ❑ It is also an **uninformed or brute-force or blind search** in which we don't have any domain knowledge.
- ❑ It is a **Last In First Out (LIFO)** method which uses Stack Data structure.



- ❑ **Deepest Node**
- ❑ **In Complete**
- ❑ **Non Optimal**
- ❑ **Time Complexity**

Push = Insert

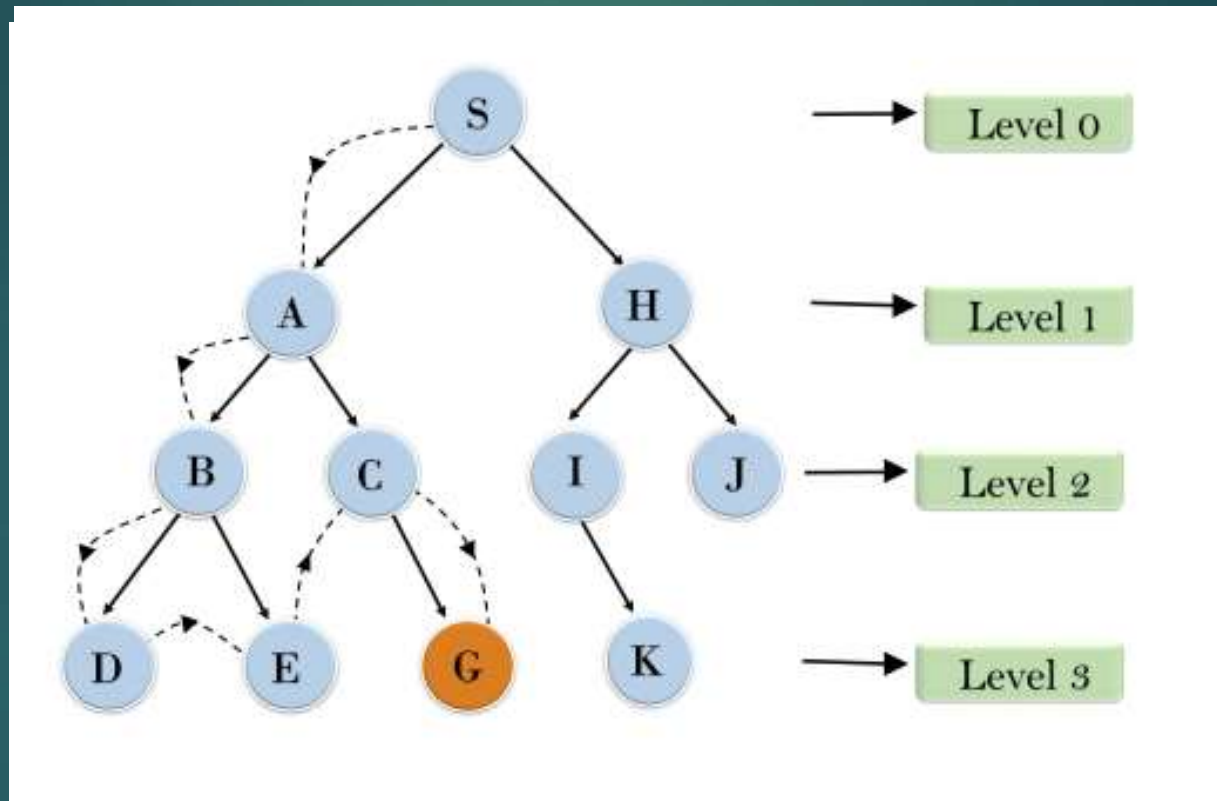
Pop = Remove

# Depth-first search

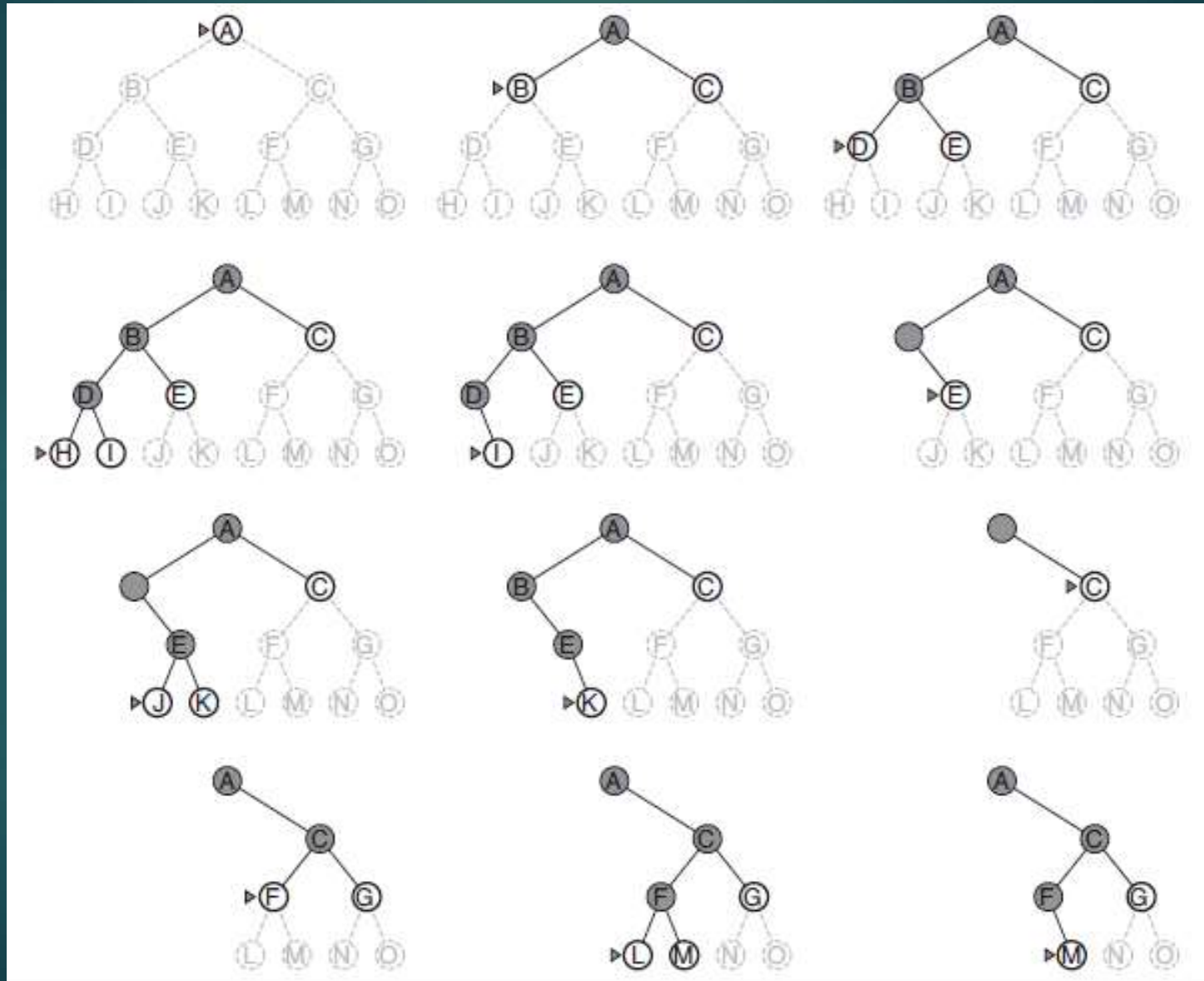
- ▶ **Depth-first search** always expands one of the nodes at the deepest level of the tree.
- ▶ Only when the search hits a dead end does the search go back and expand nodes at shallower levels.
- ▶ Stack function is used to represent DFS.
- ▶ Depth-first search has very modest memory requirements.

# Traversing technique

(Root node--->Left node ----> right node)



# Depth-first search on a binary tree



# Depth-first search Algorithm:

- Push root node onto a stack

- While (stack is not empty):

  - a) Pop a node

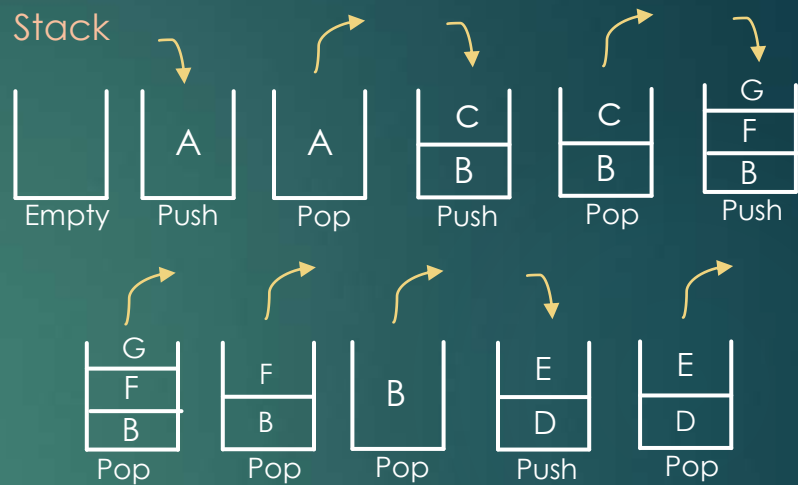
  - b) if it is a goal node

    - Stop Algorithm

  - c) if it is not a goal node


    - Push all the children nodes into the stack

- Return Failure



Shortest Path  $A \rightarrow C \rightarrow G$  (ACG)

- Two ways, But mechanism same
- Follow same direction (left to right)

- 
- Deepest Node - Depth wise search
  - Incomplete : DFS might stuck in infinite path or infinite loop. So, we might not reach goal state some times.
  - Non-optimal : DFS may or may not provide optimal solution.
  - There could be another solution with low cost.
  - Time complexity :

→  $O(b^d)$  for DFS Algorithms

b- branching factor

d- depth

E.g.: G is goal state

$b = 2$  (max),  $d = 2$ ;  $\therefore b^d = 2^2 = 4$



## ❑ **Advantage:**

- ▶ DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- ▶ It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

## ❑ **Disadvantage:**

- ▶ There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- ▶ DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.



# 8-Puzzle problem without Heuristic

- ❑ Uninformed search
- ❑ Breadth First Search (BFS)
- ❑  $O(b^d)$
- ❑ Four Moves( Empty space or tile can only move Left, Right, Top or Bottom)



- ❑ Problem: Solve the following 8-Puzzle using BFS without Heuristic. Mobile or board games, you might have played where you need to reach S to G state.

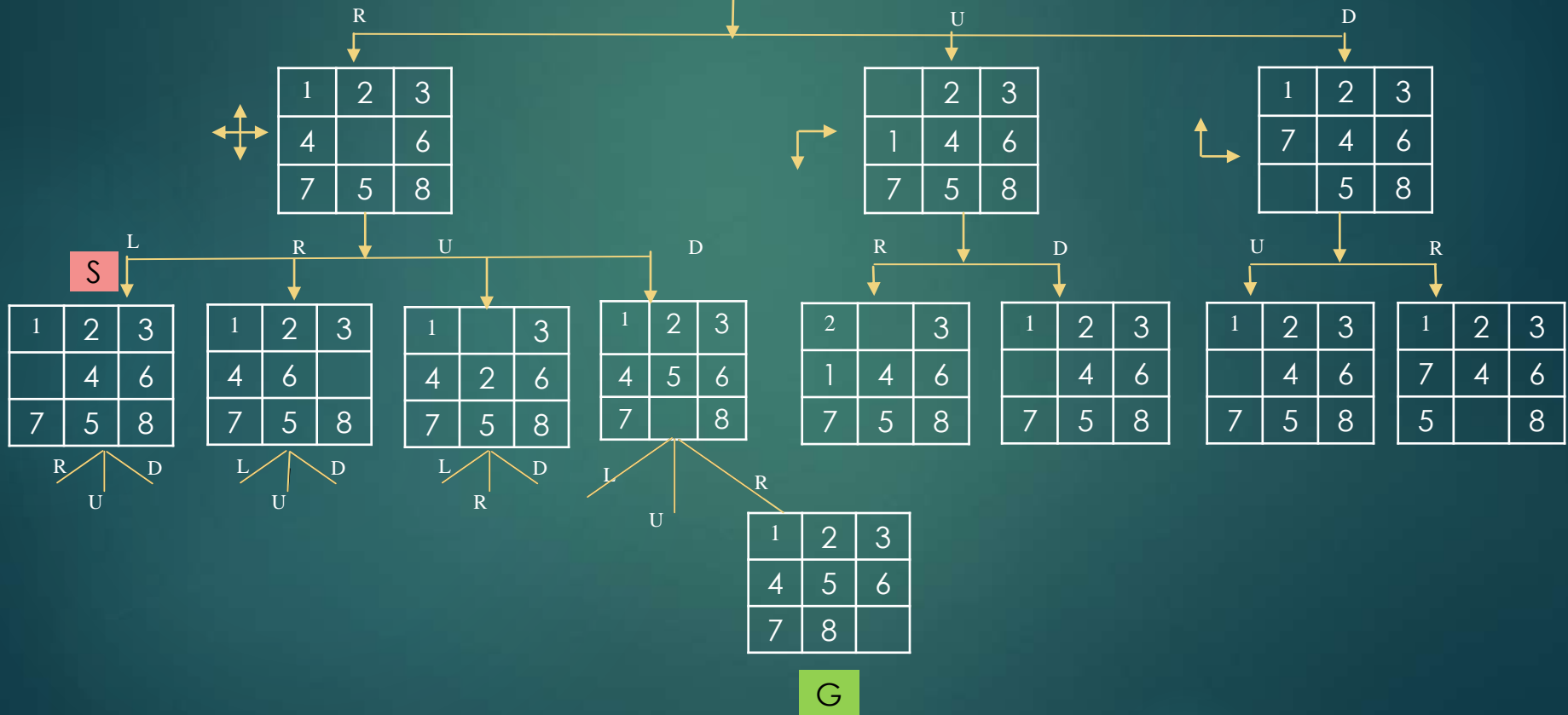
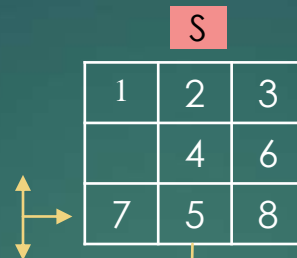
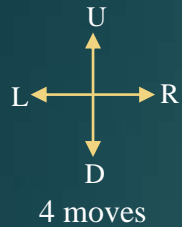
1	2	3
	4	6
7	5	8

S

1	2	3
4	5	6
7	8	

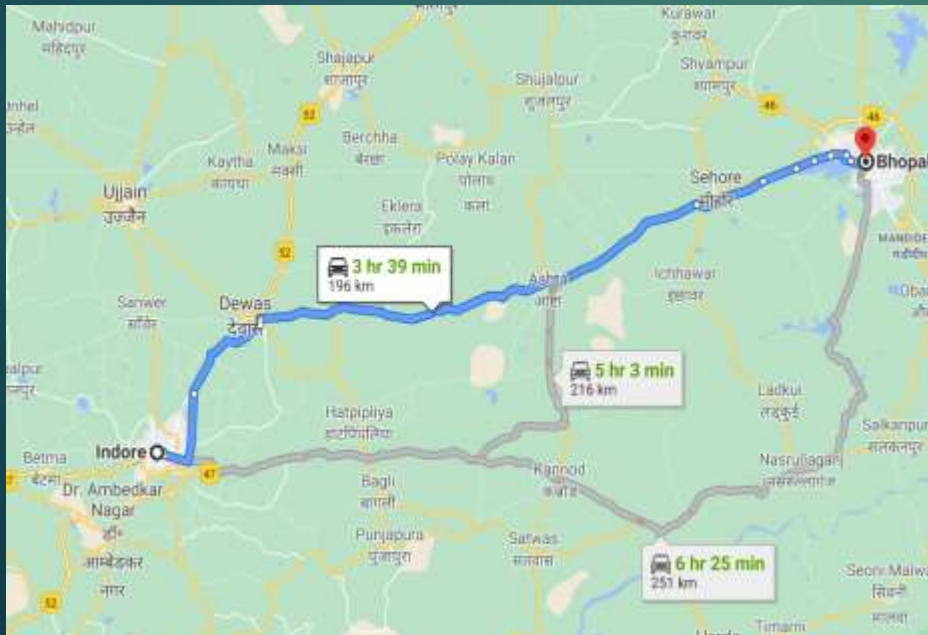
G

# 8 – Puzzle Problem Without Heuristic



# Heuristic in AI

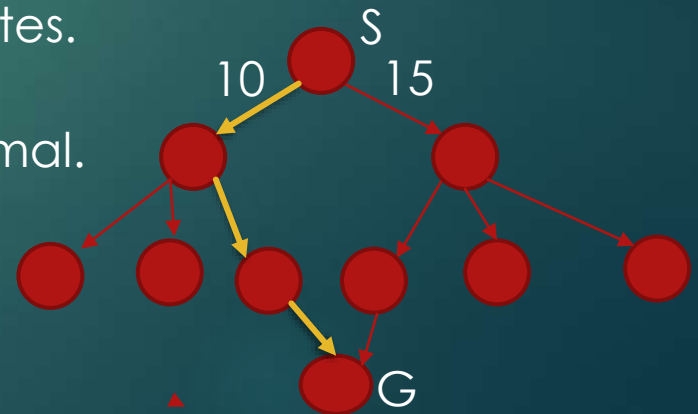
- ❑ Heuristic is a kind of support or help which is used to find the goal state with less computational complexity (Time and distance etc.)
- ❑ Heuristic in AI is also called as rule of Thumb Which gives hints to reach destination.
- ❑ It is a technique designed to solve problems quickly.
- ❑ Example: Directions on a road or distance board of the destination on road are examples of heuristics.



# Heuristic in AI

- ❑ In uninformed search, we are doing blind search where as informed search we use heuristic for finding the goal state.
- ❑ Uninformed search time complexity is exponential i.e.  $O(b^d)$
- ❑ Example:
  - 8-Puzzle: time complexity is  $3^{20}$
  - 15-Puzzle: time complexity is  $10^{15}$
  - 24-Puzzle: time complexity is  $10^{24}$
  - Chess: : time complexity is  $35^{80}$
  - These Uninformed search problems are also called as NP(Non-Polynomial) Problems. These problems will give Optimal solution at the cost of higher computational complexity.
- ❑ In informed search, using heuristic, we try to reach the goal state quickly without exploring all the possible states.

- Provide good solution but not the optimal.
- Polynomial Problem.



# How to calculate Heuristic?

- ❑ Euclidian distance: It is also called as straight line distance between two nodes.

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



- ❑ Man-hattan distance:

$$Mdist = |x_2 - x_1| + |y_2 - y_1|$$

1	3	2
6	5	4
	8	7

S

1	2	3
4	5	6
7	8	

G

- Example: 8-Puzzle problem

$$\text{Cost} = 0 + 1 + 1 + 2 + 0 + 2 + 7 + 0$$

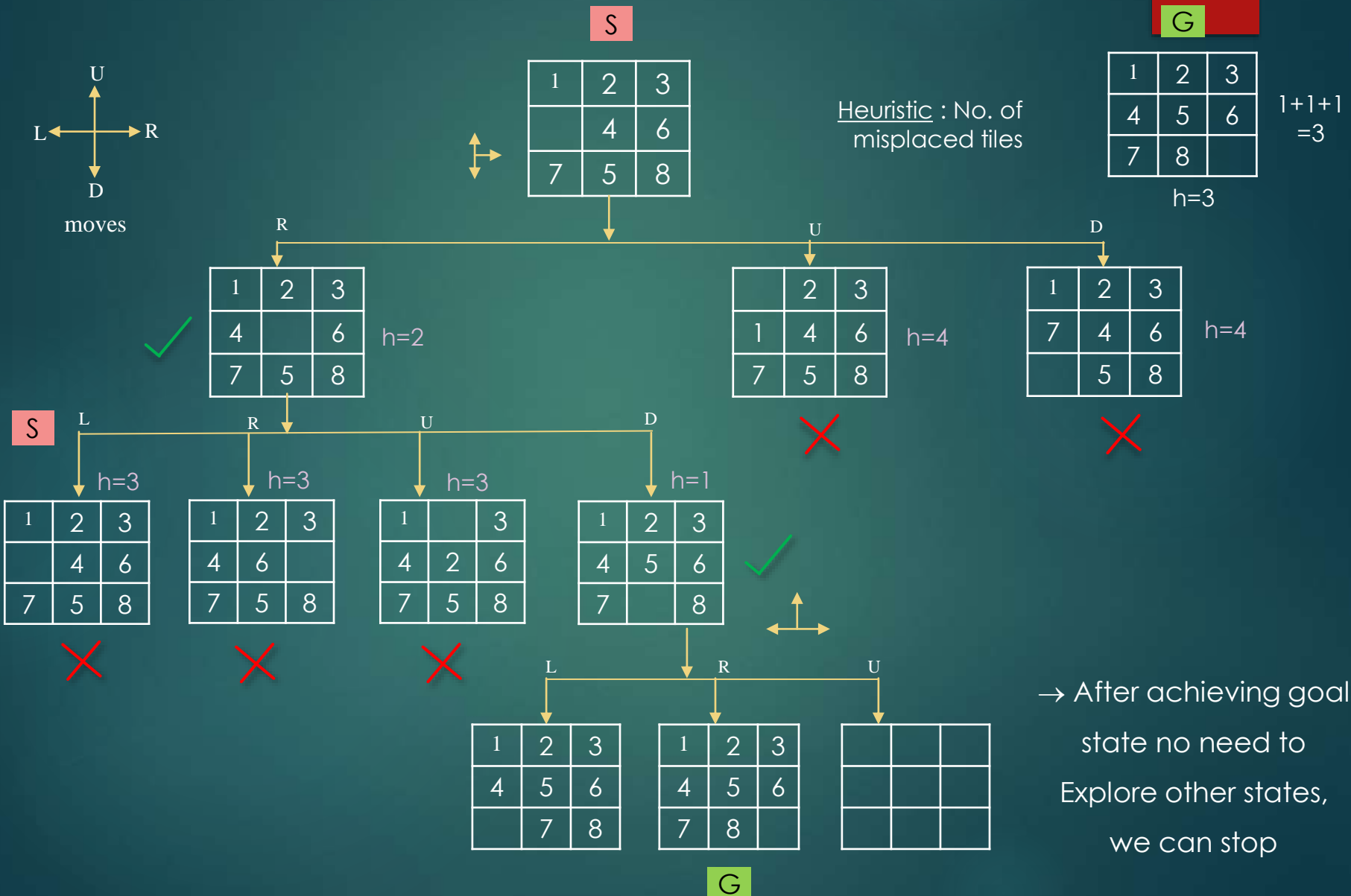
- ❑ Number of Misplaced tiles:

- Cost is the Number of misplaced tiles of S compared to G. Cost = 5.



# Informed Search

# 8 – Puzzle Problem With Heuristic

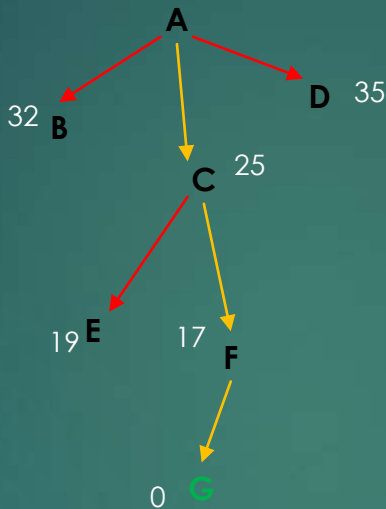
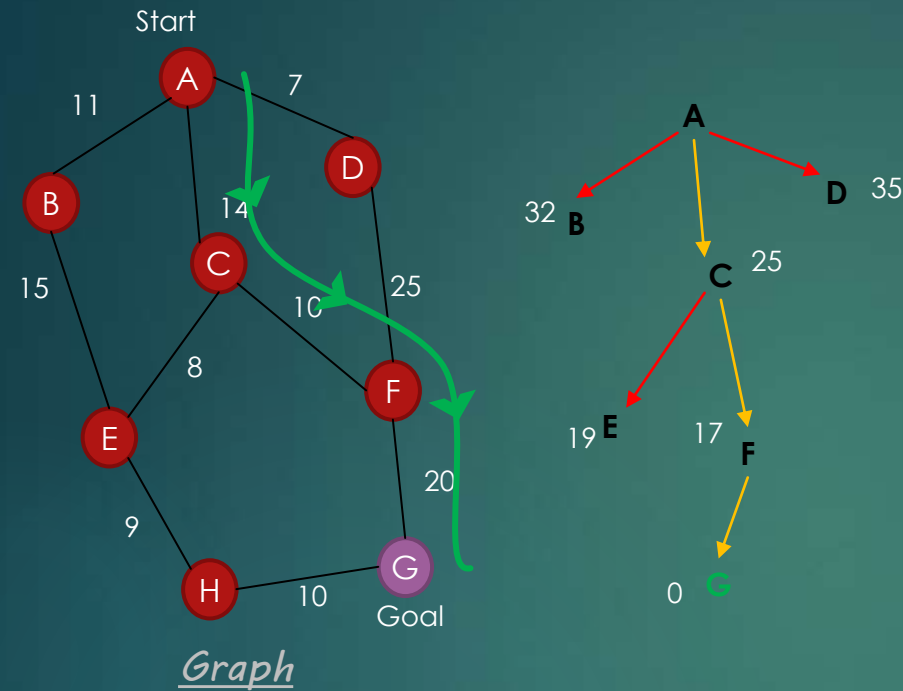


# Best First Search (BFS)

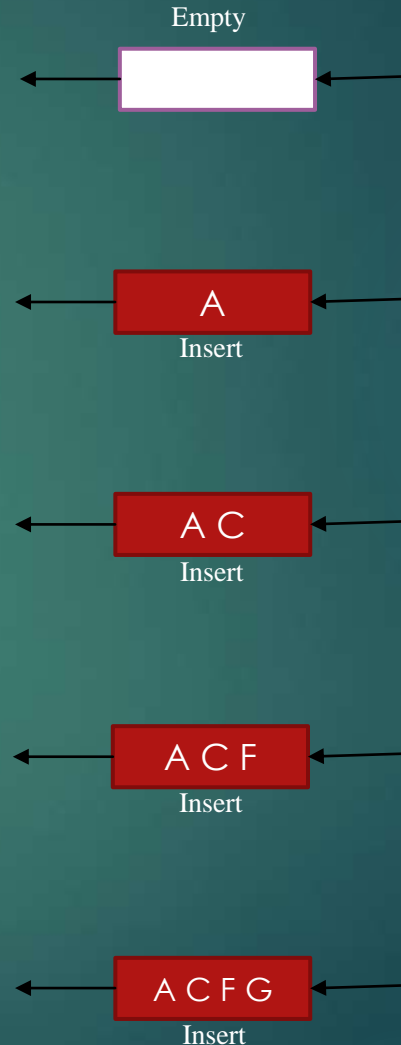
- ❑ It is a **informed search** in which we will be having **heuristic** to find the goal state quickly.
- ❑ Algorithm:
  - Let **OPEN** be a priority Queue containing Initial state.
  - Loop
    - If OPEN is empty return failure
    - Otherwise Node  $\leftarrow$  Remove – First(OPEN)
      - If node is a Goal  
Then return the path from initial to Node
      - else generate all successors of Node and put the generated Nodes into **OPEN** according to their **f(heuristic) values**.
  - End Loop
  - It's a complete algorithm



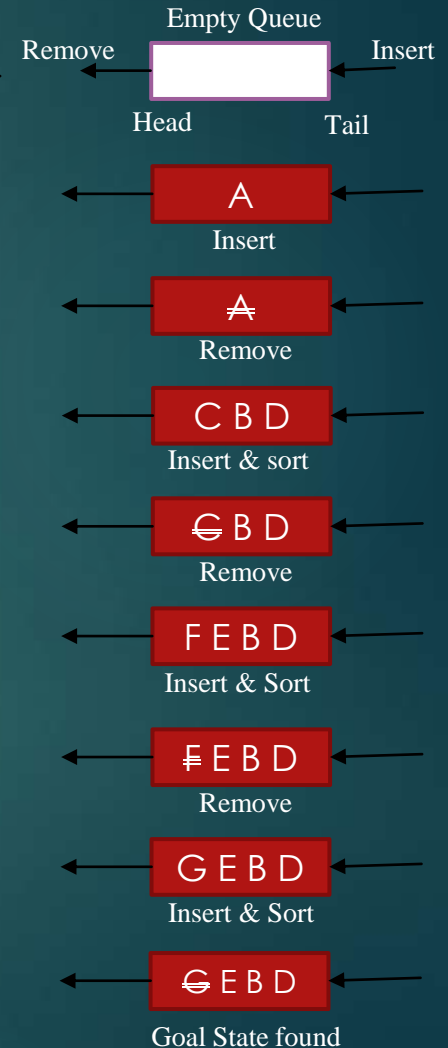
# Best First Search (BFS)



Closed Queue



Open (Priority) queue



## Heuristic:

Euclidian or straight line distance

- A → G = 40      E → G = 19
- B → G = 32      F → G = 17
- C → G = 25      H → G = 10
- D → G = 35      G → G = 0

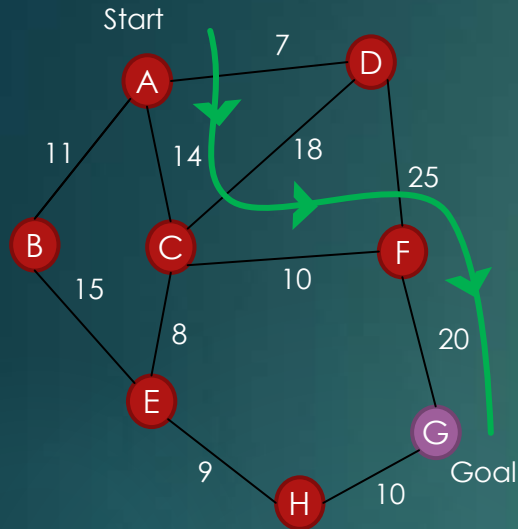
Best First search (BFS) works on ‘Heuristic’ but not on ‘cost’

Path : **A → C → F → G**

# Beam Search Algorithm

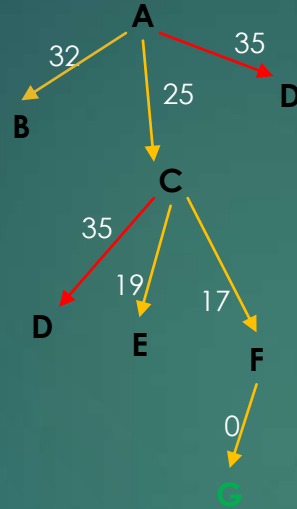
- ❑ It is an **informed search** technique.
- ❑ Take care of **space complexity (Constant)**
- ❑ Beam Width (Beta) is given in advance for every problem.
- ❑ It is the **extension of Best first search** method.
- ❑ Searching will be done based on **Minimum heuristic value**.
- ❑ Its an **incomplete algorithm** since we are pruning branches based on beam width Beta value.
- ❑ If  $\beta = 1$ , then it is called hill climbing algorithm.

# Beam Search technique

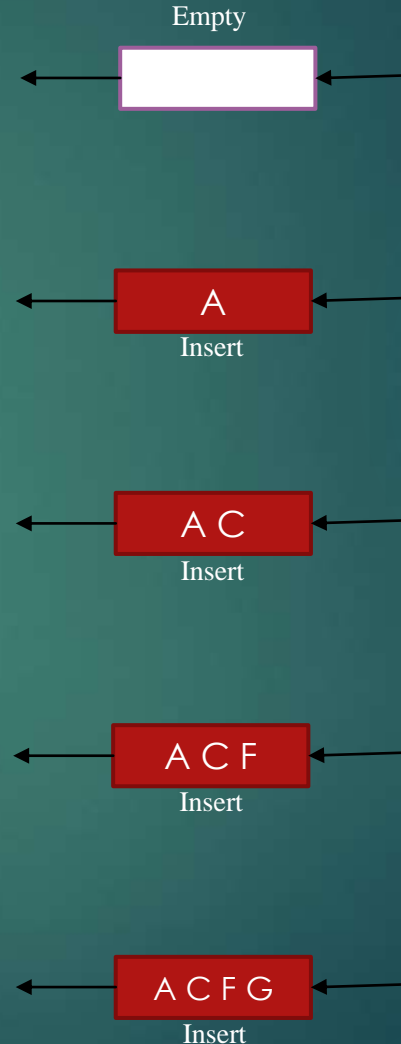


*Graph*

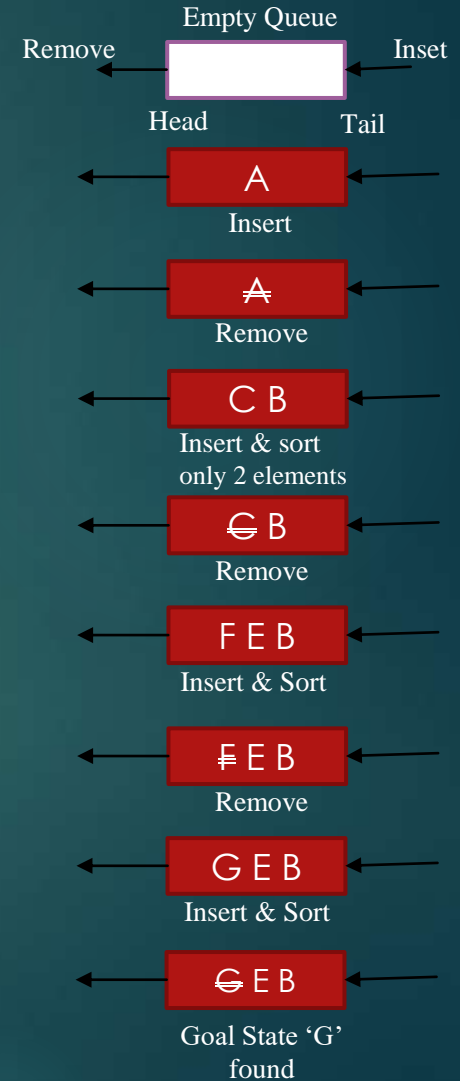
$\beta = 2$



Closed Queue



Open (Priority) queue



Straight line distance

$A \rightarrow G = 40$	$E \rightarrow G = 19$
$B \rightarrow G = 32$	$F \rightarrow G = 17$
$C \rightarrow G = 25$	$H \rightarrow G = 10$
$D \rightarrow G = 35$	$G \rightarrow G = 0$

Path : **A** → **C** → **F** → **G**

# Hill climbing Algorithm

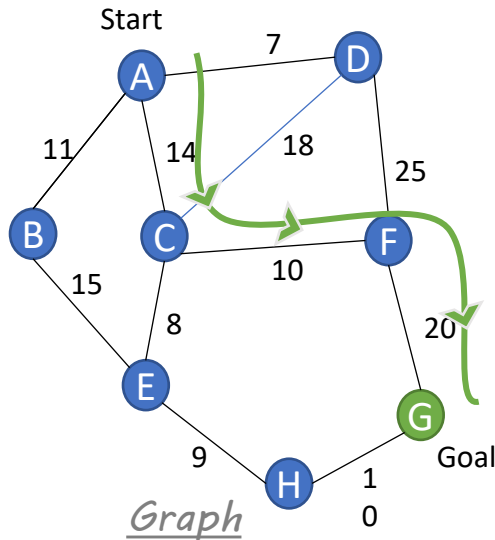
- ❑ It is a local search algorithm which does not have global domain knowledge.
- ❑ It is a greedy approach which moves in the direction of best heuristic value otherwise it will stop.
- ❑ **Back tracking:** Like uninformed and informed search techniques that discussed so far it can not back track. If algorithm does not find best solution in a particular branch.
- ❑ It is an informed search technique.
- ❑ Take care of space complexity (Constant), i.e. Beam Width (Beta) is 1.
- ❑ It is the extension of Beam search method.
- ❑ Searching will be done based on Minimum heuristic value.
- ❑ It is an incomplete algorithm since we are pruning all branches except keeping one with minimum heuristic value.

# Hill climbing Algorithm

□ Algorithm steps:

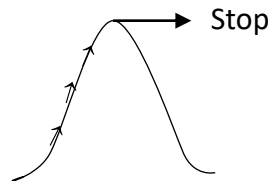
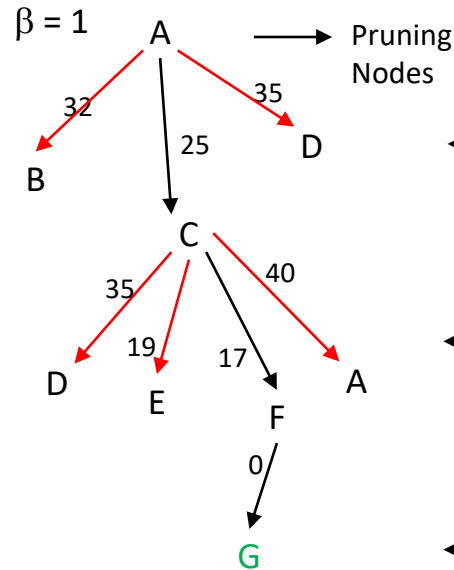
1. Evaluate initial state.
2. Loop Until a solution is found or there are no operators (Nodes) left
  - select and apply a new operator (node)
  - evaluate the new state:
  - if goal then quit
  - If better than current state then it is new current state.

# Hill climbing technique



Straight line distance:

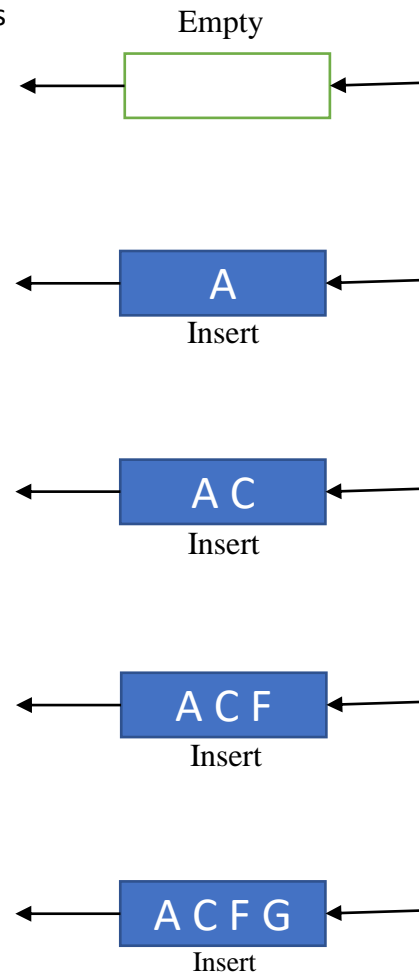
$A \rightarrow G = 40$	$E \rightarrow G = 19$
$B \rightarrow G = 32$	$F \rightarrow G = 17$
$C \rightarrow G = 25$	$H \rightarrow G = 10$
$D \rightarrow G = 35$	$G \rightarrow G = 0$



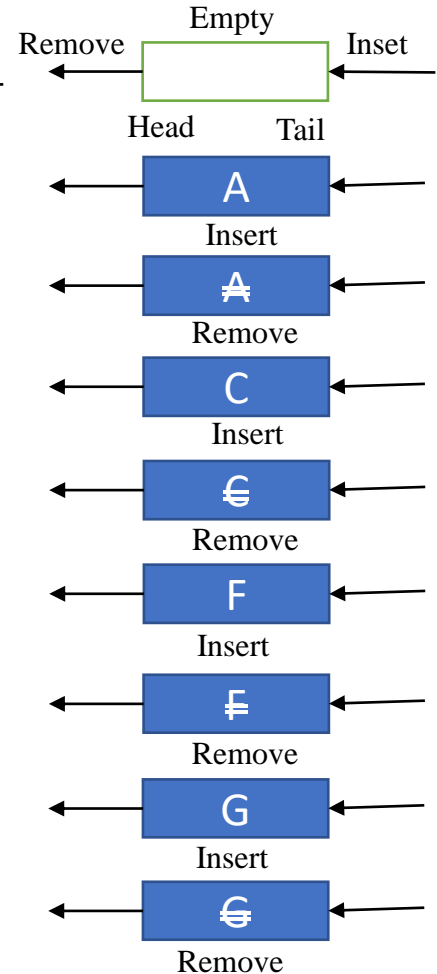
Goal state 'G' found

Path :  $A \rightarrow C \rightarrow F \rightarrow G$

Closed Queue

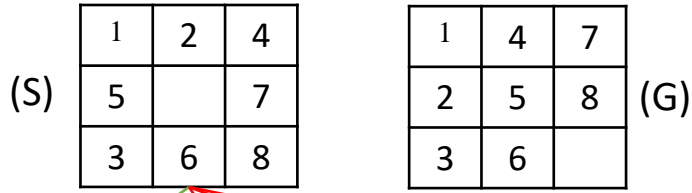


Open (Priority) queue



# Hill Climbing Example

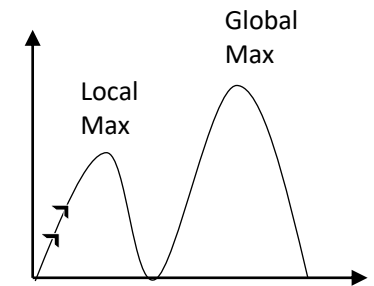
Heuristic: Misplaced tiles



Problems in Hill Climbing:

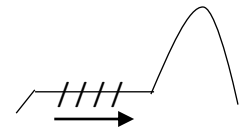
- Local Maximum*

In Local Search, we don't have complete information of the domain, Because of that, we may or may not reach global optimum



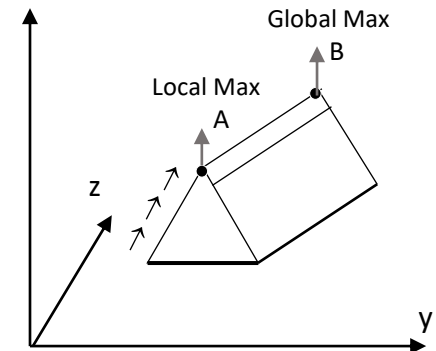
- Plateau/ Flat Maximum*

If all the heuristic values in a state are same it is flat Max.

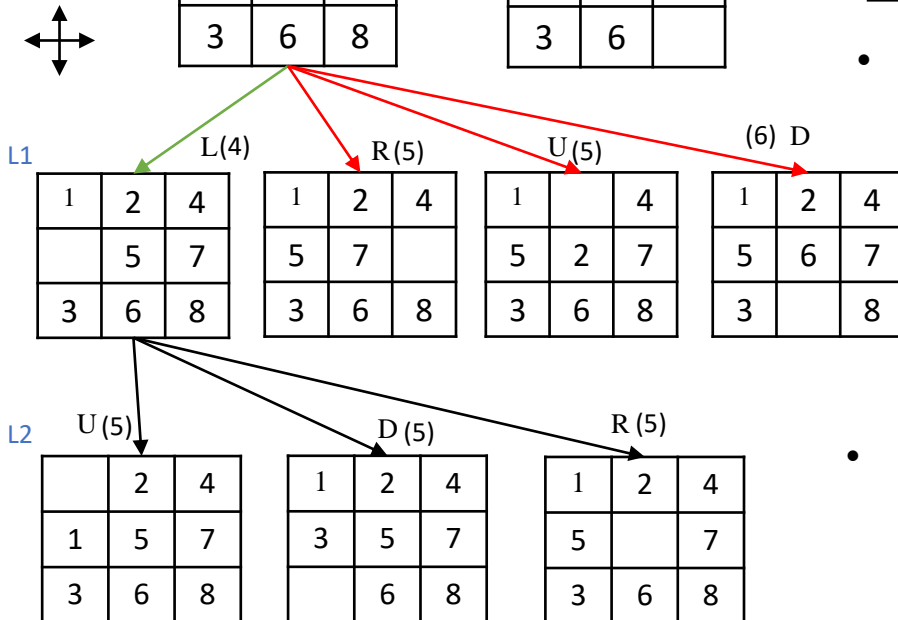


- Ridge*

Special case of local maximum.



Eg.: Mobile



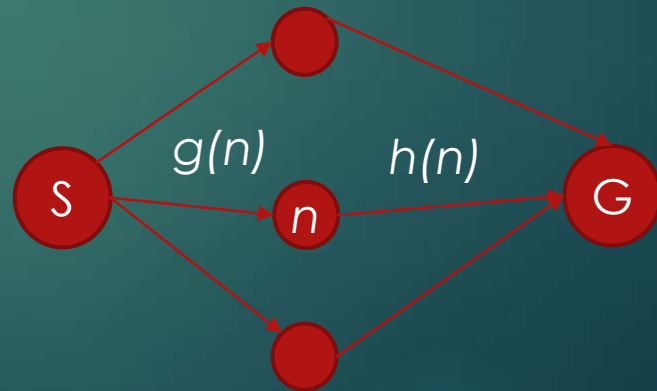
# A\* Algorithm

- ❑ Informed search technique.
- ❑ A\* is called as admissible which means this algorithm gives optimal solution for sure.
- ❑ A\* algorithm works on the following equation:

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

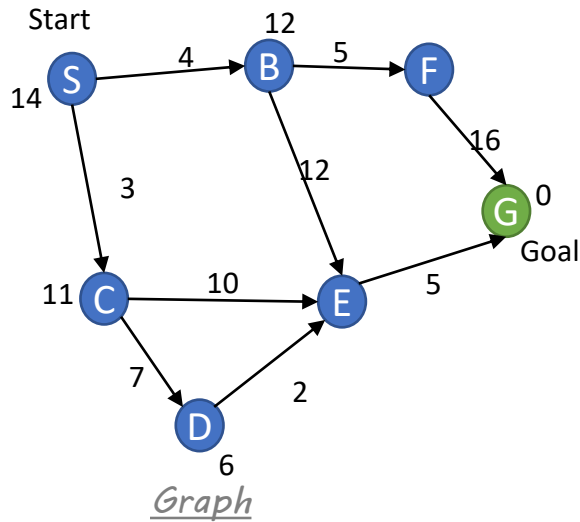
where,

- $g(n)$  is the actual cost from start node  $S$  to node  $n$ .
- $h(n)$  is the estimated cost or heuristic cost from node  $n$  to Goal node  $G$ .





# A\* Algorithm:



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

$g(n)$  = Actual cost from  $s \rightarrow n$

$h(n)$  = estimated cost from  $n \rightarrow G$

$s \rightarrow s$ :

$$f(s) = g(s) + h(s) = 0 + 14 = 14$$

$s \rightarrow B$ :

$$f(B) = g(B) + h(B) = 4 + 12 = 16$$

## Time Complexity

$$T_c = O(b^d) \text{ (max Value)}$$

$$SC = O(b^d)$$

$b$  = branch factor

$d$  = depth

$s \rightarrow C$ :

$$f(c) = g(c) + h(c) = 3 + 11 = 14$$

$SC \rightarrow E$ :

$$f(E) = 3 + 10 + 4 = 17$$

$SC \rightarrow D$ :

$$f(D) = 3 + 7 + 6 = 16$$

$SCD \rightarrow E$ :

$$f(E) = 3 + 7 + 2 + 4 = 16$$

$SCDE \rightarrow G$ :

$$f(G) = 12 + 5 + 0 = 17$$

## Straight line distance:

$$S \rightarrow G = 14 \quad E \rightarrow G = 4$$

$$B \rightarrow G = 12 \quad F \rightarrow G = 11$$

$$C \rightarrow G = 11 \quad G \rightarrow G = 0$$

$$D \rightarrow G = 6$$

$h(n)$