

# Heuristic Searching

## **Kecerdasan Komputasional**

# Algoritma Pencarian (Searching)

- Merupakan algoritma untuk mencari kemungkinan penyelesaian
- Sering dijumpai oleh peneliti di bidang AI
- Dalam melakukan pencarian, salah satu cara yang banyak digunakan untuk menggambarkan masalah adalah dengan mencantumkan atau menggambarkan semua kemungkinan keadaan yang ada.
- Memecahkan masalah berarti bergerak atau berpindah dari satu ruang (node) dari titik awal sampai titik yang dituju (ditentukan), oleh karena itu kita memerlukan satu set operator untuk bergerak dari satu node ke node lainnya.

# Mendefinisikan permasalahan

- Mendefinisikan suatu **state space** (ruang keadaan)
- Menetapkan satu atau lebih **state awal**
- Menetapkan satu atau lebih **state tujuan**
- Menetapkan **rules** (kumpulan aturan)
  - Generalisasi
  - Asumsi

# Atribut Pencarian

- **Optimalisasi:** apakah algoritma dapat menemukan cost path terendah untuk mencapai goal?
- **Kelengkapan:** apakah algoritma akan menemukan jalur menuju goal/tujuan jika memang ada?
  - digunakan untuk mendefinisikan “return failure otherwise”
  - jika tidak ada solusi, dan algoritma tidak dapat mendeteksinya (mendeteksi state berulang-ulang(loop)), maka akan menjadi infinite search dan tidak akan ada hasil yang dikembalikan.
- **Kompleksitas Waktu:** waktu yang dibutuhkan untuk melakukan pencarian (contoh: jumlah nodes yang digeneralisasikan selama proses pencarian).
- **Kompleksitas waktu:** memori yang dibutuhkan.

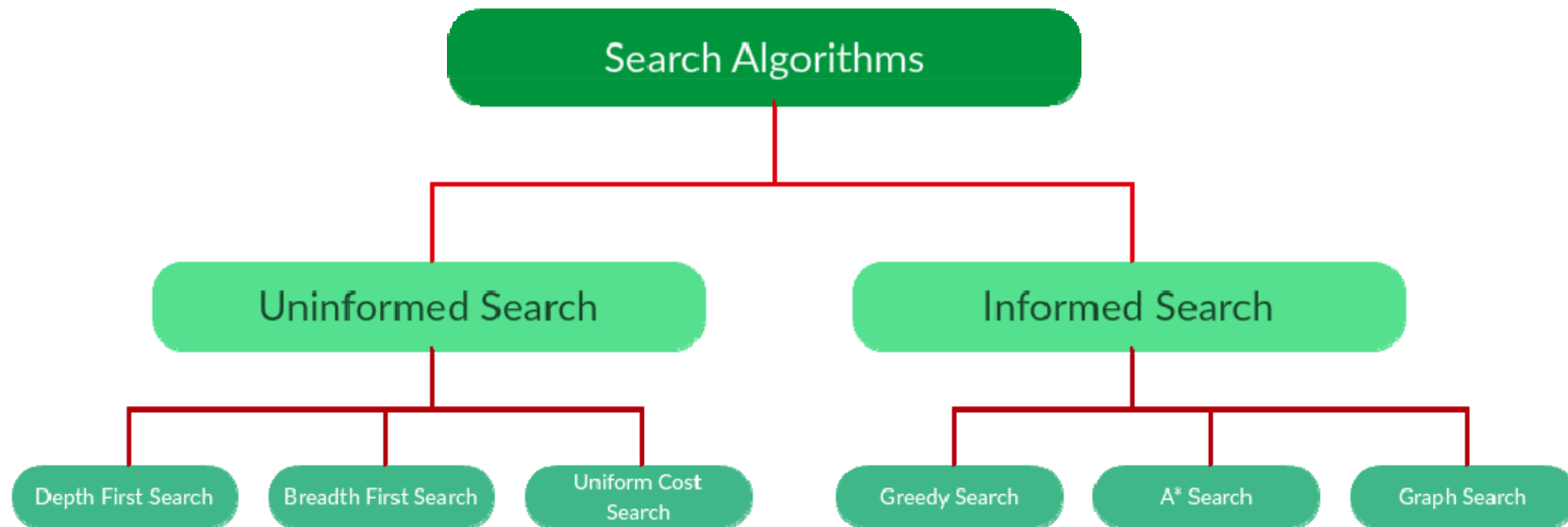
## Kriteria mengukur perfomansi metode pencarian

- ***Completeness*** : apakah metode tersebut menjamin penemuan solusi jika solusinya memang ada?
- ***Time complexity*** : berapa lama waktu yang diperlukan?  
[semakin cepat, semakin baik]
- ***Space complexity*** : berapa banyak memori yang diperlukan
- ***Optimality*** : apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi berbeda?

# Metode searching

- Terdapat dua metode untuk melakukan pencarian yaitu dengan
  - BLIND SEARCH
  - HEURISTIC SEARCH

# Types of search algorithms



# **HEURISTIC SEARCHING**

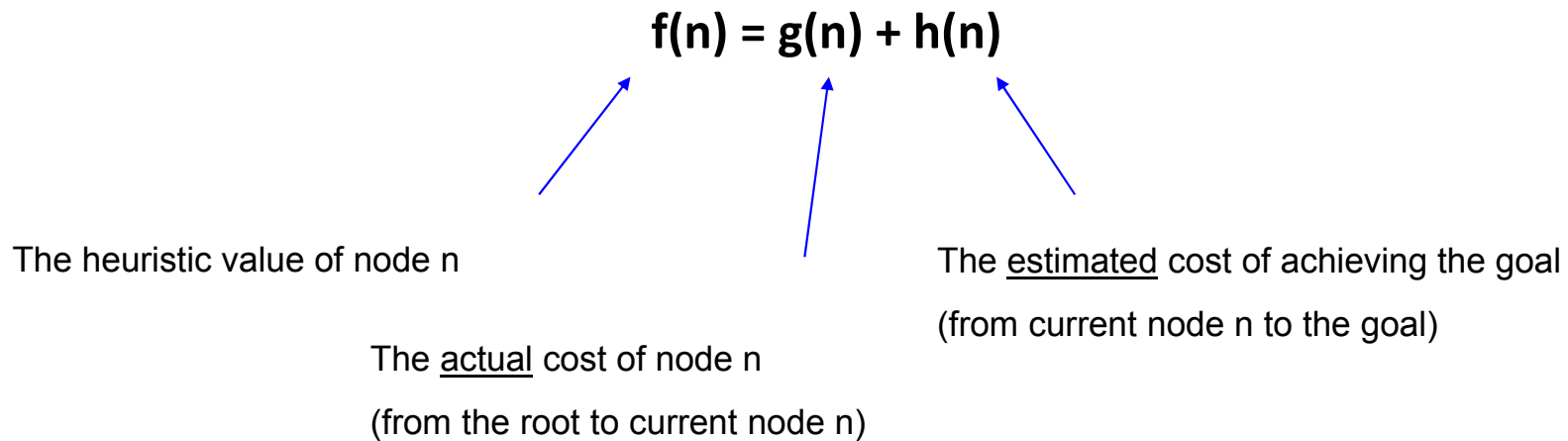
## **(Informed searching)**



# Metode pencarian heuristics

- Pencarian heuristik adalah teknik pencarian yang menggunakan **metode heuristik** untuk pergerakannya (berpindah).
- Heuristik sendiri merupakan **aturan praktis yang mungkin mengarah pada solusi**.
- **Heuristik membantu mengurangi jumlah alternatif dari bilangan eksponensial menjadi bilangan polinomial.**
- Here, the **algorithms have information** on the goal state, which helps in more **efficient searching**.
- Dalam pencarian heuristik setiap state diberi sebuah “**heuristic value**” (**h-value**) yang digunakan pencarian dalam **memilih langkah terbaik selanjutnya**.
- Contoh
  - Generate and Test
  - Hill Climbing
  - Best First Search (A\* dan Greedy)
  - Branch and bound
  - Dynamic programming
  - Alpha Beta Pruning, dll

## What is in a “heuristic?”



### Heuristics Search :

- In an informed search, a **heuristic is a *function* that estimates how close a state is to the goal state.**
- For examples – Manhattan distance, Euclidean distance, etc.
- Lesser the distance, closer the goal.
- Different heuristics are used in different informed algorithms.

# Heuristic / Informed Search Algorithms

- This information is obtained by something called a *heuristic*.
- Example of Informed Search Algorithms:
  1. Generate and Test
  2. Best First Search
    - 2.1. Greedy Search
    - 2.2. A\* Tree Search
  3. A\* Graph Search
  4. Hill Climbing
  5. Branch and bound
  6. Dynamic programming
  7. Alpha Beta Pruning

## Heuristic Searching 1.

### Generate and Test Algorithm

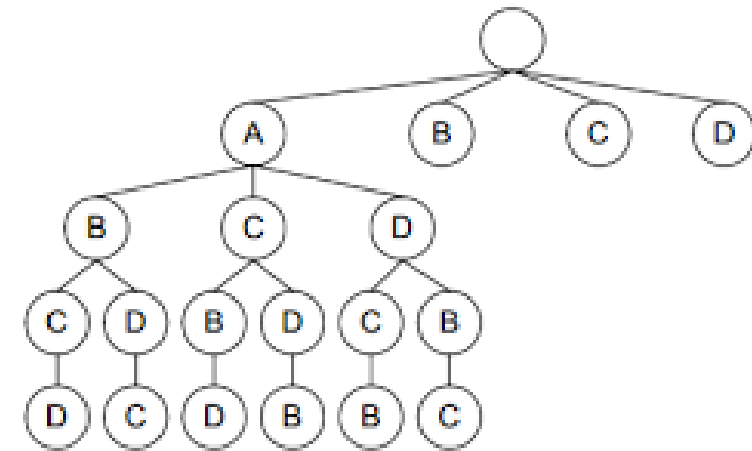
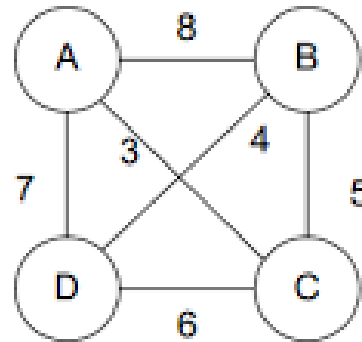
- Merupakan algoritma searching yang paling sederhana yang **menjamin untuk menemukan solusi jika dilakukan secara sistematis dan ada solusinya**
- Pada prinsipnya metode ini merupakan penggabungan antara depth-first search dengan pelacakan mundur (backtracking), yaitu bergerak ke belakang menuju pada suatu keadaan awal.
- Efisien untuk masalah sederhana.

# Kelemahan Generate and Test algorithm

- Kurang efisien untuk masalah yang besar atau kompleks.
- Perlu membangkitkan semua kemungkinan sebelum dilakukan pengujian
- Membutuhkan waktu yang cukup lama dalam pencariannya karena harus menentukan semua kemungkinan terlebih dahulu sebelum melakukan pengujian.

# Contoh Generate & Test : “Travelling Salesman Problem (TSP)”

- Seorang salesman ingin mengunjungi n kota.
- Jarak antara tiap-tiap kota sudah diketahui.
- Kita ingin mengetahui rute terpendek dimana setiap kota hanya boleh dikunjungi tepat 1 kali.
- Misalkan ada 4 kota (A, B, C, dan D) dengan jarak antara tiap-tiap kota seperti tampak pada graph.



Alur pencarian dengan *Generate and Test*

Pencarian ke-	Lintasan	Panjang Lintasan	Lintasan Terpilih	Panjang Lintasan Terpilih
1	ABCD	19	ABCD	19
2	ABDC	18	ABDC	18
3	ACBD	12	ACBD	12
4	ACDB	13	ACBD	12
5	ADBC	16	ACBD	12
Dst...				

Algoritma  
**Generate & Test**



# Algoritma Generate and Test

1. Tentukan suatu kemungkinan solusi (menentukan suatu titik tertentu atau lintasan tertentu dari keadaan awal).
2. Uji untuk melihat apakah node tersebut benar-benar merupakan solusinya dengan cara membandingkan node tersebut atau node akhir dari suatu lintasan yang dipilih dengan kumpulan tujuan yang diharapkan.
3. Jika solusi ditemukan, keluar. Jika tidak, ulangi kembali langkah yang pertama.

## Heuristic Searching 2.

### Best First Search Algorithm

<https://www.mygreatlearning.com/blog/best-first-search-bfs/>

- Use an **evaluation function to decide which among the various available nodes is the most promising** (or 'BEST') before traversing to that node.
- The Best first search uses the concept of a Priority queue and heuristic search.
- To search the graph space, the BFS method uses two lists for tracking the traversal.
- An 'OPEN' list which keeps track of the current 'immediate' nodes available for traversal and 'CLOSED' list that keeps track of the nodes already traversed.



# Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
  2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
  3. Repeat the next steps until GOAL node is reached
    1. If OPEN list is empty, then EXIT the loop returning 'False'
    2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also capture the information of the parent node
    3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
    4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
    5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function  $f(n)$
- This algorithm will traverse the shortest path first in the queue.
  - The time complexity of the algorithm is given by  $O(n \cdot \log n)$ .

# Variants of Best First Search

## 2.1. Greedy Best First Search

## 2.2. A\* Best First Search.

- **The only difference between Greedy BFS and A\* BFS is in the evaluation function.**
- For **Greedy BFS**, the evaluation function is  **$f(n) = h(n)$** .
- For **A\* BFS**, the evaluation function is  **$f(n) = g(n) + h(n)$** .

## 2.1. Greedy Best First Search Algorithm (1)

<https://www.javatpoint.com/ai-informed-search-algorithms>

<https://www.mygreatlearning.com/blog/best-first-search-bfs/>

- The Greedy BFS algorithm **selects the path which appears to be the best** at the moment.
- It is **the combination of depth-first search and breadth-first search** algorithms.
- It uses the heuristic function and search.
- Best-first search allows us to take the advantages of both algorithms.
- With the help of best-first search, at each step, we can choose **the most promising node**.

# Greedy Best First Search Algorithm (2)

<https://www.geeksforgeeks.org/search-algorithms-in-ai/>

- 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.

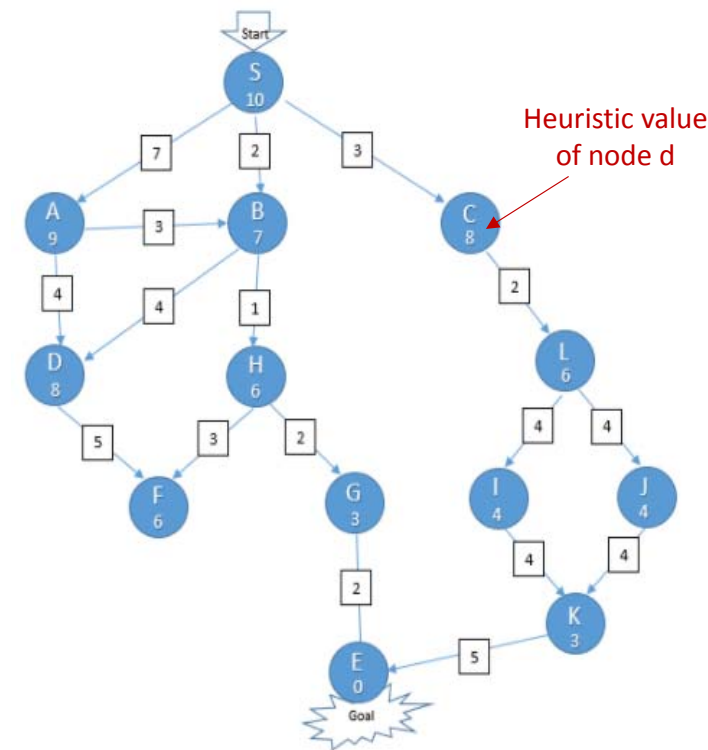
$$f(n) = h(n)$$

- Where,  **$h(n)$**  is
  - **estimated cost from node  $n$  to the goal.**
  - **The “closeness” is estimated by a heuristic  $h(x)$ .**
  - The lower the value of  $h(x)$ , the closer is the node from the goal.
- The greedy best first algorithm is implemented by the **priority queue**.
- **Strategy:** Expand the node closest to the goal state, i.e. expand the node with lower  $h$  value.

# Greedy Best First Search Algorithm (3)

## Example of application

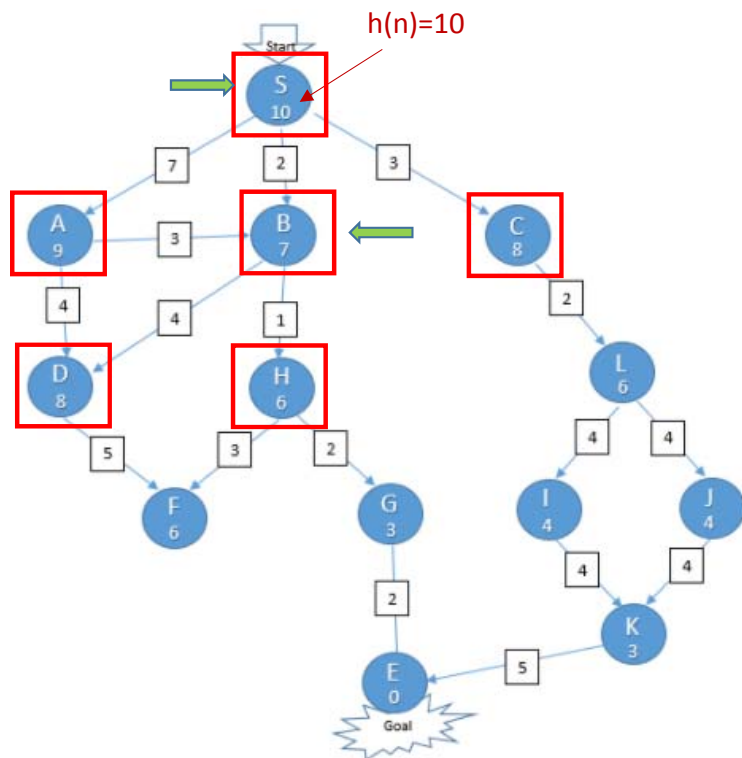
- Let's say we want to drive from city S to city E in the **shortest possible road distance**, and we want to do it in the fastest way, **by exploring the least number of cities in the way**, i.e. **the least number of steps**.
- Whenever we arrive at an intermediate city**, we get to know the trip distance from that city to our goal city E. **This distance is an approximation of how close we are to the goal from a given node and is denoted by the heuristic function  $h(n)$** . **This heuristic value is mentioned within each node**. However, note that this is not always equal to the actual road distance, as the road may have many curves while moving up a hill, and more.
- Also, **when we travel from one node to the other**, we get to know the **actual road distance between the current city and the immediate next city** on the way and is mentioned over the paths in the given figure. **The sum of the distance from the start city to each of these immediate next city is denoted by the function  $g(n)$** .
- At any point, the decision on which city to go next is governed by our evaluation function. The **city which gives the least value for this evaluation function will be explored first**.



# Greedy 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  $n$ , 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 1: Greedy Best-first Search Algorithm (1) $f(n) = h(n)$



**Greedy BFS with evaluation function  $f(n) = h(n)$**

Step 1 - Start by adding the start node (S) to the open list with the path distance as 0

OPEN		CLOSED	
Node	$h(n)$	Node	Parent Node
S	10		

Repeat the next steps until the OPEN List is empty or the Goal node is moved to the CLOSED list

Step 2 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	$h(n)$	Node	Parent Node
A	9	S	
B	7		
C	8		

RE-ORDER

Step 2 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN		CLOSED	
Node	$h(n)$	Node	Parent Node
B	7	S	
C	8		
A	9		

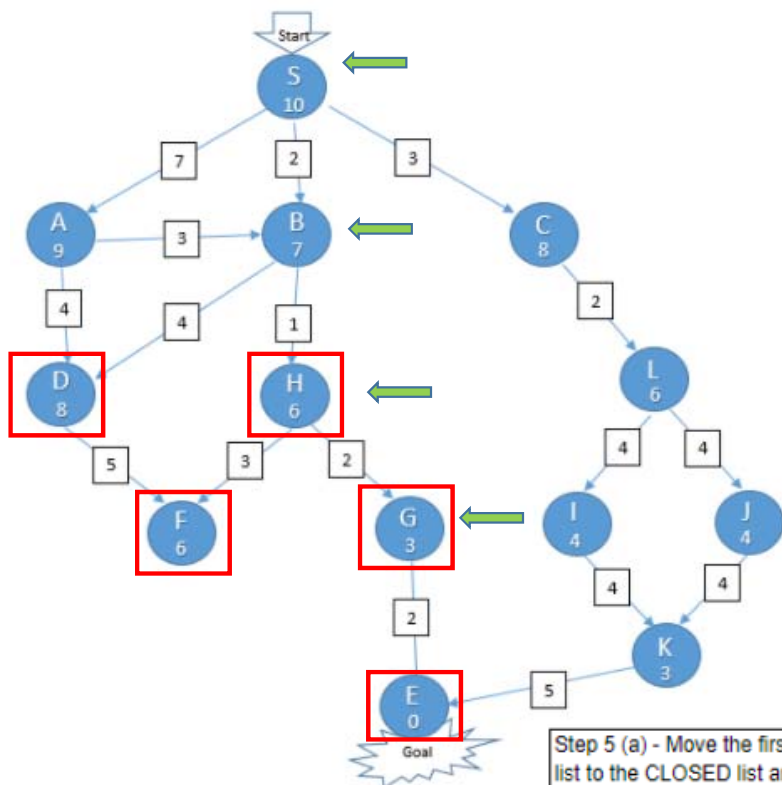
RE-ORDER

Step 3 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	$h(n)$	Node	Parent Node
C	8	S	
A	9	B	S
D	8		
H	6		

Step 3 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN		CLOSED	
Node	$h(n)$	Node	Parent Node
H	6	S	
C	8	B	S
D	8		
A	9		



**Example 1: Greedy Best-  
first Search Algorithm (2)**

Step 4 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
C	8	S	
D	8	B	S
A	9	H	B
F	6		
G	3		

## RE-ORDER

Step 4 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
G	3	S	
F	6	B	S
C	8	H	B
D	8		
A	9		

## RE-ORDER

Step 5 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
E	0	S	
F	6	B	S
C	8	H	B
D	8	G	H
A	9		

Step 5 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
F	6	S	
C	8	B	S
D	8	H	B
A	9	G	H
E	0		

Step 6 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
F	6	S	
C	8	B	S
D	8	H	B
A	9	G	H
		E	G

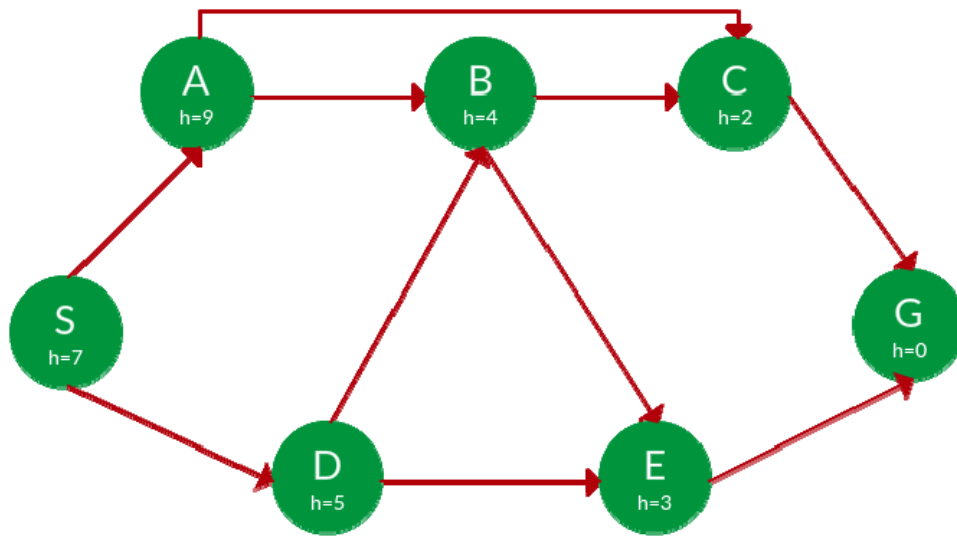
EXIT returning 'True' as the Goal node (E) is moved to the CLOSED list. Backtrack the closed list to get the optimal path (E → G → H → B → S)

**PATH: E – G – H – B – S**



## Example 2: Greedy BFS Search

**Question.** Find the path from S to G using greedy search. The heuristic values  $h$  of each node below the name of the node.



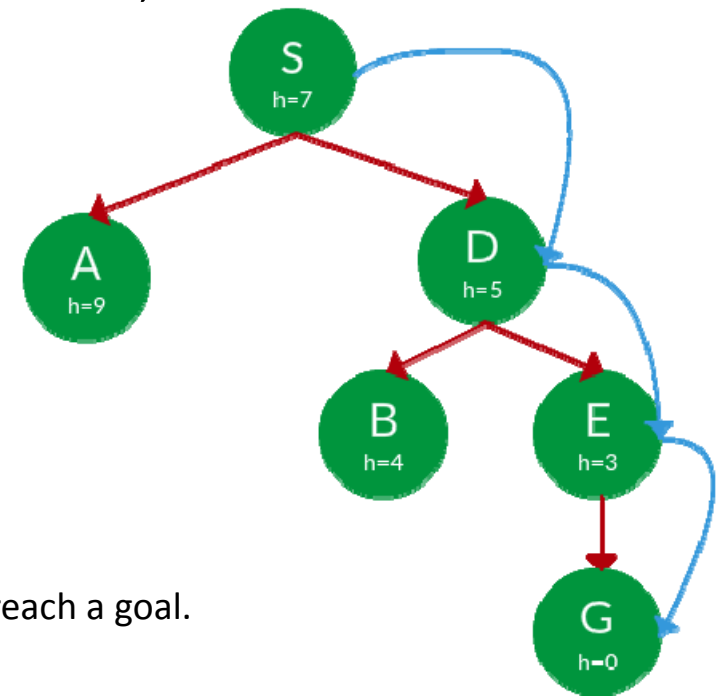
**Path:** S  $\rightarrow$  D  $\rightarrow$  E  $\rightarrow$  G

**Advantage:** Works well with informed search problems, with fewer steps to reach a goal.

**Disadvantage:** Can turn into unguided DFS in the worst case.

**Solution.** Starting from S, we can traverse to A( $h=9$ ) or D( $h=5$ ). We choose D, as it **has the lower heuristic cost**. Now from D, we can move to B( $h=4$ ) or E( $h=3$ ). We choose E with lower heuristic cost. Finally, from E, we go to G( $h=0$ ). This entire traversal is shown in the search tree below, in blue.

Selesaikan  
menggunakan  
model tabel  
seperti pada  
Example 1



# Greedy Best-first Search Algorithm

- **Advantages:**

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is **more efficient than Breadth FS and Depth FS** 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**.

# Greedy Best First Search algorithm

- **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.

## 2.2. A\* Tree (A\* Best First) Search Algorithm

<https://www.geeksforgeeks.org/search-algorithms-in-ai/>

- A\* Tree Search, or simply known as A\* Search, **combines the strengths of uniform-cost search (UCS) and greedy search.**
- In this search, **the heuristic is the summation of the cost in UCS, denoted by  $g(x)$ , and the cost in greedy search, denoted by  $h(x)$ .**
- The summed cost is denoted by  $f(x)$ .
- **$f(x)$**  in A\* search is denoted as

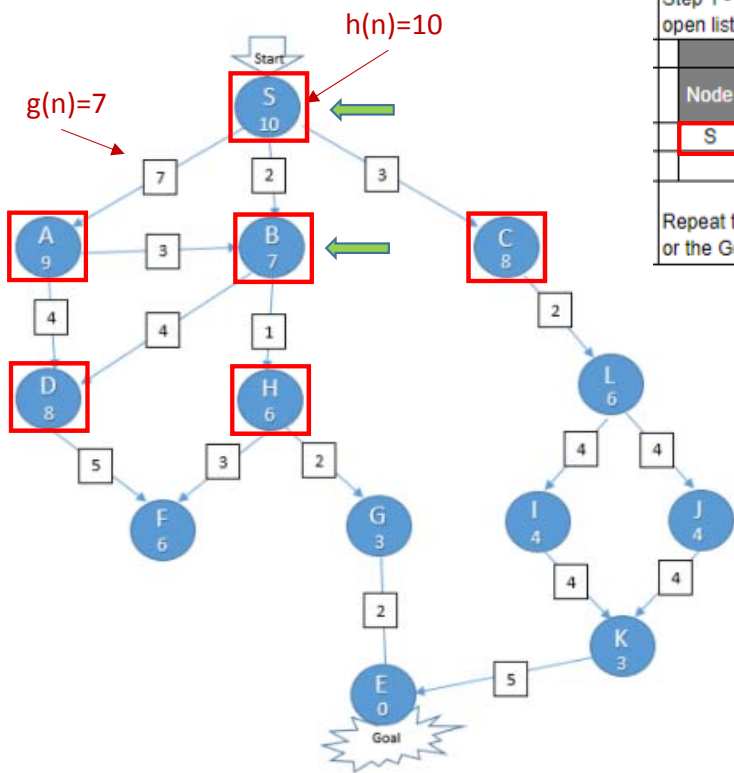
$$f(x) = g(x) + h(x)$$

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

- Here,  $g(x)$  is called the **backward cost**, and is the cumulative cost of a node from the root node.
- And,  $h(x)$  is called the **forward cost**, and is an estimation of the distance of the current node from the goal node.
- A\* search is optimal only when for all nodes, the forward cost for a node  $h(x)$  underestimates the actual cost  $h^*(x)$  to reach the goal.
- This property of A\* heuristic is called **admissibility**.
- **Admissibility:**  $0 \leq h(x) \leq h^*(x)$
- **Strategy:** Choose the node with lowest  $f(x)$  value.

Example 1: A\* Best-first Search  
Algorithm (1)

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



A\* BFS with evaluation function  $f(n) = h(n) + g(n)$

Step 1 - Start by adding the start node (S) to the open list with the path distance as 0

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
S	0	10	10		

Repeat the next steps until the OPEN List is empty or the Goal node is moved to the CLOSED list

Step 2 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
A	7	9	16	S	
B	2	7	9		
C	3	8	11		

Step 2 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
B	2	7	9	S	
C	3	8	11		
A	7	9	16		

RE-ORDER by  $f(n)$

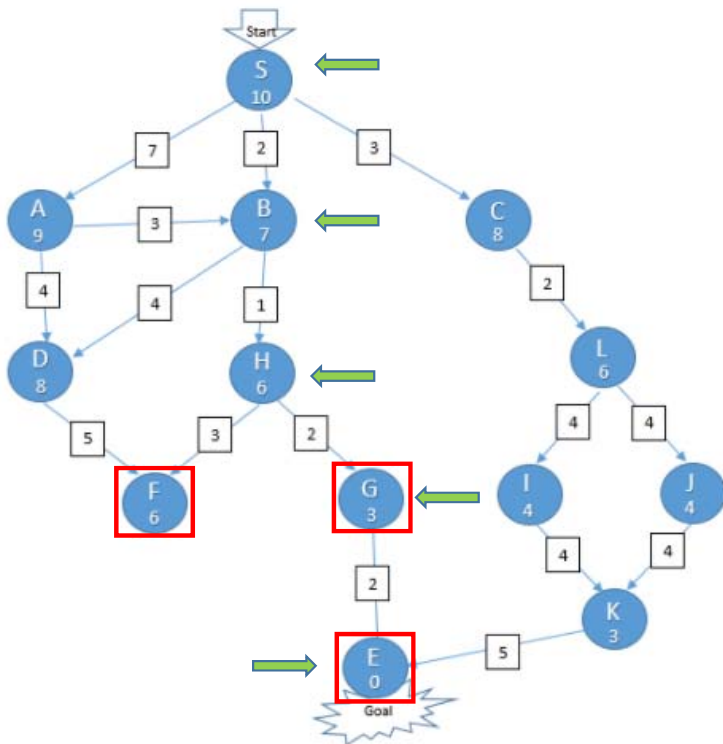
Step 3 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
A	7	9	16	B	S
D	6	8	14		
H	3	6	9		

Step 3 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
H	3	6	9	S	
C	3	8	11	B	S
D	6	8	14		
A	7	9	16		

RE-ORDER by  $f(n)$



Step 5 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
F	6	6	12	B	S
D	6	8	14	H	B
A	7	9	16	G	H
E	7	0	7		

Step 4 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
D	6	8	14	B	S
A	7	9	16	H	B
F	6	6	12		
G	5	3	8		

## Example 1: A\* Best-first Search Algorithm (2)

RE-ORDER by f(n)

Step 5 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
E	7	0	7	S	
C	3	8	11	B	S
F	6	6	12	H	B
D	6	8	14	G	H
A	7	9	16		

RE-ORDER by f(n)

Step 4 (b) - Re-order the list in ascending order of the combined heuristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
G	5	3	8	S	
C	3	8	11	B	S
F	6	6	12	H	B
D	6	8	14		
A	7	9	16		

Step 6 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
F	6	6	12	B	S
D	6	8	14	H	B
A	7	9	16	G	H
				E	G

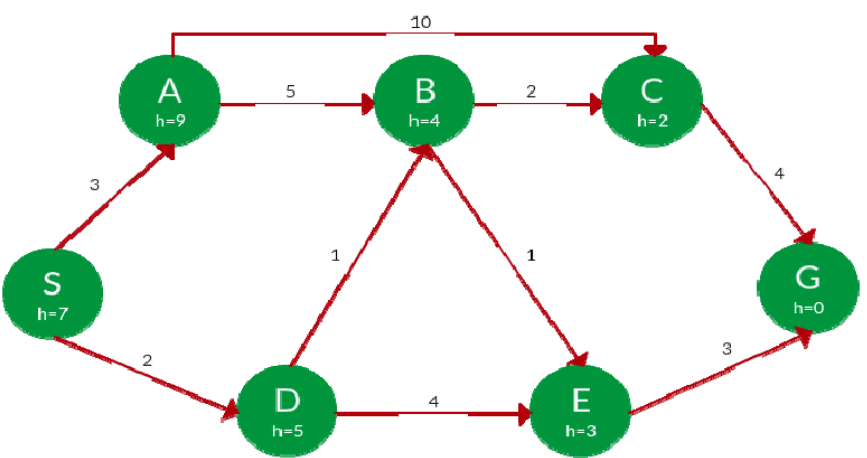
EXIT returning 'True' as the Goal node (E) is moved to the CLOSED list. Backtrack the closed list to get the optimal path (E → G → H → B → S)

PATH: E – G – H – B – S

# Example 2: A\* Tree (Best First) Search

$f(x) = g(x) + h(x)$   
 $g(x) = \text{backward}$   
 $h(x) = \text{forward}$

**Question.** Find the path to reach from S to G using A\* search.



**Solution.** Starting from S, the algorithm computes  $g(x) + h(x)$  for all nodes in the fringe at each step, **choosing the node with the lowest sum**. The entire working is shown in the table below.

Note that in the fourth set of iteration, we get two paths with equal summed cost  $f(x)$ , so we expand them both in the next set. The path with lower cost on further expansion is the chosen path.

PATH	$h(x)$	$g(x)$	$f(x)$
S	7	0	7
S -> A	9	3	12
S -> D ✓	5	2	7
S -> D -> B ✓	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C ✓	2	3 + 2 = 5	7
S -> D -> B -> E ✓	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G ✓	0	4 + 3 = 7	7

Path: S -> D -> B -> E -> G  
Cost: 7

Selesaikan menggunakan model tabel seperti pada Example 1



## Heuristic Searching 3.

### Branch and Bound Search Algorithm


<https://www.geeksforgeeks.org/job-assignment-problem-using-branch-and-bound/>

- Branch and bound is an algorithm design paradigm which is generally used for **solving combinatorial optimization problems**.
- The selection rule for the next node in BreadthFS and DepthFS is “blind”. i.e., the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.
- The search for an **optimal solution** can often be speeded by using an “**intelligent**” **ranking function**, also called an approximate cost function to avoid searching in sub-trees that do not contain an optimal solution.
- It is similar to BreadthFS-like search but with one major optimization.
- Instead of following FIFO order, we **choose a live node with least cost**.
- We may not get optimal solution by following node with least promising cost, but it will provide very good chance of getting the search to an answer node quickly.

# Job Assignment Problem using Branch And Bound

<https://www.geeksforgeeks.org/job-assignment-problem-using-branch-and-bound/>

- Let there be  $N$  workers and  $N$  jobs.
- Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment.
- **GOAL:** It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Worker A takes 8 units of time to finish job 4.

An example job assignment problem. Green values show optimal job assignment that is A-Job2, B-Job1, C-Job3 and D-Job4

## There are two approaches to calculate the cost function

- For each worker, we choose job with minimum cost from list of unassigned jobs (take minimum entry from each row).
- For each job, we choose a worker with lowest cost for that job from list of unassigned workers (take minimum entry from each column).

The first approach is followed:

For each worker, we choose job with minimum cost from list of unassigned jobs (take minimum entry from each row).

- Let's take below example and try to calculate promising cost when Job 2 is assigned to worker A.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

- Since Job 2 is assigned to worker A (marked in green), cost becomes 2 and Job 2 and worker A becomes unavailable (marked in red).

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

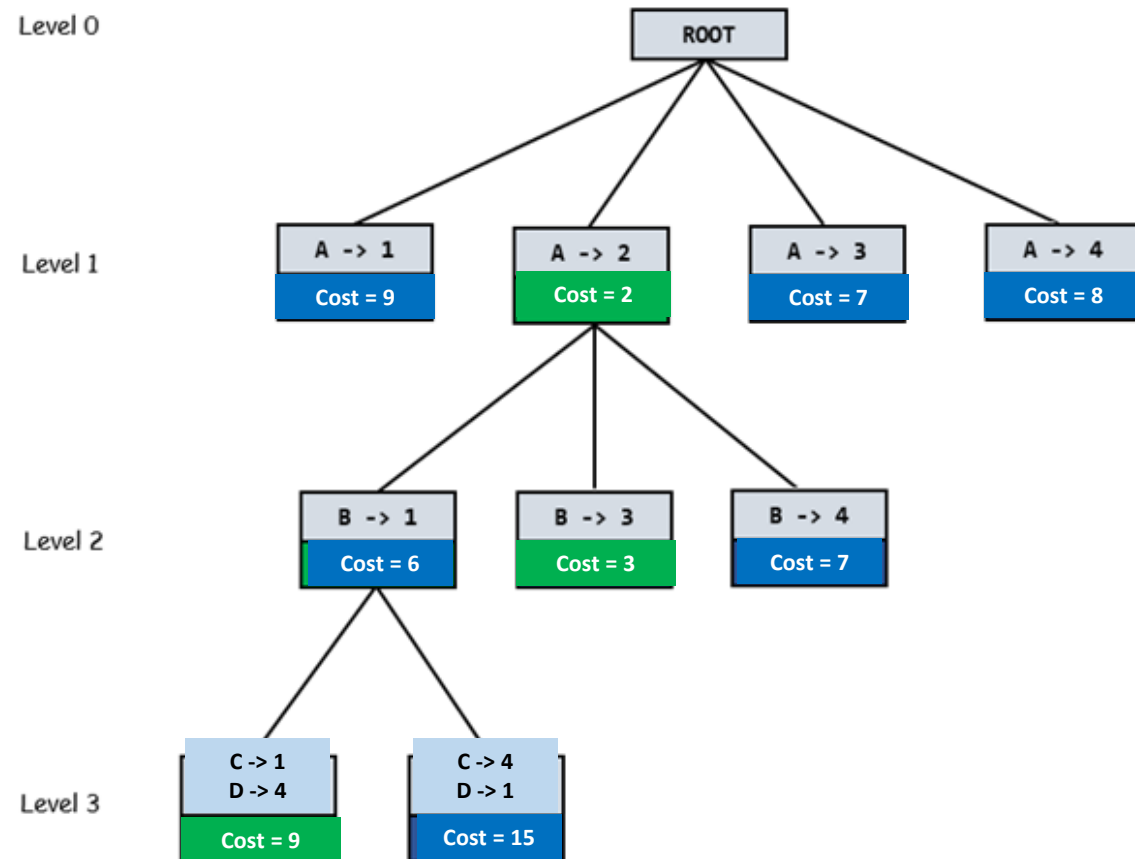
- Now we assign job 3 to worker B as it has minimum cost from list of unassigned jobs. Cost becomes  $2 + 3 = 5$  and Job 3 and worker B also becomes unavailable.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

- Finally, job 1 gets assigned to worker C as it has minimum cost among unassigned jobs and job 4 gets assigned to worker C as it is only Job left. Total cost becomes  $2 + 3 + 5 + 4 = 14$ .

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Below diagram shows complete search space diagram showing optimal solution path in green.



## Heuristic Searching 3:

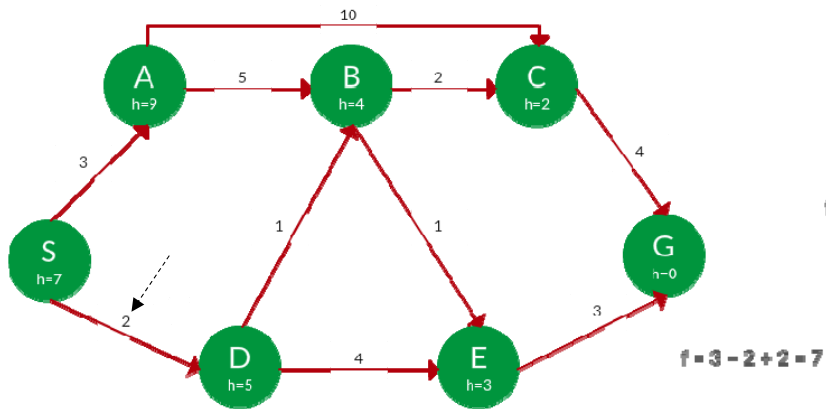
# A\* Graph Search Algorithm

<https://www.geeksforgeeks.org/search-algorithms-in-ai/>

- A\* tree search works well, except that **it takes time re-exploring the branches it has already explored.**
- In other words, if the same node has expanded twice in different branches of the search tree, A\* search might explore both of those branches, thus **wasting time**
- A\* Graph Search, **or simply Graph Search**, removes this limitation by adding this rule: **do not expand the same node more than once.**
- **Heuristic.** Graph search is optimal only when the forward cost between two successive nodes A and B, given by  $h(A) - h(B)$ , **is less than or equal to the backward cost between those two nodes  $g(A \rightarrow B)$ .**
- This property of graph search heuristic is called **consistency**.
- Consistency:  $h(A) - h(B) \leq g(A \rightarrow B)$

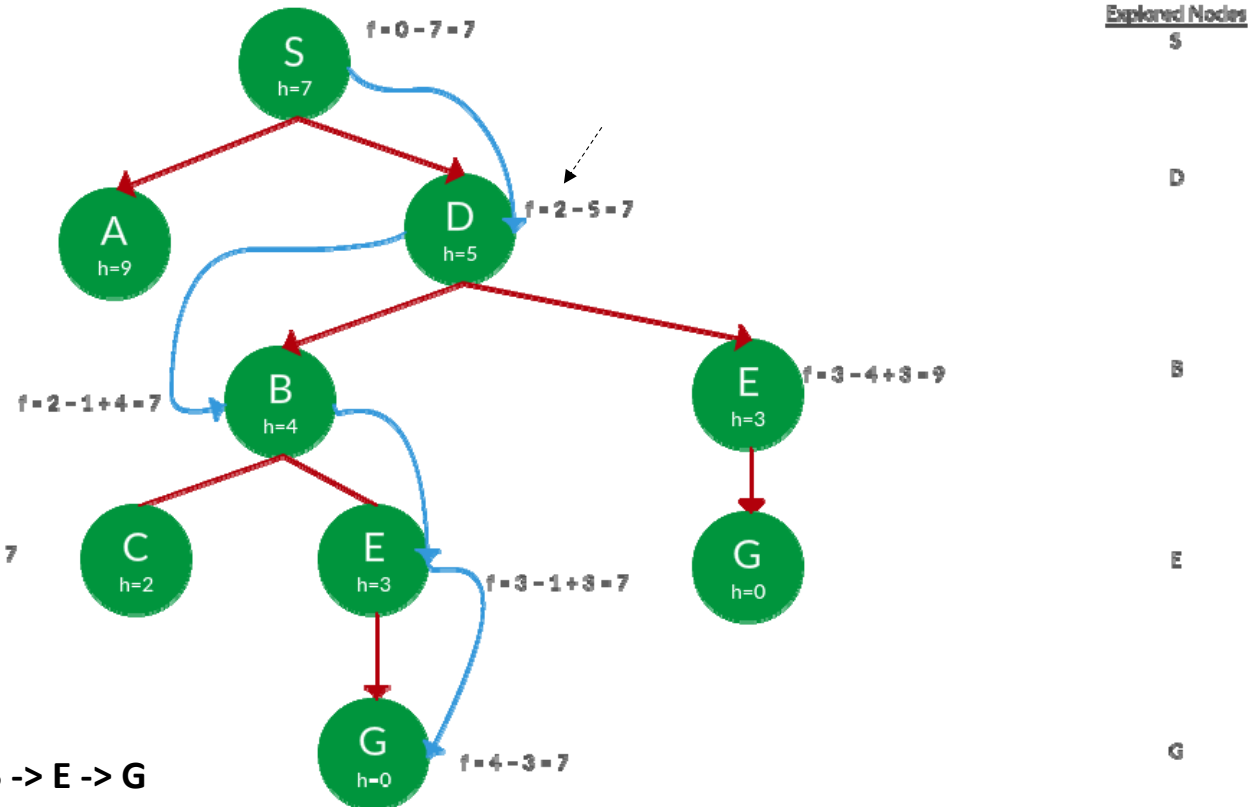
# Example of A\* Graph Search

**Question.** Use graph search to find path from S to G in the following graph.



Path: S -> D -> B -> E -> G  
Cost: 7

**Solution.** We solve this question pretty much the same way we solved last question, but in this case, we keep a track of nodes explored so that we don't re-explore them.



## Heuristic Searching 4:

### Hill Climbing

- Hill climbing search is a local search problem.
- *The purpose of the hill climbing search is to climb a hill and reach the topmost peak/point of that hill.*
- It is based on the **heuristic search technique** where the person who is climbing up on the hill estimates the direction which will lead him to the highest peak.



# Hill Climbing

- Metode ini hampir sama dengan metode **Generate and Test**, hanya saja proses pengujian dilakukan dengan menggunakan fungsi heuristic.
- Pembangkitan keadaan berikutnya tergantung pada feedback dari prosedur pengetesan.
- Tes yang berupa fungsi heuristic ini akan menunjukkan seberapa baiknya nilai terkaan yang diambil terhadap keadaan-keadaan lainnya yang mungkin.

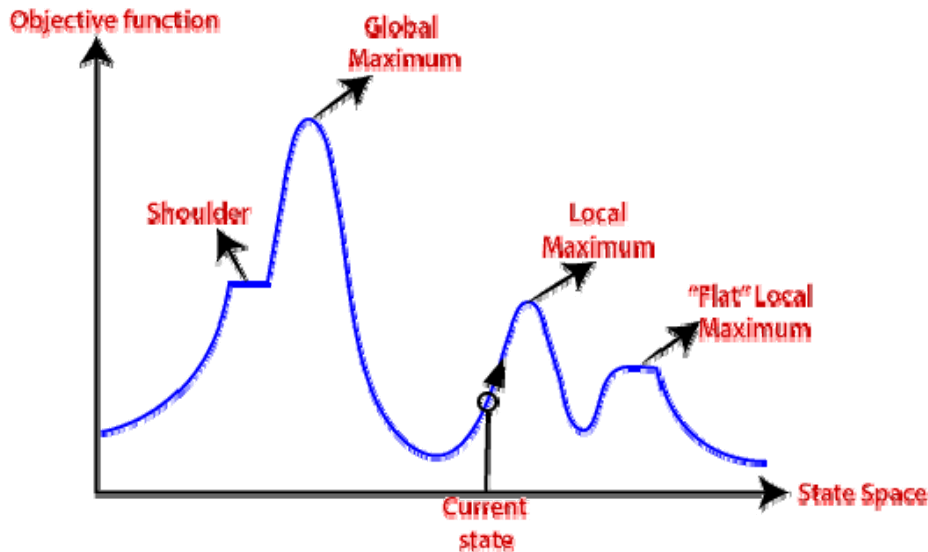
# State-space Landscape of Hill climbing algorithm

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 maximize).

**X-axis** : denotes the state space, i.e., 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).**



A one-dimensional state-space landscape in which elevation corresponds to the objective function

## The topographical regions.

The concept of hill climbing algorithm, consider the below landscape representing the **goal state/peak** and the **current state** of the climber

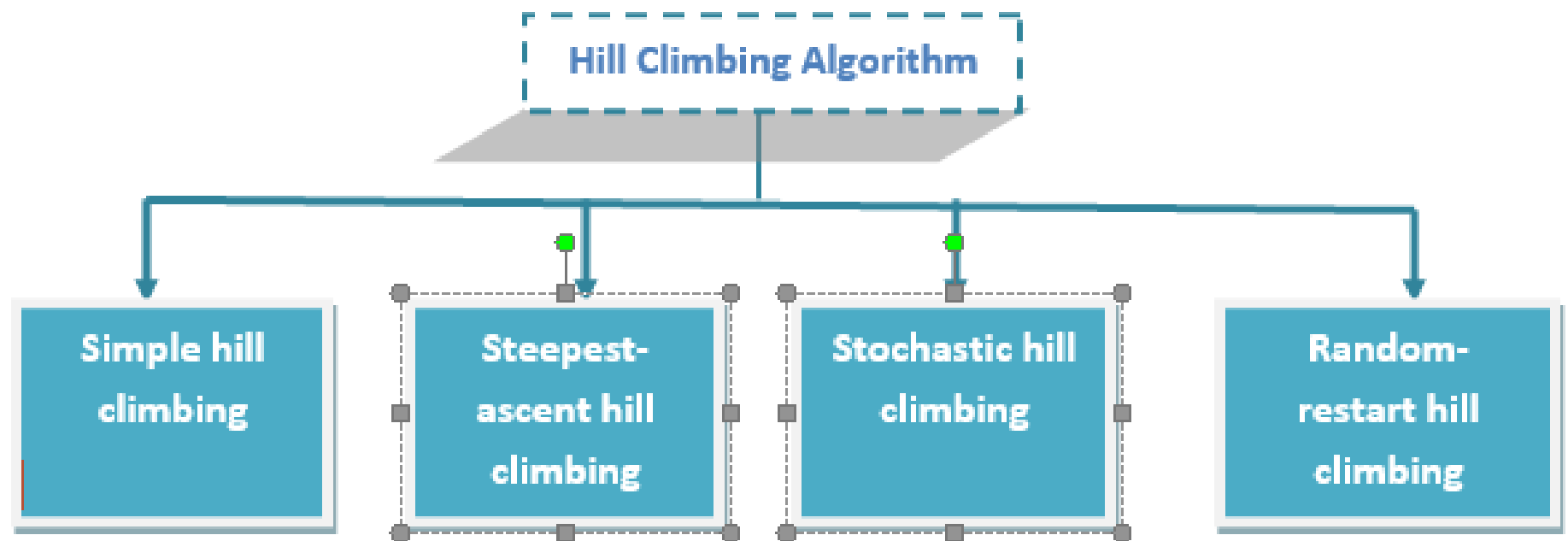
- **Global Maximum:** It is the highest point on the hill, which is the goal state.
- **Local Maximum:** It is the peak higher than all other peaks but lower than the global maximum.
- **Flat local maximum:** It is the flat area over the hill where it has no uphill or downhill. It is a saturated point of the hill.
- **Shoulder:** It is also a flat area where the summit is possible.
- **Current state:** It is the current position of the person.

# The topographical regions

The concept of hill climbing algorithm, consider the below landscape representing the **goal state/peak** and the **current state** of the climber

- **Global Maximum:** It is the highest point on the hill, which is the goal state.
- **Local Maximum:** It is the peak higher than all other peaks but lower than the global maximum.
- **Flat local maximum:** It is the flat area over the hill where it has no uphill or downhill. It is a saturated point of the hill.
- **Shoulder:** It is also a flat area where the summit is possible.
- **Current state:** It is the current position of the person.

# Types of Hill climbing search algorithm



# Types of Hill Climbing

## 1. Simple Hill Climbing

- Simple hill climbing is **the simplest way** to implement a hill-climbing algorithm.
- It **only evaluates the neighbour node state at a time** and **selects the first one which optimizes current cost and set it as a current state**.
- It **only checks it's one successor state**, and if it finds better than the current state, then move else be in the same state.

## 2. Steepest-Ascent hill climbing

- The **steepest-Ascent algorithm** is a **variation of the simple hill-climbing** algorithm.
- This algorithm **examines all the neighbouring nodes** of the current state and **selects one neighbour node which is closest to the goal state**.
- This algorithm **consumes more time** as it searches for multiple neighbours.

## 3. Stochastic hill climbing

- Stochastic hill climbing **does not examine for all its neighbours** before moving.
- Rather, this search algorithm **selects one neighbour node at random** and **evaluate it as a current state** or examine another state.

# Simple hill climbing search

- Simple hill climbing is the simplest technique to climb a hill.
- The task is to **reach the highest peak of the mountain.**
- The movement of the climber depends on his move/steps.
- If he finds his next step better than the previous one, he continues to move else remain in the same state.
- This search focus only on his previous and next step.

# Steepest-Ascent hill climbing

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make the current state as your initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
  - Let **S** be a state such that any successor of the current state will be better than it.
  - For each operator that applies to the current state;
    - Apply the new operator and generate a new state.
    - Evaluate the new state.
    - If it is goal state, then return it and quit, else compare it to the **S**.
    - If it is better than **S**, then set new state as **S**.
    - If the **S** is better than the current state, then set the current state to **S**.
- **Step 5:** Exit.

## Simple hill climbing algorithm

{ It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node. }

- **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 step 2.
- **Step 5:** Exit.

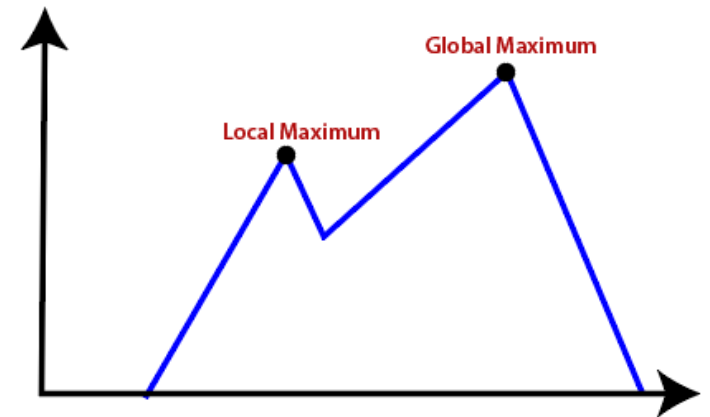


# Limitations of Hill climbing algorithm

- Hill climbing algorithm is a **fast and furious approach**.
- It **finds the solution state rapidly** because it is quite easy to improve a bad state.
- But, there are some limitations of this search:
  - **Local Maxima**
  - **Plateau**
  - **Ridges**

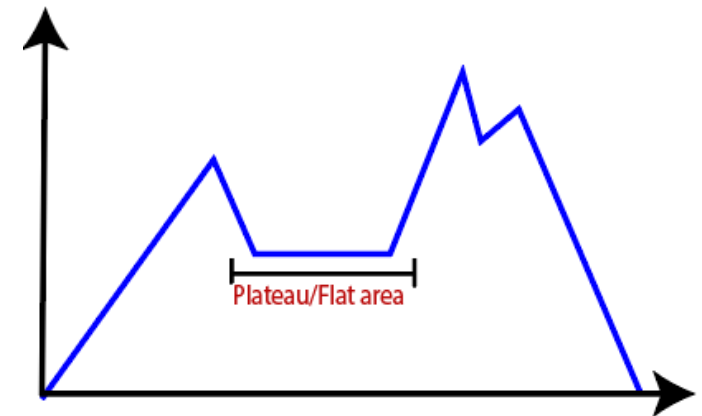
# Local Maxima

- It is that peak of the mountain which is highest than all its neighboring states but lower than the global maxima.
- It is not the goal peak because there is another peak higher than it.
- **Solution:**
  - **Backtracking** technique can be a solution of the local maximum in state space landscape.
  - **Create a list of the promising** path so that the algorithm can backtrack the search space and explore other paths as well.



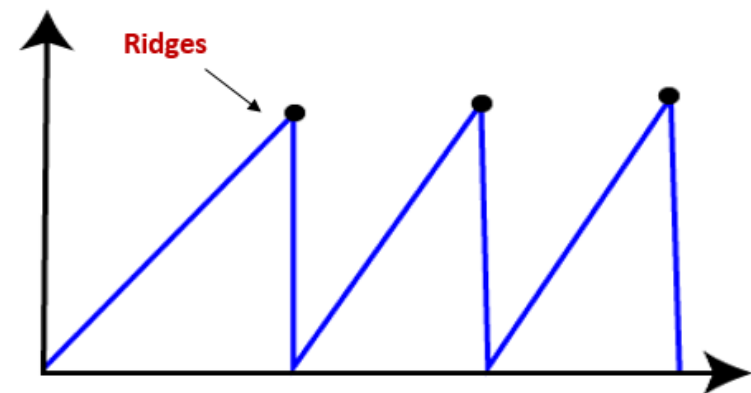
# Plateau

- It is a flat surface area where no uphill exists.
- It becomes difficult for the climber to decide that in which direction he should move to reach the goal point.
- Sometimes, the person gets lost in the flat area.
- **Solution:**
  - Take **big steps** or very little steps while searching, to solve the problem.
  - **Randomly select a state which is far away from the current state** so it is possible that the algorithm could find non-plateau region.



# Ridges

- It is a challenging problem where the person finds two or more local maxima of the same height commonly.
- It becomes difficult for the person to navigate the right point and stuck to that point itself.
- **Solution:**
  - by using of bidirectional search, or
  - by moving in different directions



# Referensi

- [https://users.cs.cf.ac.uk/Dave.Marshall/AI2/node22.html#figdep\\_tree](https://users.cs.cf.ac.uk/Dave.Marshall/AI2/node22.html#figdep_tree)
- <https://www.geeksforgeeks.org/search-algorithms-in-ai/>
- <https://www.javatpoint.com/ai-uninformed-search-algorithms>
- <https://syifamss.wordpress.com/2017/12/08/metode-pencarian-buta-blind-search-metode-pencarian-heuristik/>
- <http://alfanfikri27.blogspot.com/2017/12/metode-blind-search-heuristik.html>
- <https://fitriahadiarief.wordpress.com/2017/12/09/metode-pencarian-buta-blind-search-method-dan-metode-pencarian-heuristik/>