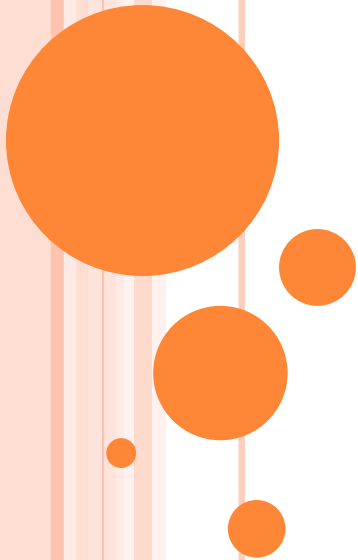


# **SEARCHING ALGORITHMS-I**

**(INTRODUCTION, HEURISTIC FUNCTIONS,  
BEST FIRST, HILL CLIMBING, BEAM SEARCH)**



## **Informed Search:**

- Searching with the information
- Use knowledge to find steps to the solution
- Quick solution
- Less complexity

**Heuristic Function:** User defined functions that is an estimate of the cost/distance to the goal state from the current state.

**Examples are A\*, Heuristic BFS, Heuristic DFS, Best first search.**



# HEURISTIC FUNCTIONS

- The word heuristic has been derived from Greek word eureka or eurisko, which means “*I found*” or “*I have found it*”.
- Heuristic functions in searching are human defined functions that is an estimate of the **cost or distance** to the goal from the node.
- The idea of using heuristic function is **to guide the search**, so that it has a tendency to explore the region leading to the goal.
- To incorporate heuristic values in search, the node pair values in the search, the node pair representation will be augmented with the heuristic value such as  
searchNode=(currentState,parentState, heuristicValue)

# HEURISTIC FUNCTION EXAMPLE 1- ROUTE FINDING PROBLEMS

- In a route finding problem in a city, the heuristic function could be measure of a distance of the node from the goal node.
- If the location of each node is in terms of its co-ordinates, then one heuristic function could be **Euclidean distance** of the node from goal node.

$$H(n)=\text{Euclidean Distance}(n, \text{goal})= \sqrt{(x_{\text{goal}} - x_n)^2 + (y_{\text{goal}} - y_n)^2}$$

- This function gives an estimate of the distance and the actual distance may be more than the straight distance.
- Another distance metric that could be used is the **Manhattan distance** or the city block distance (assumes edges from a grid as the streets in the Manhattan)

$$H(n)=\text{Manhattan Distance}(n, \text{goal})= |x_{\text{goal}} - x_n| + |y_{\text{goal}} - y_n|$$

- Similarly, we can use other distance measure such as Minkowski distance.

# HEURISTIC FUNCTION EXAMPLE 2- EIGHT PUZZLE PROBLEM

- In a 8-puzzle problem, following heuristic functions can be defined:
  - $H_1(n)$ : number of misplaced tiles in  $n$  as compared to the goal node.
  - $H_2(n)$ : number of correctly placed tiles in  $n$  as compared to the goal node.
  - $H_3(n)$ : the sum of Manhattan distance of each tile from the destination.
- For instance, for the node (n) and goal node given below,

Initial:

2		3
1	8	4
7	6	5

Goal:

1	2	3
8		4
7	6	5

- The values of three heuristic function defined above are:

$H_1(n) = 3$  (1,2, 8 are not at correct positions)

$H_2(n) = 5$  (3,4,5,6,7 are at correct position)

$H_3(n) = 1$  (for tile 1) + 1 (for tile 2) + 0 + 0 + 0 + 0 + 0 + 1 (for tile 8) = 3

# HOW TO DECIDE A HEURISTIC FUNCTION FOR A PROBLEM?

- In the previous examples, it is shown that there can be more than one heuristic for a problem.
- The heuristic function for a problem is generally decided on the basis of average branching factor which is computed as follows:

$$\text{AverageBranchingFactor} = \frac{\text{Number of nodes expanded}}{\text{Path of Optimal Solution}}$$

- Ideal value of this factor is 1 and maximum value for a particular problem is known (such as 4 for 8-puzzle problem).
- Thus, number of runs with different initial and goal states are executed with all the heuristic functions and one which gives best average branching factor is considered.

# HEURISTIC FUNCTIONS CONTD.....

- Heuristic functions improve the searching efficiency possibly by **sacrificing** the **completeness** and **optimal property** of algorithms.
- They can guide the search direction but overlook excellent paths or points of interest.
- On average they find **good solutions** (not possibly **optimal**) solutions for hard problems (like travelling salesman problem) in exponential time.



# INFORMED SEARCHING ALGORITHMS

- The various types of informed searching algorithms are:
  - Best first Search
  - Hill Climbing
    - Simple Hill Climbing
    - Steepest Hill Climbing
  - Beam Search
  - A\* Searching
  - Iterative Deepening A\* Searching





# BEST FIRST SEARCH

- The best first search algorithm picks the most promising node (in terms of heuristic value) from the OPEN list in every iteration until the goal node is found.
- It maintains the OPEN list as a **priority queue sorted** on the **heuristic value**, so that the node with the best heuristic value automatically comes to the head of the list.
- The algorithm then picks up the node from the head of OPEN.



# BEST FIRST SEARCH 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  $n$ , to generate its successors.

**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 picks the best successor( on the basis of heuristic function value), and then checks 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.



# PROBLEM - I

- Solve the following 8-puzzle problem using Best First Search. State the heuristic used.

2		3
1	8	4
7	6	5

Initial State

1	2	3
8		4
7	6	5

Final State



# SOLUTION- PROBLEM I

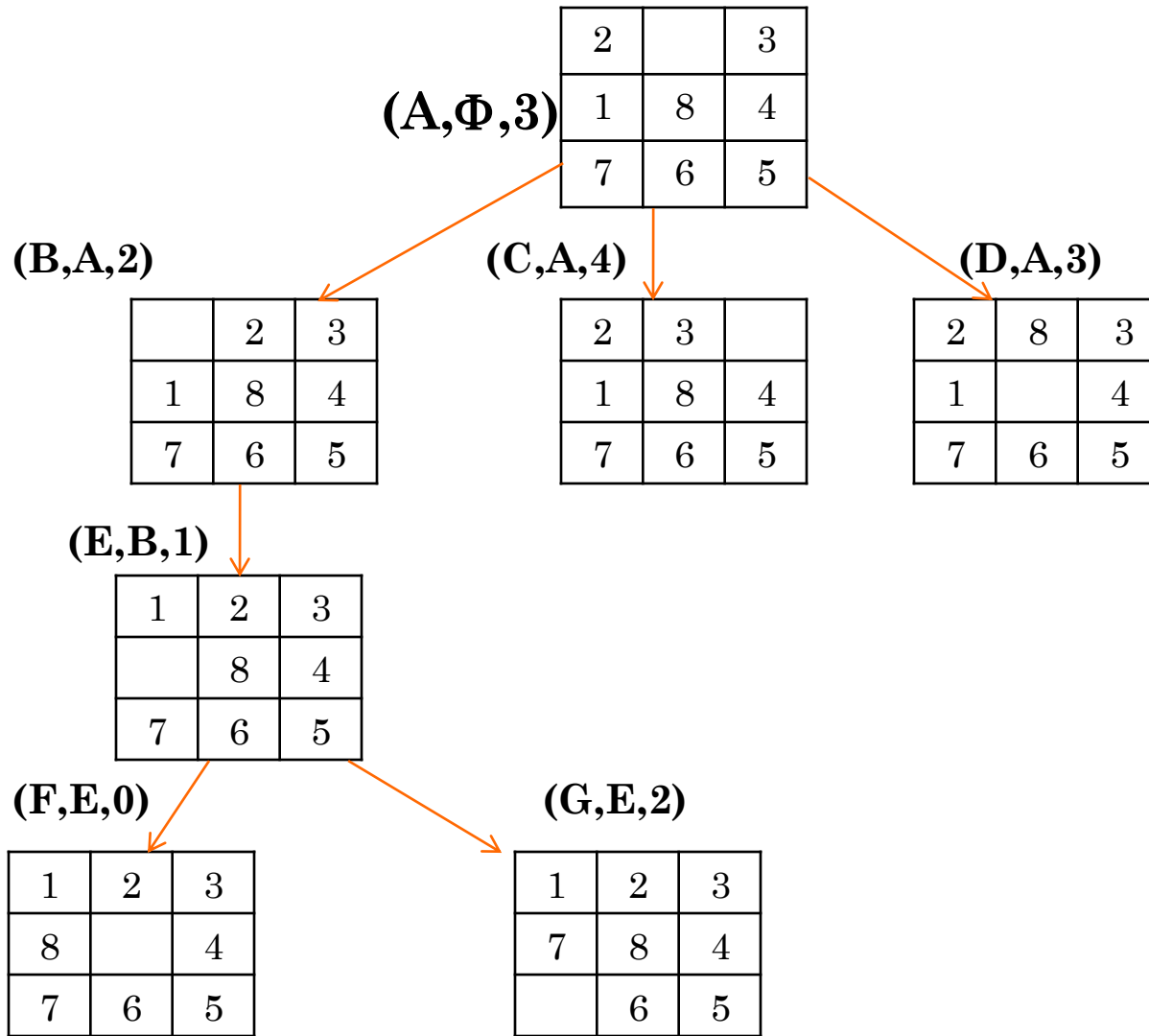
- Let the MOVEGEN function generates successors by moving the blank tile LEFT, RIGHT, UP, DOWN (in the same sequence).
- Let the heuristic function be number of misplaced tiles.
- The OPEN and CLOSED lists while exploring the search space (shown in next slide) are as follows:

Step	OPEN	CLOSED
1	$\{(A, \Phi, 3)\}$	$\{ \}$
2	$\{(B, A, 2), (D, A, 3), (C, A, 4)\}$	$\{(A, \Phi)\}$
3	$\{(E, B, 1), (D, A, 3)(C, A, 4)\}$	$\{(A, \Phi), (B, A)\}$
4	$\{(F, E, 0), (G, E, 2), (D, A, 3)(C, A, 4)\}$	$\{(A, \Phi), (B, A)(E, B)\}$
5	$\{(G, E, 2), (D, A, 3)(C, A, 4)\}$	$\{(A, \Phi), (B, A)(E, B)(F, E)\}$

- Therefore, the path is  $A \rightarrow B \rightarrow E \rightarrow F$



# SOLUTION- PROBLEM I




# PERFORMANCE OF BEST FIRST SEARCH

## ○ Completeness

- The best first search is complete, at least **for finite search space**. This is because it explores the nodes in search space until goal is found or OPEN is empty. The only difference is it explores nodes in order of their heuristic value.
- For **infinite search space**, the completeness of best first search depends upon the heuristic function. If the heuristic function is good enough, the search will end up in goal state.

## ○ Optimality

- The optimality of the best first algorithm also depends upon the heuristic function.
  - The heuristic function may find the optimal path or sub optimal path to problem.
  - If two nodes have same heuristic value but one node is expensive to achieve, the best first search algorithm has no way of discriminating it.
- 

# PERFORMANCE OF BEST FIRST SEARCH CONTD...

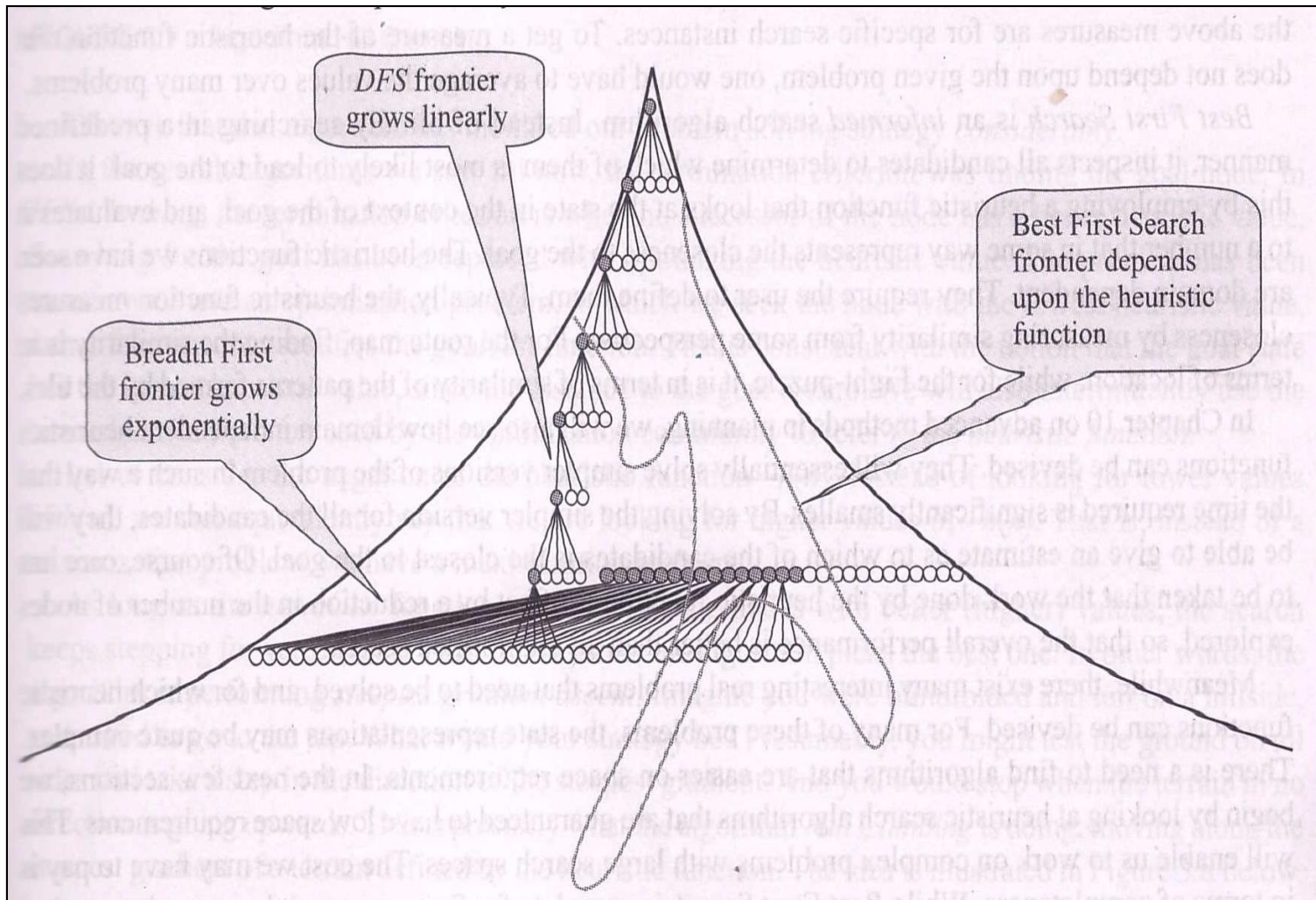
## ○ Space Complexity

- The size of OPEN list depends upon the accuracy of heuristic function.
- If the heuristic function is accurate then the search will find goal directly, and the OPEN will grow linearly.
- Otherwise, the search algorithm may go down some path, change its mind, and move to another branch in search tree.
- For most interesting problems, it is difficult to devise accurate heuristic functions, and thus space grows exponentially (depicted in the figure in next slide).
- **Worst case:**  $O(b^d)$  if heuristic function is not good.

## ○ Time Complexity

- Like space complexity, time complexity is also dependent on the accuracy of heuristic function and time may grow exponentially.

# PERFORMANCE OF BEST FIRST SEARCH CONTD...





# HILL CLIMBING

- The searching algorithm generally moves up in the direction of better value of the heuristic function i.e. uphill/downhill (uphill in maximization and downhill in minimization) .
- It breaks its “moving up/down loop” when it reaches its peak i.e. no neighbor node has a better value.
- It does not maintain a search tree.
- It only looks for the immediate neighbor of the current state. It only stores the current node state and its heuristic function value.
- There are two variants of Hill Climbing:
  - Simple Hill Climbing
  - Steepest Hill Climbing.



# SIMPLE HILL CLIMBING

- Local search algorithm, i.e. it has local domain knowledge and has no global domain knowledge
- Greedy approach: It moves till it gets best move and stops when best move is not found.
- No backtrack.

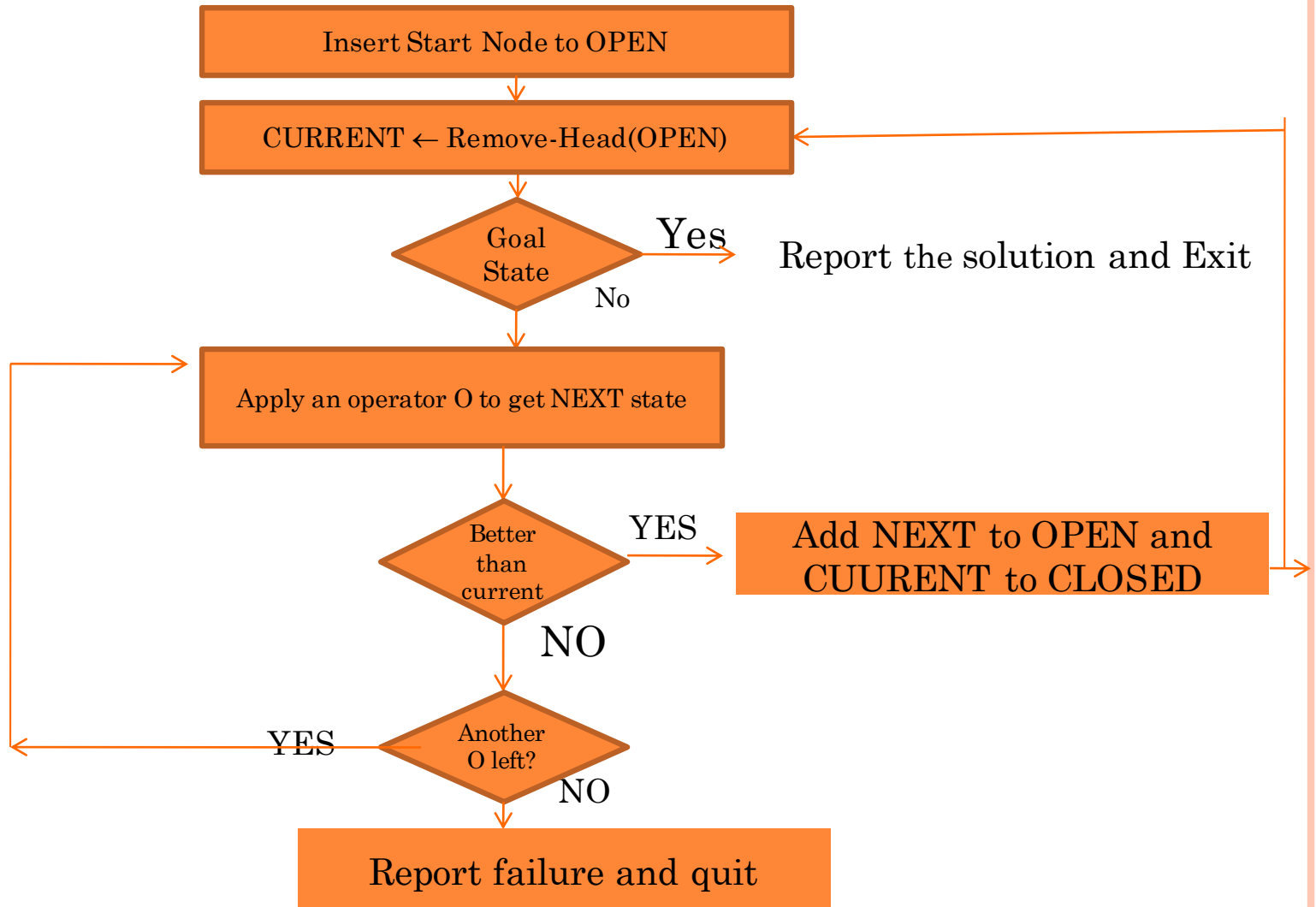


# SIMPLE HILL CLIMBING

- It examines the neighboring nodes one by one and selects the **first** neighboring node which optimizes the current node as next node.
- Simple Hill Climbing algorithm works as follows:
  - Chooses the head node of OPEN as current node.
  - If it is a goal node then it returns success.
  - Otherwise, it **randomly** generates a new next node and checks if its value is better than current node or not. If it is better than current then it is added to OPEN otherwise a different operator/rule is applied to generate the new state.
  - The process continues until a new node is added to OPEN or no operator is left to be applied.



# SIMPLE HILL CLIMBING-FLOW CHART



# PROBLEM - I

- Solve the following 8-puzzle problem using simple hill climbing problem. State the heuristic used.

2		3
1	8	4
7	6	5

Initial State

1	2	3
8		4
7	6	5

Final State



# SOLUTION- PROBLEM I

- Let the heuristic be number of misplaced tiles from goal state. **Any value of HF less than the parent value is a better state.**

**Step1: Initial State(S):**

2		3
1	8	4
7	6	5

$$HF = 1 \text{ (for 1)} + 1 \text{ (for 2)} + 1 \text{ (for 8)} = 3$$

**Step 2: I state:**

	2	3
1	8	4
7	6	5

$$HF = 1 \text{ (for 1)} + 1 \text{ (for 8)} = 2$$

**HF(I) < HF (initial state) . Therefore I state, is current state and new state is produced after applying a valid operator.**

# SOLUTION- PROBLEM I CONTD....

## Step 3: II State:

1	2	3
	8	4
7	6	5

$$HF = 1 \text{ (for 8)} = 1$$

$HF(II) < HF(I)$  . Therefore II state, is current state and new state is produced after applying a valid operator.

## Step 4: III state

1	2	3
8		4
7	6	5

Goal State. Stop the execution of the algorithm.



# SOLUTION-PROBLEM I CONTD...

- The contents of OPEN and CLOSED are as follows:

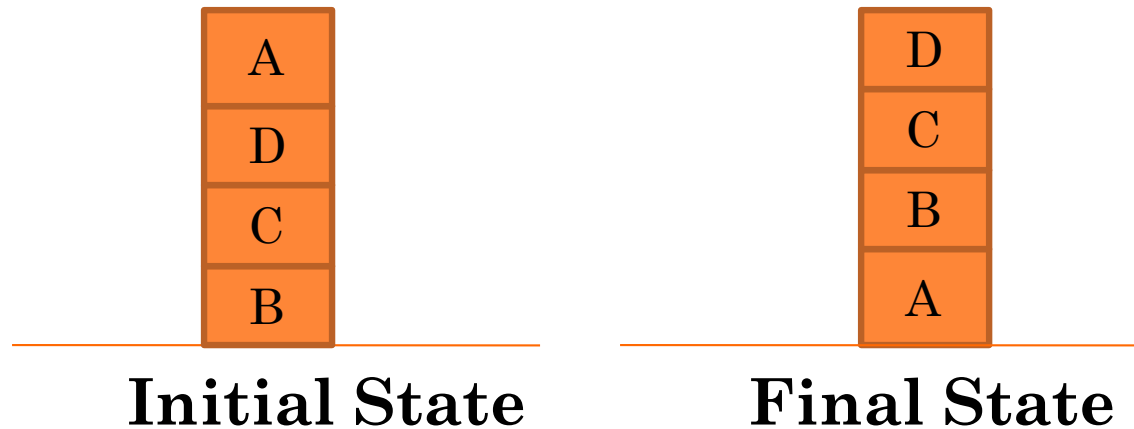
Step	OPEN	CLOSED
1	$\{(S, \Phi, 3)\}$	$\{ \}$
2	$\{(I, S, 2)\}$	$\{(S, \Phi)\}$
3	$\{(II, I, 1)\}$	$\{(S, \Phi)(I, S)\}$
4	$\{(III, II, 0)\}$	$\{S, \Phi)(I, S)(II, I)\}$
5	$\{ \}$	$\{(S, \Phi)(I, S)(II, I)(III, II)\}$





## PROBLEM II

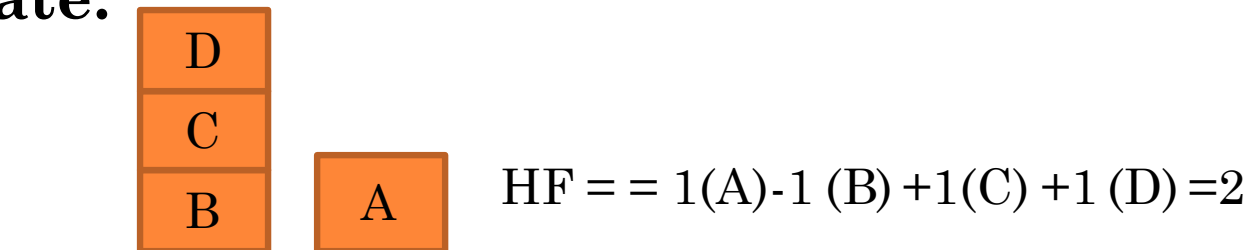
- Find the solution to the following blocks world problem using simple hill method.



# SOLUTION-PROBLEM II

- Let the heuristic be +1 if the block is resting on the correct block and -1 if it is resting on the incorrect block.
- $HF(\text{Initial State}, S) = -1(A) + 1(D) + 1(C) - 1(B) = 0$
- Successor of Initial state

Step 1: **I state:**



Better than initial state (No need to apply any rule). Therefore Current state = First State

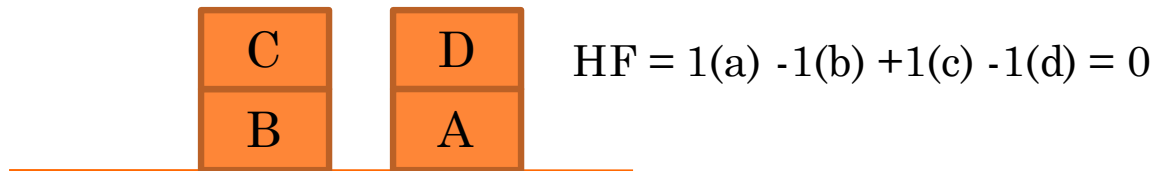
Step 2: **II state**



$HF(II) < HF(I)$ . So state is rejected. Apply another operator on state I.

# SOLUTION-PROBLEM II CONTD...

## III state:



Again  $HF(III) < HF(I)$  and no other operator is possible on state I. Therefore, return fail. (Local Maxima)

The contents of OPEN and CLOSED at each step according to first heuristic is:

Step	OPEN	CLOSED
1	$\{(S, \Phi, 0)\}$	$\{ \}$
2	$\{(1, S, 2)\}$	$\{(S, \Phi, 0)\}$
3	$\{ \}$	$\{(S, \Phi), (1, S)\}$



# SOLUTION-PROBLEM II CONTD...

Let another heuristic be :

+n for block which is resting on the current support structure and n is equal to number of blocks below it.

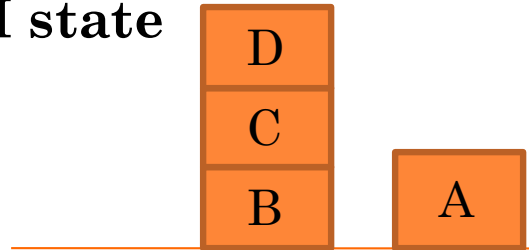
-n for block which is resting on the incurrent support structure and n is equal to number of blocks below it.

$$HF(\text{Initial State}, S) = -3 (A) - 0 (B) - 1 (C) - 2 (D) = -6$$



# SOLUTION-PROBLEM II CONTD...

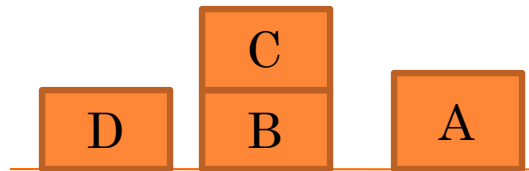
**Step 1: I state**



$$HF(I) = 0 (A) - 0 (B) - 1 (C) - 2 (D) = -3$$

$HF(I) > HF(\text{Initial State})$ . Therefore I state is current state and an operator is applied on it.

**Step2: II state**



$$HF(II) = 0 (A) - 0 (B) - 1 (C) - 0 (D) = -1$$

$HF(II) > HF(I)$ . Therefore II state is current state and an operator is applied on it.



# SOLUTION-PROBLEM II CONTD....

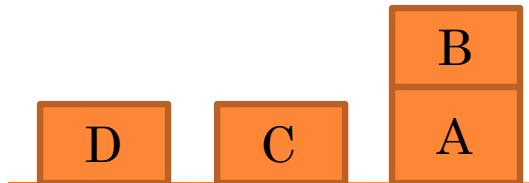
## Step 3: III state



$$HF(III) = 0 (A) - 0 (B) - 0 (C) - 0 (D) = 0$$

$HF(III) > HF(II)$ . Therefore III state is current state and an operator is applied on it.

## Step 4: IV state

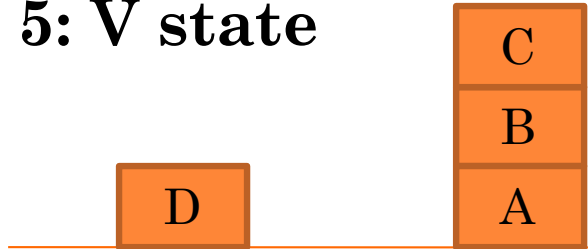


$$HF(IV) = 0 (A) + 1 (B) - 0 (C) - 0 (D) = 1$$

$HF(IV) > HF(III)$ . Therefore IV state is current state and an operator is applied on it.

# SOLUTION-PROBLEM II CONTD...

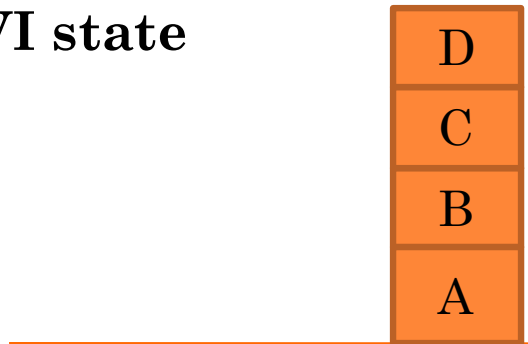
**Step 5: V state**



$$HF(V) = 0 (A) + 1 (B) + 2 (C) - 0 (D) = 3$$

$HF(V) > HF(IV)$ . Therefore V state is current state and an operator is applied on it.

**Step 6: VI state**



VI state is goal state. So report it and stop.



## SOLUTION-PROBLEM II CONTD...

- The contents of OPEN and CLOSED according to second heuristic are as follows:

Step	OPEN	CLOSED
1	$\{(S, \Phi, -6)\}$	$\{ \}$
2	$\{(I, S, -3)\}$	$\{(S, \Phi)\}$
3	$\{(II, I, -1)\}$	$\{(S, \Phi)(I, S)\}$
4	$\{(III, II, 0)\}$	$\{S, \Phi)(I, S)(II, I)\}$
5	$\{(IV, III, 1)\}$	$\{(S, \Phi)(I, S)(II, I)(III, II)\}$
6	$\{(V, IV, 3)\}$	$\{(S, \Phi)(I, S)(II, I)(III, II)(IV, III)\}$
7	$\{(VI, V, 6)\}$	$\{(S, \Phi)(I, S)(II, I)(III, II)(IV, III)(V, IV)\}$
8	$\{ \}$	$\{(S, \Phi)(I, S)(II, I)(III, II)(IV, III)(V, IV)(VI, V)\}$





# STEEPEST HILL CLIMBING

- Simple hill climbing algorithm selects the **first state** which is better than the current state and **does not explore the rest of the states** at that level which could be even better than the selected state.
- This problem is overcome in steepest hill climbing that selects the best among the children state as the current state.
- Steepest hill climbing is also called **gradient search**.



# STEEPEST HILL CLIMBING

- It first examines all the neighboring nodes and then selects the node closest to the solution state as next node.
- Simple Hill Climbing algorithm works as follows:
  - Chooses the head node of OPEN as current node and add it to CLOSED.
  - If it is a goal node then it returns success.
  - Otherwise, it applies **all operators/rules** on the current state and **picks up the best node**. If the best node is better than current then it is added to OPEN.
  - The process continues until OPEN is empty or success is returned.



# STEEPEST HILL CLIMBING-ALGORITHM

## ○ ALGORITHM

Insert the start node in OPEN

Loop until success is returned or OPEN is empty

Node  $(N, P, H_N) \leftarrow \text{Remove-Head}(\text{OPEN})$  and add it to CLOSED

If N is goal node then return success & path to N

Else

Apply all operators to generate NEW nodes and pick the best node

i.e.  $\text{NEXT} \leftarrow \max_h(\text{append}(\text{NEW}, \text{OPEN}))$

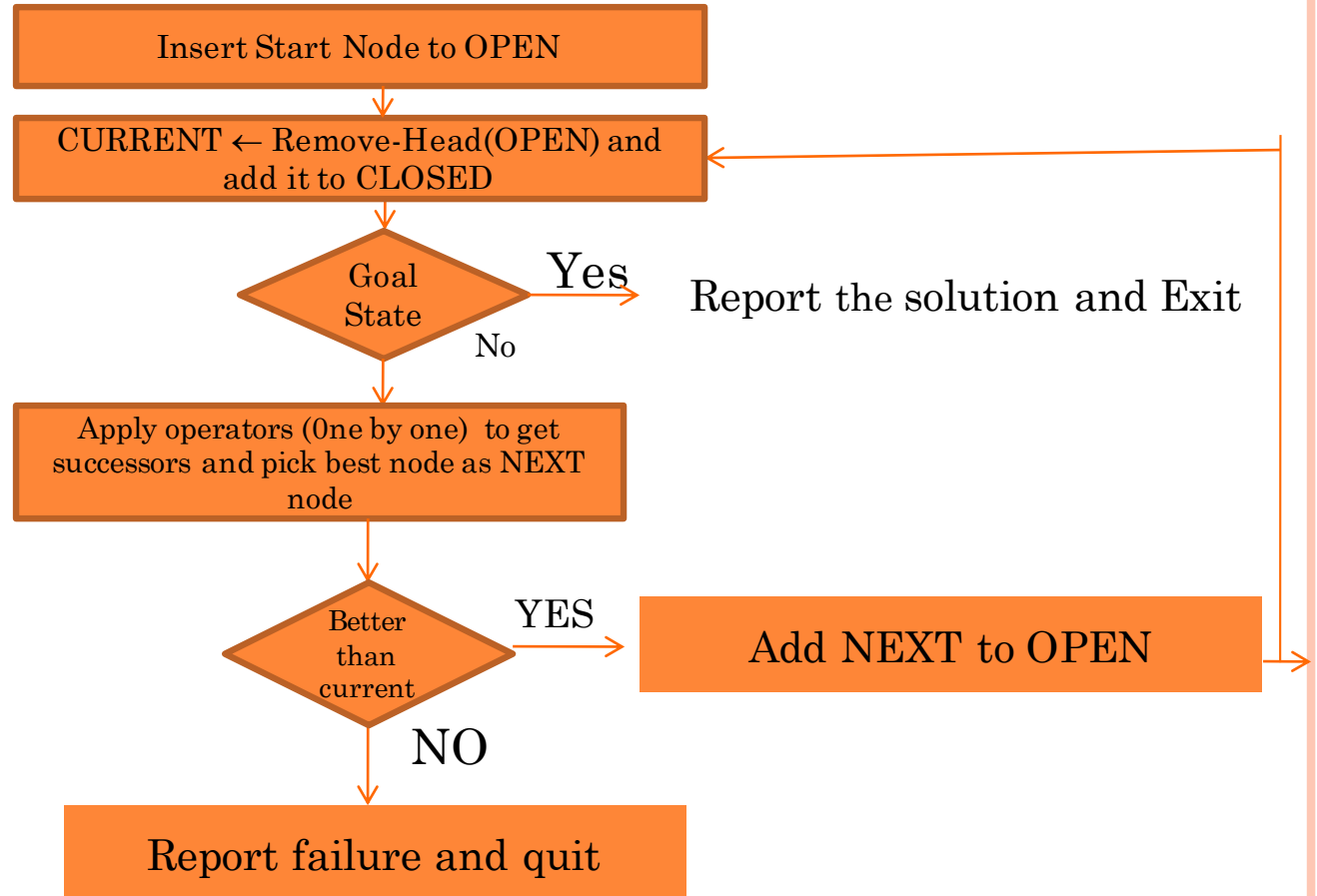
If  $h(\text{NEXT})$  is better than  $h(N)$  then add NEXT to OPEN

End If

End Loop



# STEEPEST HILL CLIMBING-FLOW CHART



# PROBLEM - I

- Solve the following 8-puzzle problem using steepest hill climbing problem. State the heuristic used.

2	8	3
1	5	4
7	6	

Initial State

1	2	3
8		4
7	6	5

Final State



# SOLUTION- PROBLEM I



**(B, A, 2)**

	2	3
1	8	4
7	6	5

**(C, A, 4)**

2	3	
1	8	4
7	6	5

**(D, A, 3)**

2	8	3
1		4
7	6	5

Step 2: OPEN = {(B, A, 2)}  
CLOSED = {(A,  $\Phi$ )}

**(E, B, 1)**

1	2	3
	8	4
7	6	5

Step 3: OPEN = {(E, B, 1)} CLOSED = {(A,  $\Phi$ ), (B, A)}

**(F, E, 0)**

1	2	3
8		4
7	6	5

**(G, E, 2)**

1	2	3
7	8	4
	6	5

Step 4: OPEN = {(F, E, 0)} CLOSED = {(A,  $\Phi$ ), (B, A), (E, B)}

Step 5: OPEN = {} CLOSED = {(A,  $\Phi$ ), (B, A), (E, B), (F, E)}

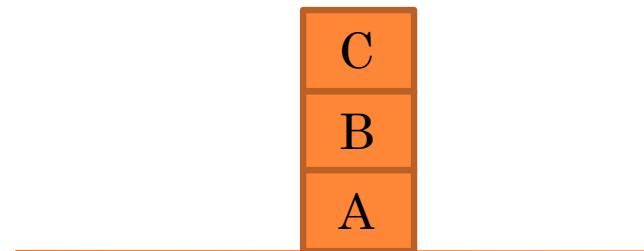


## PROBLEM- II

- Find the solution to the following blocks world problem using steepest hill method. State the heuristics used.



**Initial State**



**Final State**



# SOLUTION- PROBLEM-II

Let heuristic be :

- + score for block which is resting on the correct support structure and that score is equal to number of blocks below it.
- score for block which is resting on the incorrect support structure and that score is equal to number of blocks below it.

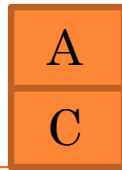




# SOLUTION- PROBLEM -II

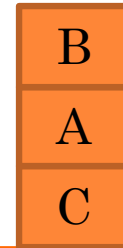
$$HF = -1 - 0 - 0 = -1$$

State 1

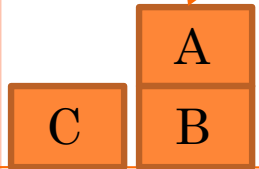


$$HF = 0$$

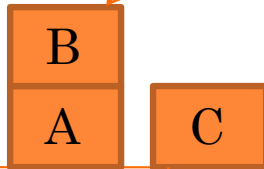
State 2



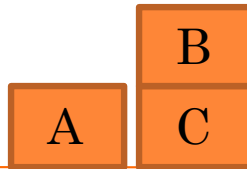
$$HF = -3$$



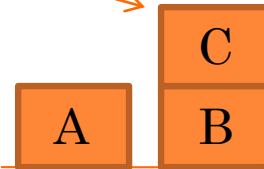
$$HF = -1$$



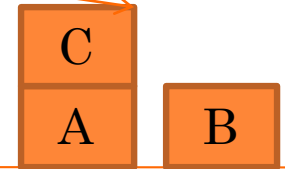
$$HF = 1$$



$$HF = -1$$

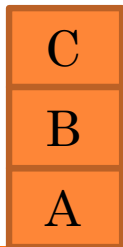


$$HF = -1$$



$$HF = -1$$

State 3



State 4



## SOLUTION- PROBLEM -II

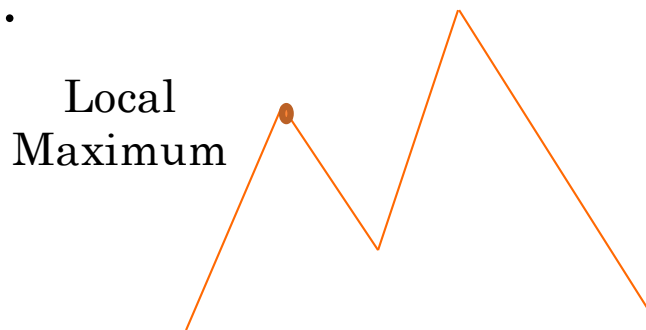
- The contents of OPEN and CLOSED are as follows:

Step	OPEN	CLOSED
1	$\{(1, \Phi, -1)\}$	$\{ \}$
2	$\{(2, 1, 0)\}$	$\{(1, \Phi)\}$
3	$\{(3, 2, 1)\}$	$\{(1, \Phi)(2, 1)\}$
4	$\{(4, 3, 3)\}$	$\{(1, \Phi), (2, 1), (3, 2)\}$
5	$\{ \}$	$\{(1, \Phi), (2, 1), (3, 2)(4, 3)\}$



# LIMITATIONS OF HILL CLIMBING ALGORITHMS

- Both simple hill climbing and steepest hill climbing methods may fail to provide a solution by reaching a state from which no subsequent improvement can be made and this state is not the solution (goal state).
- This will happen if the program has reached **local maximum, a plateau or a ridge**.
- **Local Maximum** is a state that is better than all its neighbor but is not better than some other states.
- At local maximum, all moves appear to make things worse.



# LIMITATIONS OF HILL CLIMBING ALGORITHMS

## CONTD....

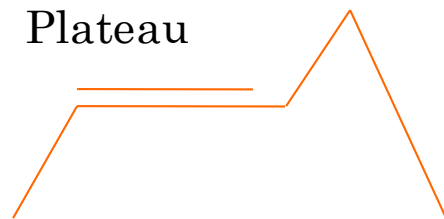
- Solution to problem of local maximum can be backtracking to some earlier node and try going to a different direction.
- This is particularly reasonable if at that node there was another direction that looked as promising as the one that was earlier chosen.
- To implement this strategy, maintain a list of paths taken and go back to one of them is the path that was taken leads to dead end.



# LIMITATIONS OF HILL CLIMBING ALGORITHMS

## CONTD....

- A plateau is another problem in hill climbing algorithms . A plateau is a flat area of the search space in which a whole set of neighboring states have the same value
- On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.

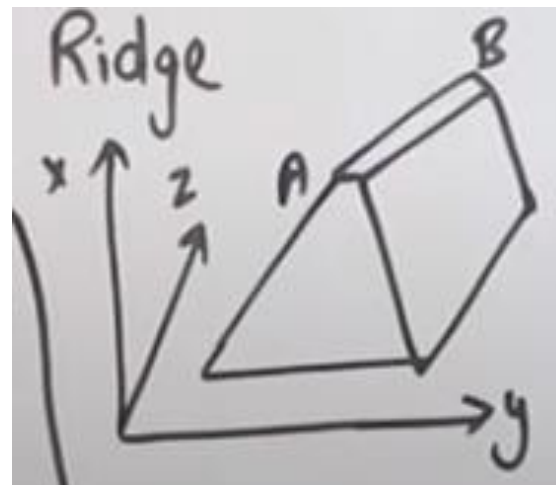


- If we are caught in this type of problem, then make a big jump to get new section of the search space.
- If the only rules available describe single small steps, apply them several times in same direction.

# LIMITATIONS OF HILL CLIMBING ALGORITHMS

## CONTD....

- A **ridge** is another problem in hill climbing algorithms.
- A ridge is a special kind of local maximum. It is an area of the search space that is higher than the surrounding areas and that itself has a slope.
- But the orientation of the high region, compared to the set of available moves and the directions in which they move, makes it impossible to traverse a ridge by single moves.
- In order to get through this problem, apply two or more rules before doing the test and this corresponds to moving in **special directions**.



# PERFORMANCE OF HILL CLIMBING ALGORITHM


## ○ Completeness:

- The hill climbing algorithms are not complete.
- The completeness of the hill climbing algorithm depends upon the quality of heuristic function.
- A heuristic function may take the search to the global maxima or minima or may stuck into local maxima/minima, plateau, or ridge.

## ○ Optimality:

- Like Best First Search, the optimality of hill climbing algorithm **depends upon the heuristic function**.

## ○ Search Complexity:

- This is the strongest feature of hill climbing.
  - It requires a constant amount of space as it need to store only copy of the current state.
- 

# PERFORMANCE OF HILL CLIMBING ALGORITHM

## CONTD....

### ○ Time Complexity:

- In hill climbing algorithm, each node generate at the most  $b$  nodes.
- Therefore, the time complexity grows linearly and is of the order  $O(b^d)$  where  $b$  is the branching factor and  $d$  is the depth at which solution lies.





# BEAM SEARCH

- It is a heuristic approach where only the most promising  $\beta$  nodes (instead of all nodes) at each step of the search are retained for further branching.
- $\beta$  is called **Beam Width**.
- Beam search is an **optimization of best-first search** that reduces its memory requirements.
- At each level of the tree it generates **all successors** of the states at the current level, and sorts them in order of heuristic function at each level, **keeps  $\beta$  nodes** and **ignores the rest**.



# BEAM SEARCH ALGORITHM

- OPEN- Priority Queue with generated nodes of the form (Node,Parent,Heuristic) sorted on the values of Heuristic.
- CLOSED- Queue of expanded nodes; initially empty
- **ALGORITHM**

Insert the start node in OPEN

Loop until success is returned or OPEN is empty

Node (N,P,H<sub>N</sub>) ← Remove-Head(OPEN) and add it to CLOSED

If N is goal node return success & path to N

Else

Generate NEW successors of N, append it to OPEN and sort the list in order of heuristic values

If  $|\text{OPEN}| > \beta$ , take the best  $\beta$  nodes (according to heuristic) and remove the others from the OPEN.

End If

End Loop



# PERFORMANCE OF BEAM SEARCH

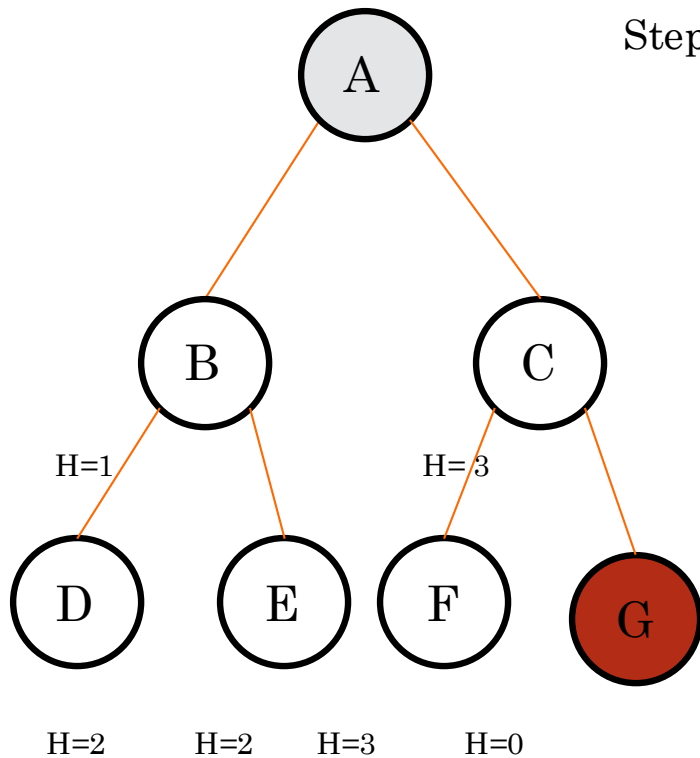
## ○ Completeness:

- In general, the Beam Search Algorithm is **not complete**.
- Even given unlimited time and memory, it is possible for the algorithm to miss the goal node when there is a path from the start node to the goal node (example in next slide).
- A **more accurate heuristic function** and a **larger beam width** can improve Beam Search's chances of finding the goal.

## ○ Optimality

- Just as the Algorithm is not complete, it is also not guaranteed to be optimal.
- This can happen because the beam width and an inaccurate heuristic function may cause the algorithm **to miss expanding the shortest path**.
- A more precise heuristic function and a larger beam width can make Beam Search more likely to find the optimal path to the goal.

# INCOMPLETE WITH $\beta=2$ BUT COMPLETE WITH $\beta=3$



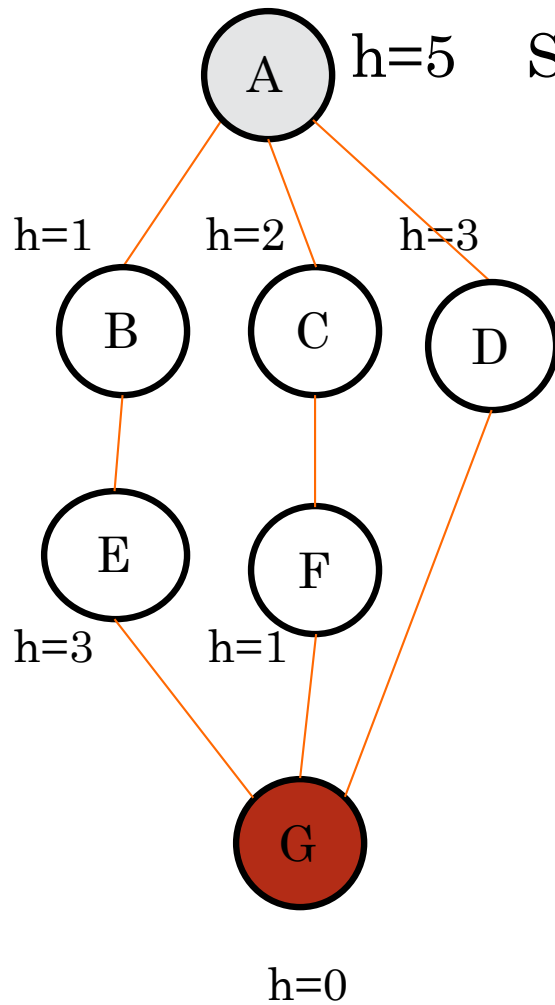
Steps:

1. OPEN =  $\{(A, \Phi, 0)\}$ , CLOSED =  $\{\}$
2. OPEN =  $\{(B, A, 1)(C, A, 3)\}$  CLOSED =  $\{(A, \Phi)\}$
3. OPEN =  $\{(D, B, 2)(E, B, 2)\}$  CLOSED =  $\{(A, \Phi)(B, A)\}$
4. OPEN =  $\{(E, B, 2)\}$  CLOSED =  $\{(A, \Phi)(B, A)(D, B)\}$
5. OPEN =  $\{\}$  CLOSED =  $\{(A, \Phi)(B, A)(D, B)(E, B)\}$

Clearly, OPEN set becomes empty without finding goal node G .  
With  $\beta = 3$ , the algorithm succeeds to find goal node G.



# NON OPTIMAL SOLUTION WITH $\beta=2$



Steps:

1. OPEN =  $\{(A, \Phi, 5)\}$  CLOSED =  $\{\}$
2. OPEN =  $\{(B, A, 1) (C, A, 2)\}$  CLOSED =  $\{(A, \Phi)\}$
3. OPEN =  $\{(C, A, 2), (E, B, 3)\}$  CLOSED =  $\{(A, \Phi) (B, A)\}$
4. OPEN =  $\{(F, C, 1) (E, B, 3)\}$  CLOSED =  $\{(A, \Phi) (B, A) (C, A)\}$
5. OPEN =  $\{(G, F, 0) (E, B, 3)\}$  CLOSED =  $\{(A, \Phi) (B, A) (C, A) (F, C)\}$
6. Found goal node, stop.

Path : A->C->F->G

But Optimal path is A->D->G



# PERFORMANCE OF BEAM SEARCH CONTD...

## ○ Space Complexity:

- The beam search algorithm has **constant space complexity** as it needs to store  $\beta$  nodes at each level.

## ○ Time Complexity:

- The beam search at each level at the maximum generate  $\beta \times b$  nodes.
- Therefore, the time complexity is of the order,  
 $O(\text{depth} \times \text{beam width} \times \text{branching factor})$

