

Course Code: CSE3002
Course Name: Artificial Intelligence



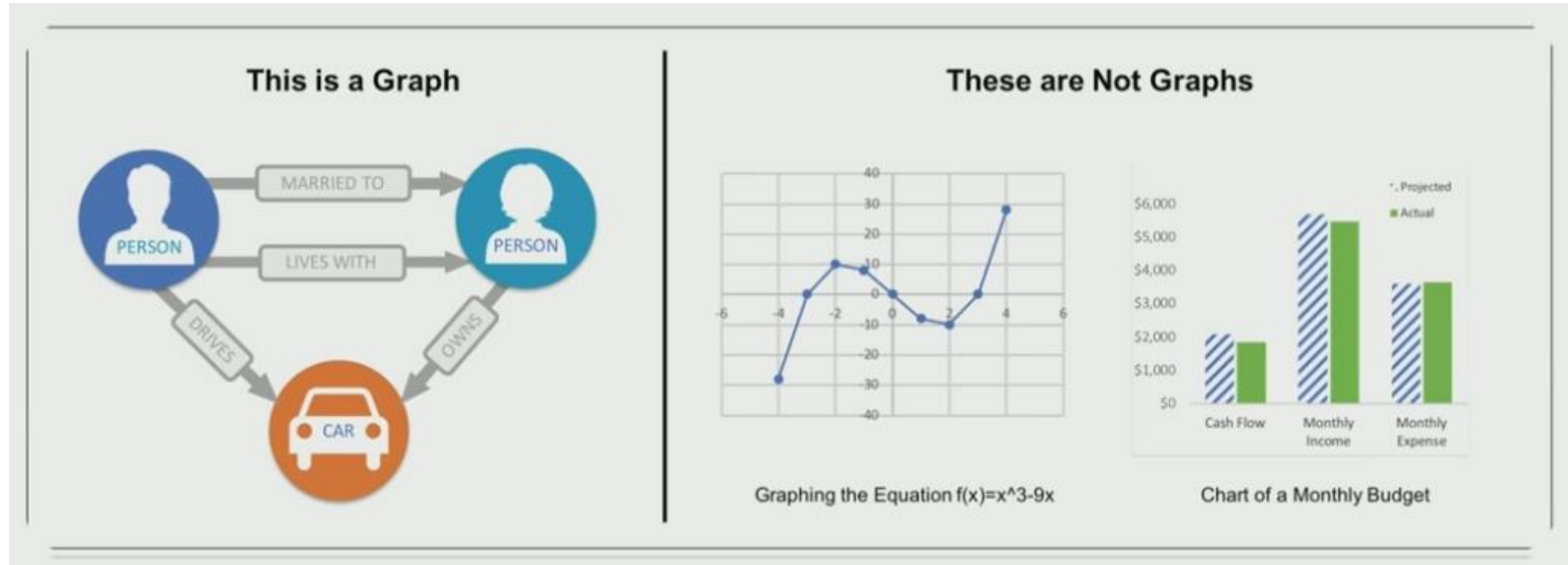
Faculty Name:
D Ramkumar
AP/SCOPE
VIT-AP

Module-2

Module 2:	Problem Solving methods	7 Hours
------------------	--------------------------------	----------------

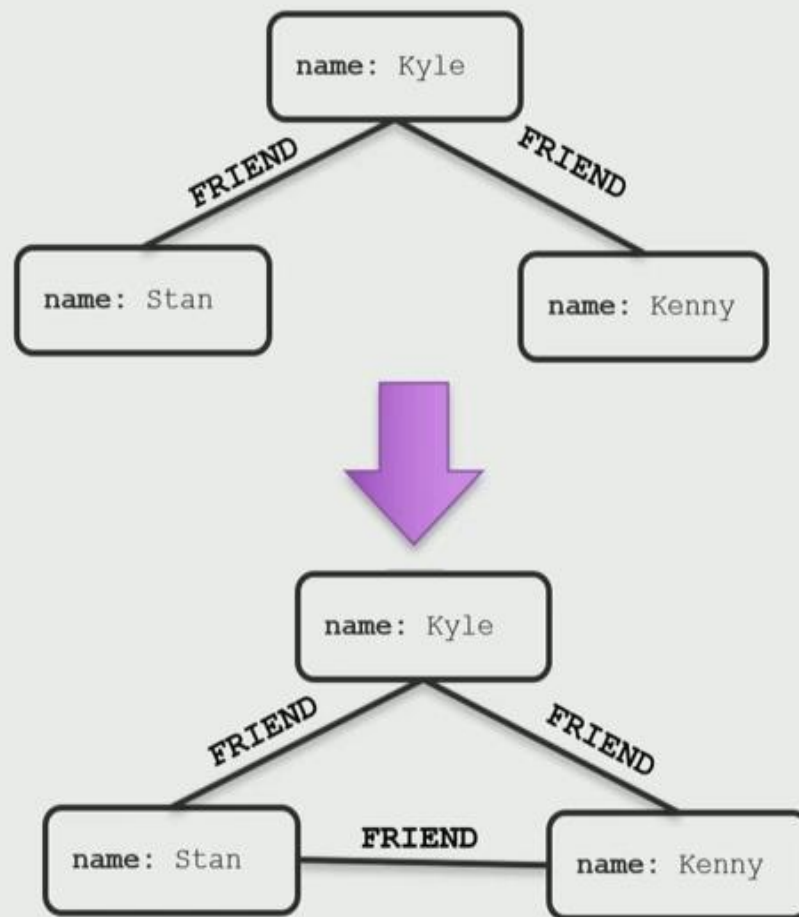
Problem graphs, Matching, Indexing and Heuristic functions -Hill Climbing-Depth first and Breath first, Constraints satisfaction - Related algorithms, Measure of performance and analysis of search algorithms.

What Is a Graph?

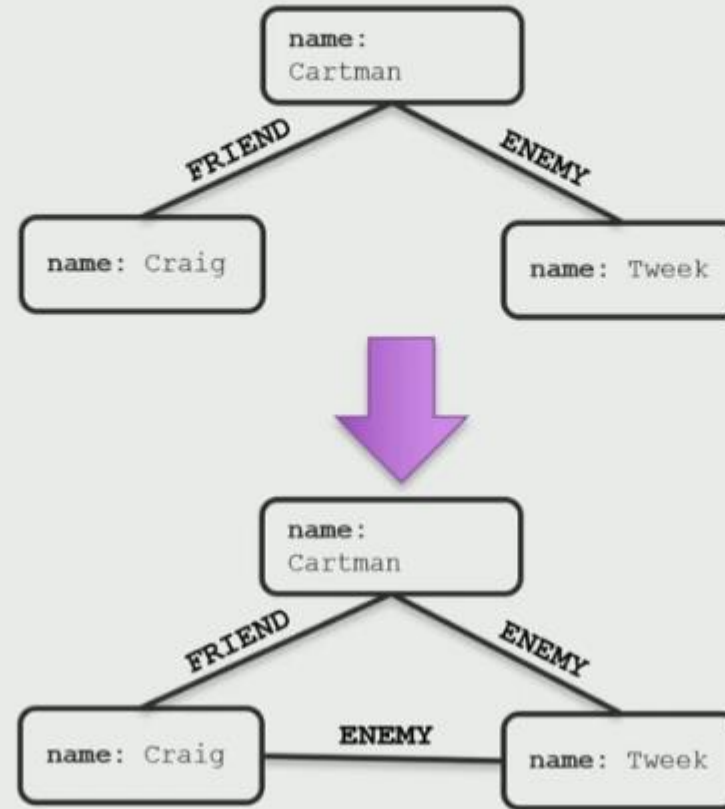


The first skill that we learn in graphs is this notion of triadic closure, which basically means “make a triangle.”

Triadic Closure

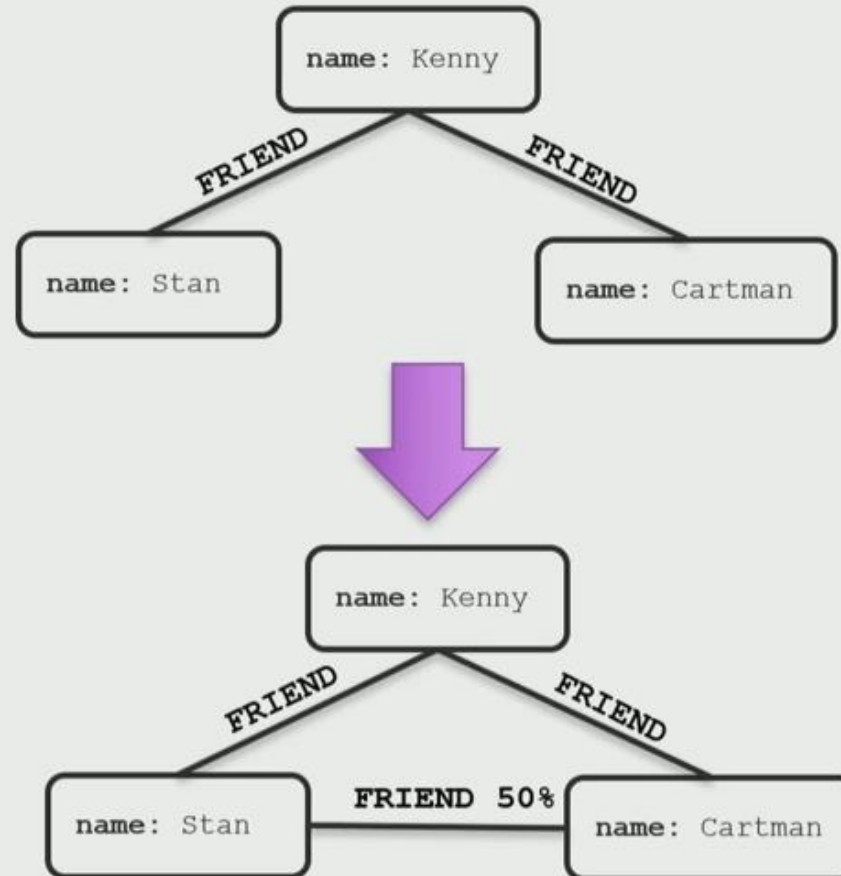


Structural Balance

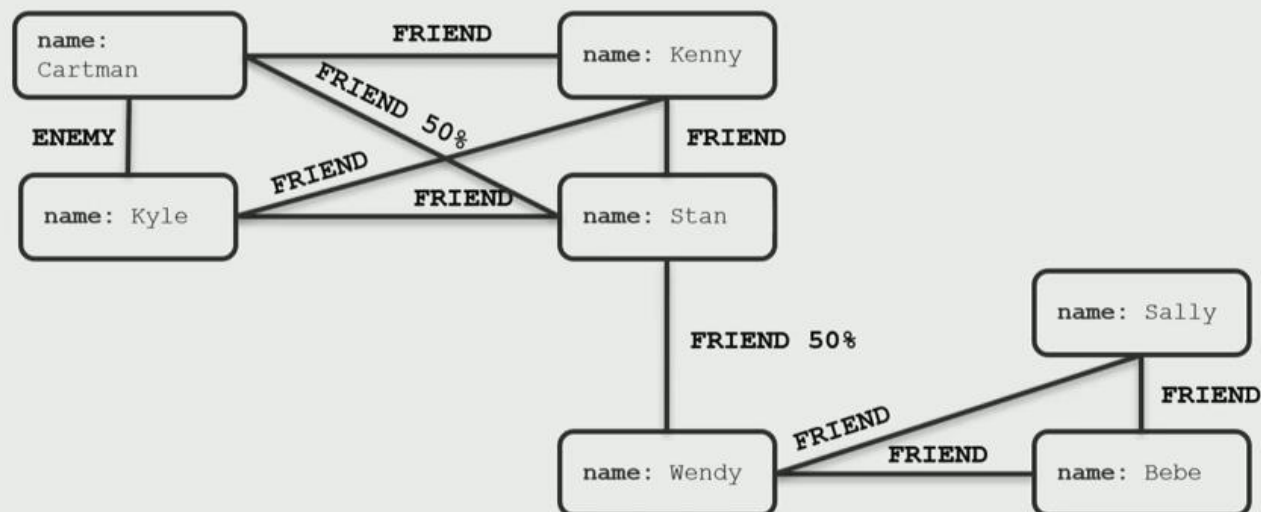


Triadic Closure

(weak relationship)



Local Bridges



What is Heuristic Search – Techniques & Hill Climbing in AI

- What is a Heuristic Search?

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed.

- A Heuristic (or a heuristic function) takes a look at search algorithms. At each branching step, it evaluates the available information and makes a decision on which branch to follow.

What is Heuristic Search – Techniques & Hill Climbing in AI

- What is a Heuristic Search?
- It does so by ranking alternatives. The Heuristic is any device that is often effective but will **not guarantee work in every case**.
- So *why do we need heuristics*? One reason is to produce, in a reasonable amount of time, a solution that is good enough for the problem in question. It doesn't have to be the best- an approximate solution **will do since this is fast enough**.
- We use this in AI because we can put it to use in situations where we can't find known algorithms.

Heuristic Search Techniques in Artificial Intelligence

Briefly, we can taxonomize such techniques of Heuristic into two categories:

a. Direct Heuristic Search Techniques in AI

- Other names for these are Blind Search, Uninformed Search, and Blind Control Strategy. These aren't always possible since they demand much time or memory.
- They search the entire state space for a solution and use an arbitrary ordering of operations. Examples of these are Breadth First Search (BFS) and Depth First Search (DFS).

Heuristic Search Techniques in Artificial Intelligence

Briefly, we can taxonomize such techniques of Heuristic into two categories:

b. Weak Heuristic Search Techniques in AI

- Other names for these are Informed Search, Heuristic Search, and Heuristic Control Strategy. These are effective if applied correctly to the right types of tasks and usually demand domain-specific information.
 - *Best-First Search*
 - *A* Search*
 - *Bidirectional Search*
 - *Tabu Search*
 - *Beam Search*
 - *Simulated Annealing*
 - *Hill Climbing*
 - *Constraint Satisfaction Problems*



Hill Climbing in AI

01

Features

Types

02

03

Problems

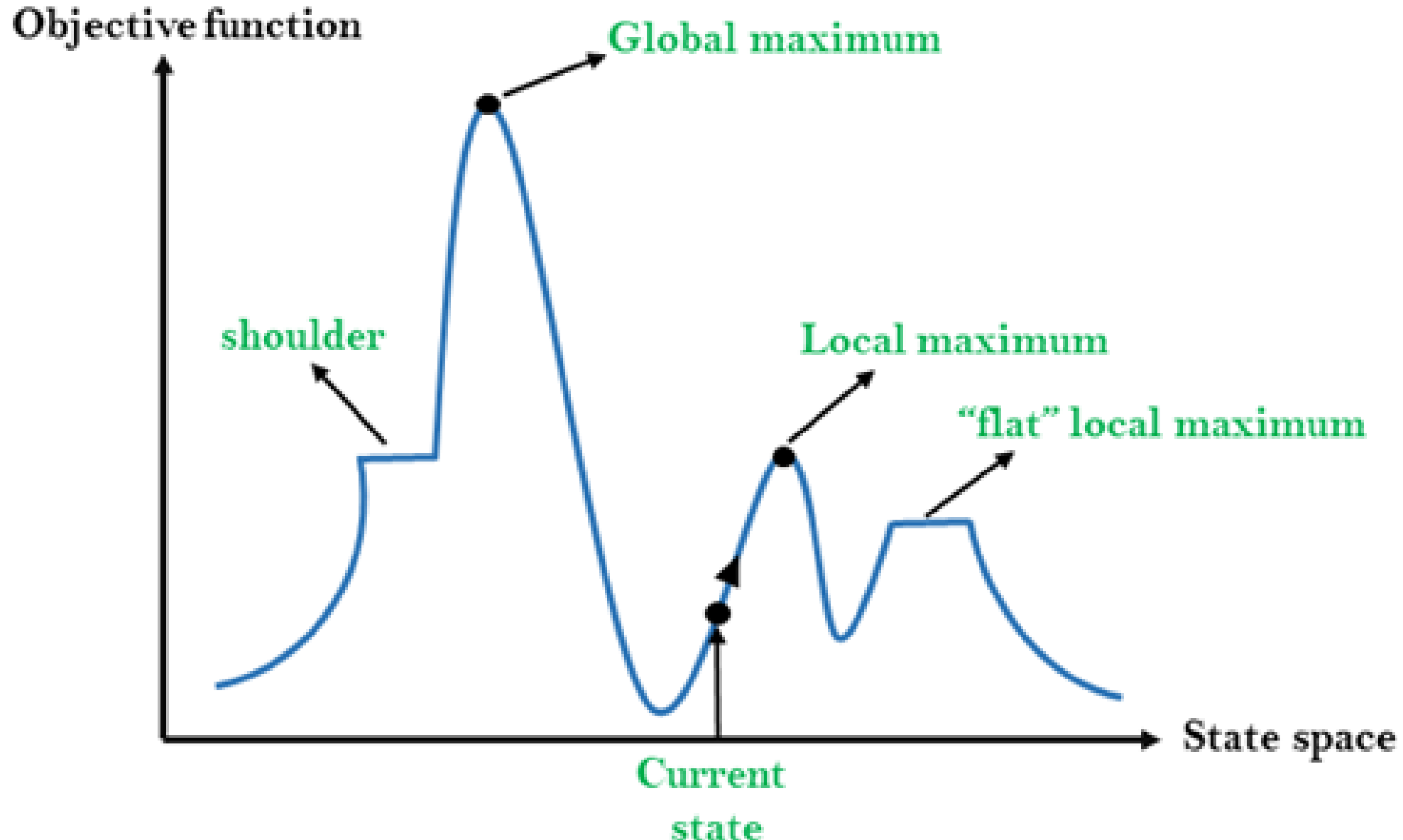
Hill Climbing in Artificial Intelligence

This is a heuristic for optimizing problems mathematically. We need to choose values from the input to maximize or minimize a real function.

Hill Climbing Search Algorithm

- Algorithm generally moves up in the direction of increasing value that is uphill
- It breaks its "moving up loop" when it reaches a "peak" where no neighbour has a higher value
- It does not maintain a search tree
- It only looks out for immediate neighbour of current state.

State-space Diagram for Hill Climbing



Different regions in the state space landscape



- **Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.
- **Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.
- **Current state:** It is a state in a landscape diagram where an agent is currently present.
- **Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.
- **Shoulder:** It is a plateau region which has an uphill edge.

HILL (Limbing Search Algorithm

- Evaluate initial state, if its goal state quit, otherwise make current state as initial state.
- Select a operator that could generate a new state
- Evaluate new state if closer to goal make it current state, if not better ignore this state
- If current is goal state then quit otherwise repeat

Features of Hill Climbing in AI

Let's discuss some of the features of this algorithm (Hill Climbing):

- It is a variant of the generate-and-test algorithm
- It makes use of the greedy approach

This means it keeps generating possible solutions until it finds the expected solution, and moves only in the direction which optimizes the cost function for it.

Types of Hill Climbing

Stochastic Hill Climbing

03

Steepest Ascent Hill Climbing

02

Simple Hill Climbing

01

- **Simple Hill Climbing-** This examines one neighboring node at a time and selects the first one that optimizes the current cost to be the next node.
- **Steepest Ascent Hill Climbing-** This examines all neighboring nodes and selects the one closest to the solution state.
- **Stochastic Hill Climbing-** This selects a neighboring node at random and decides whether to move to it or examine another.

c. Problems with Hill Climbing in AI

We usually run into one of three issues-

- **Local Maximum-** All neighboring states have values worse than the current. The greedy approach means we won't be moving to a worse state. This terminates the process even though there may have been a better solution. As a workaround, we use backtracking.
- **Plateau-** All neighbors to it have the same value. This makes it impossible to choose a direction. To avoid this, we randomly make a big jump.
- **Ridge-** At a ridge, movement in all possible directions is downward. This makes it look like a peak and terminates the process. To avoid this, we may use two or more rules before testing.

DFS (Depth First Search) algorithm

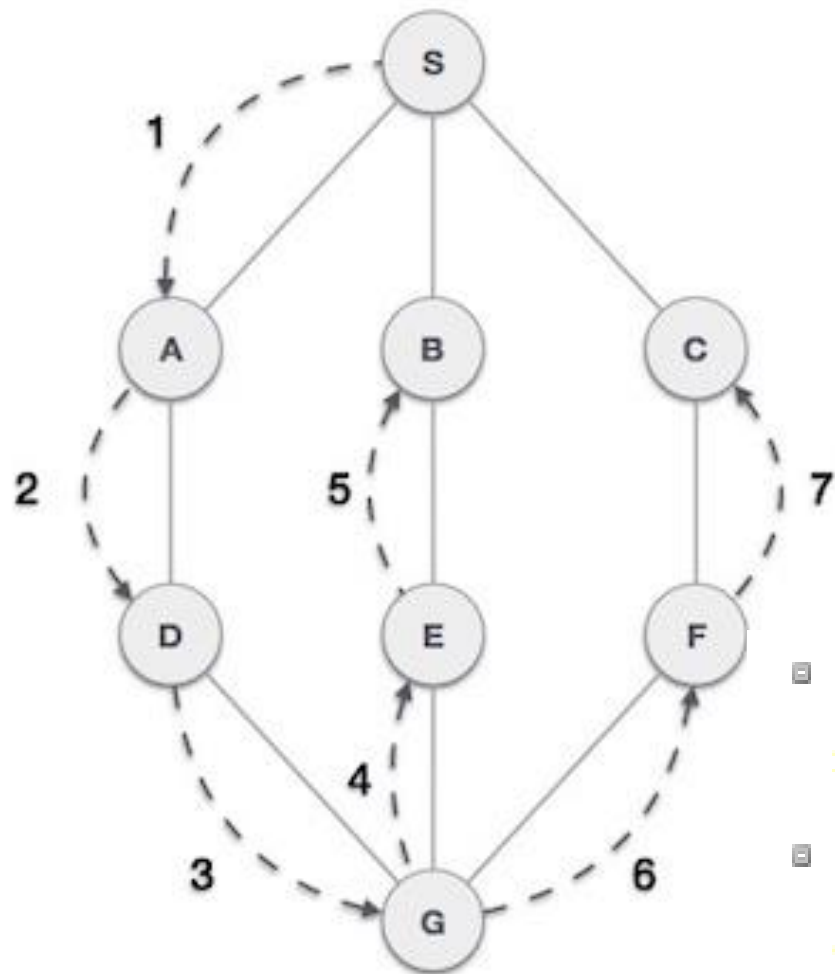
In this article, we will discuss the DFS algorithm in the data structure. It is a **recursive algorithm to search all the vertices** of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, **stack data structure can be used to implement** the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

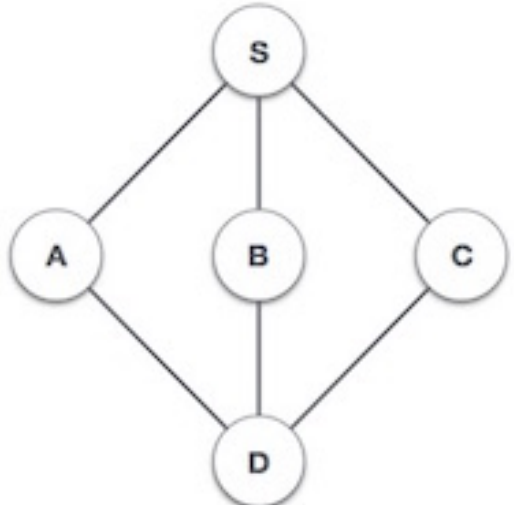

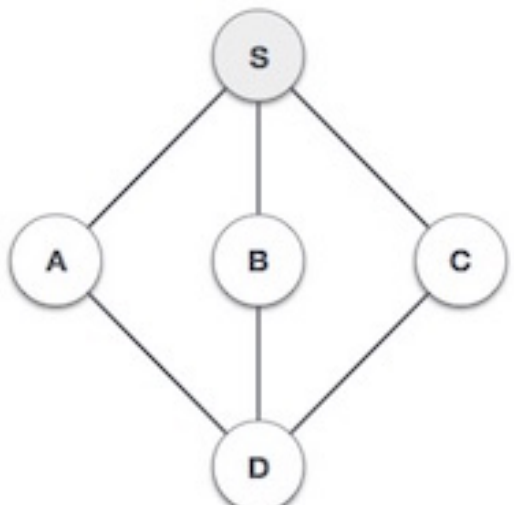
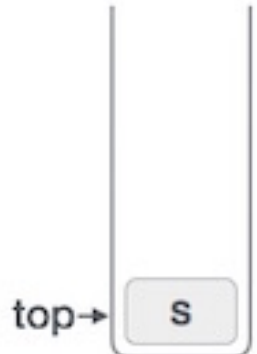
DFS (Depth First Search) algorithm

The step by step process to implement the DFS traversal is given as follows -

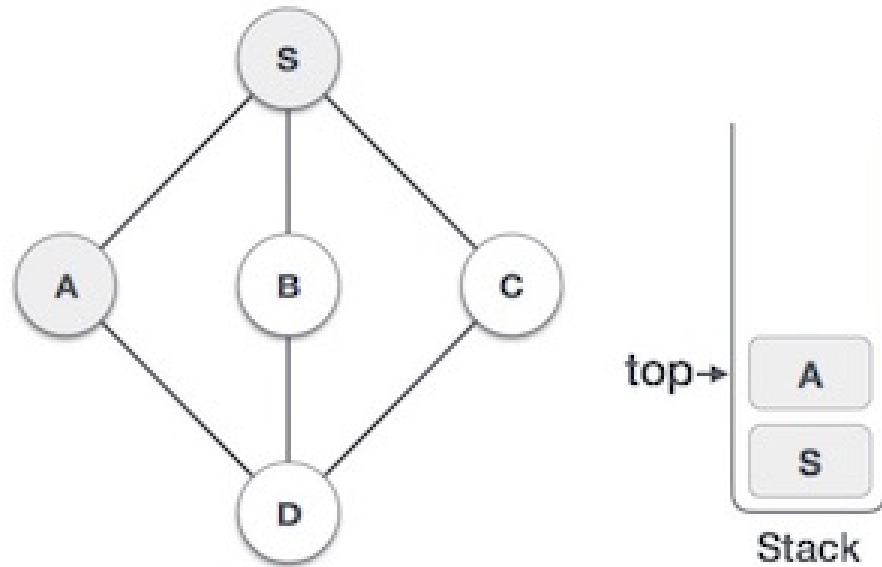
- First, create a stack with the total number of vertices in the graph.
- Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
- After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
- Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
- If no vertex is left, go back and pop a vertex from the stack.
- Repeat steps 2, 3, and 4 until the stack is empty.



- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- **Rule 2** – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- **Rule 3** – Repeat Rule 1 and Rule 2 until the stack is empty.

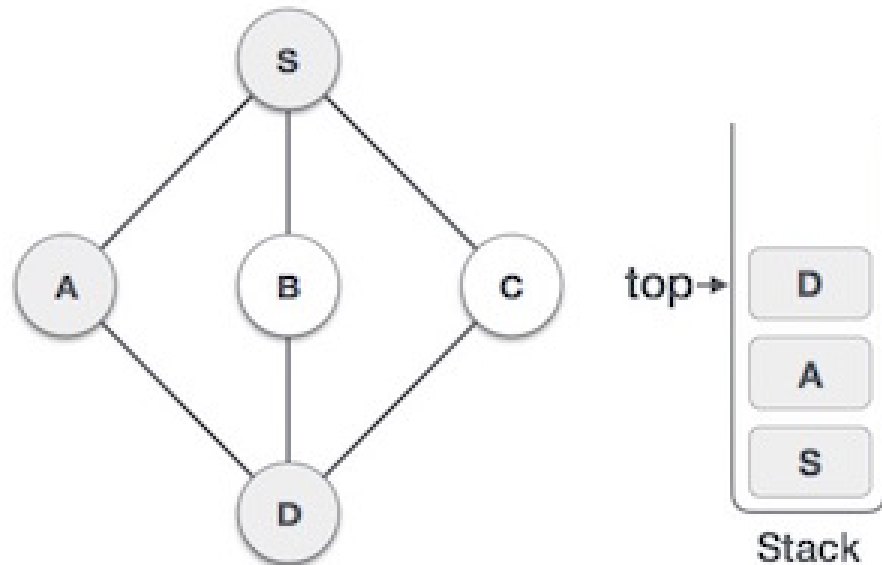
Step	Traversal	Description
1	  Stack	Initialize the stack.
2	  Stack	Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S . We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.

3

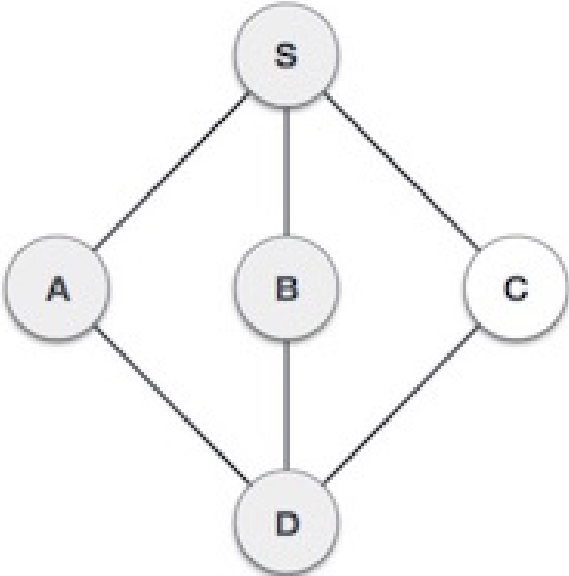
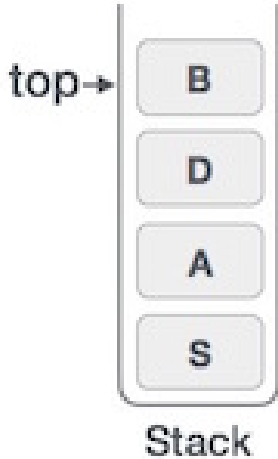
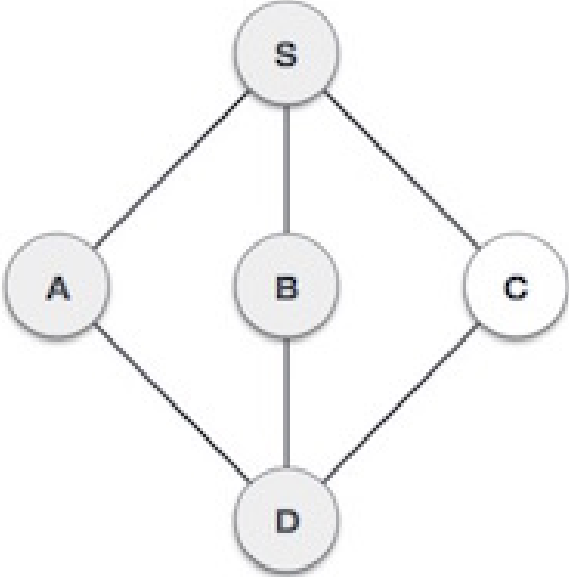
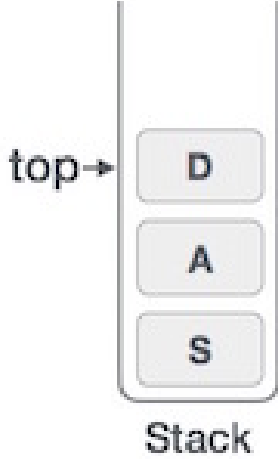


Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.

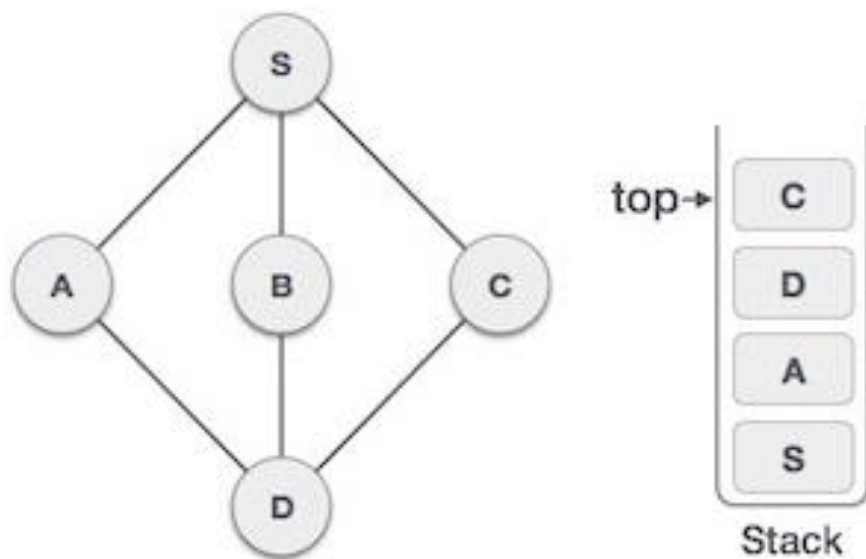
4



Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.

5	  <p>Stack</p>	<p>We choose B, mark it as visited and put onto the stack. Here B does not have any unvisited adjacent node. So, we pop B from the stack.</p>
6	  <p>Stack</p>	<p>We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack.</p>

7



Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

Constraints satisfaction Problem

Related algorithms, Measure of performance and analysis of search algorithms

A* (Star) Search in Artificial Intelligence

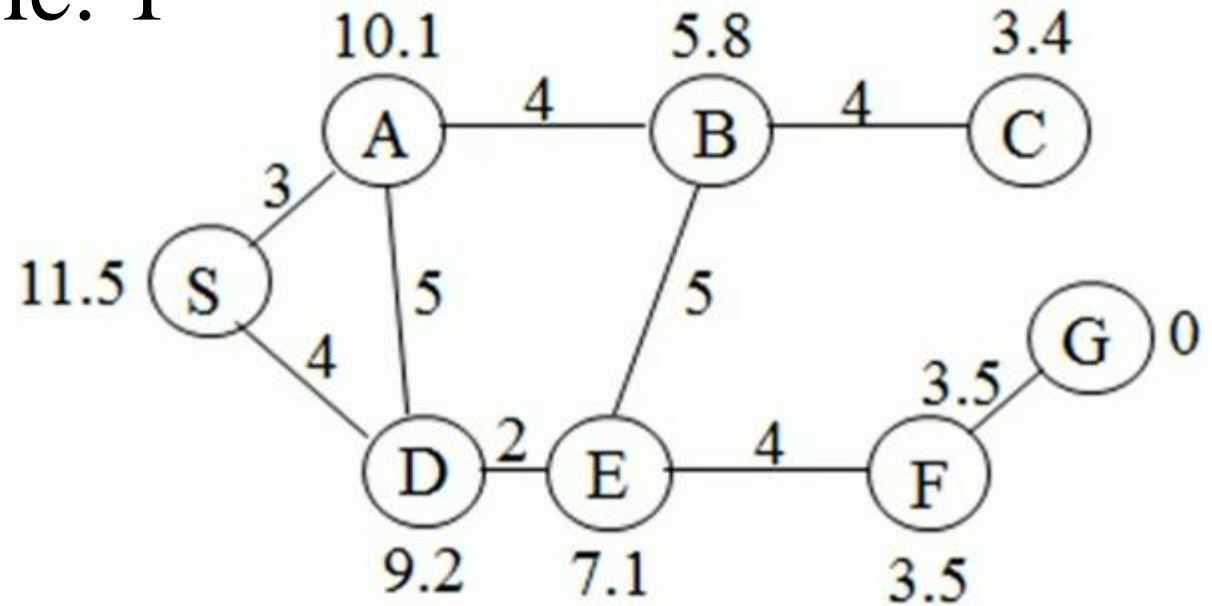
- Algorithm A* (Hart et al., 1968):

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

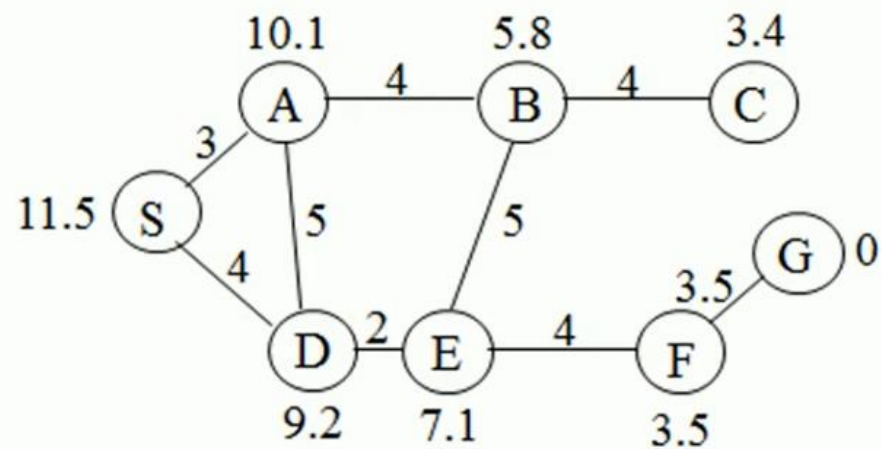
- $h(n)$ = cost of the cheapest path from node n to a goal state.
= The heuristic approximation of the value of the node
- $g(n)$ = cost of the cheapest path from the initial state to node n .
= shows the shortest path's value from the starting node to node n

A* (Star) Search in Artificial Intelligence

Example: 1



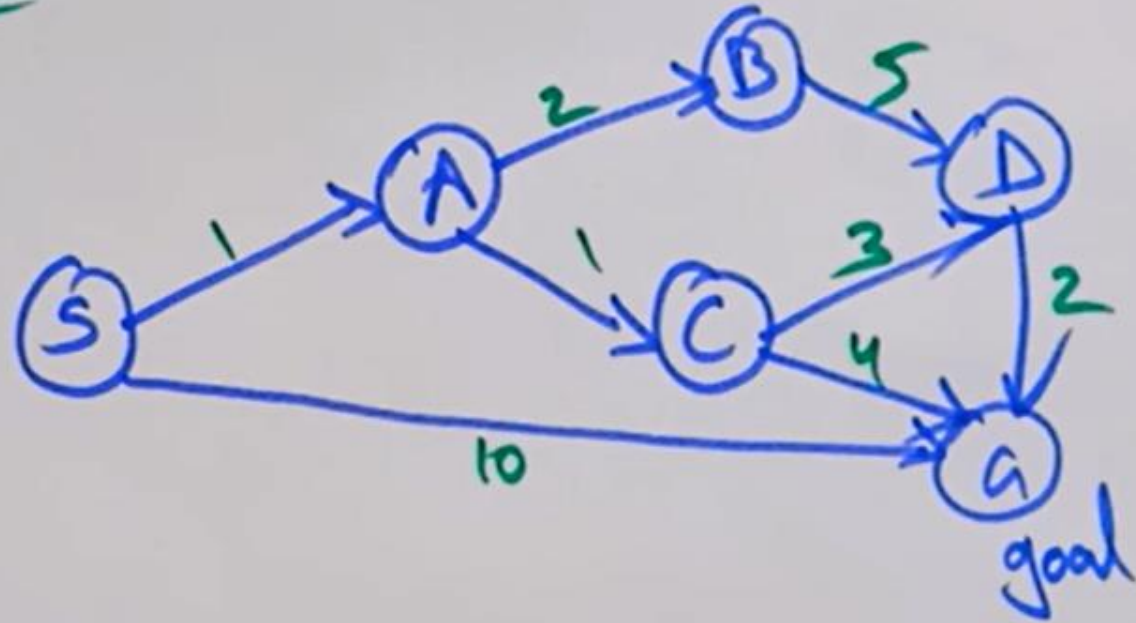
A* (Star) Search in Artificial Intelligence



A* Search Algorithm 1

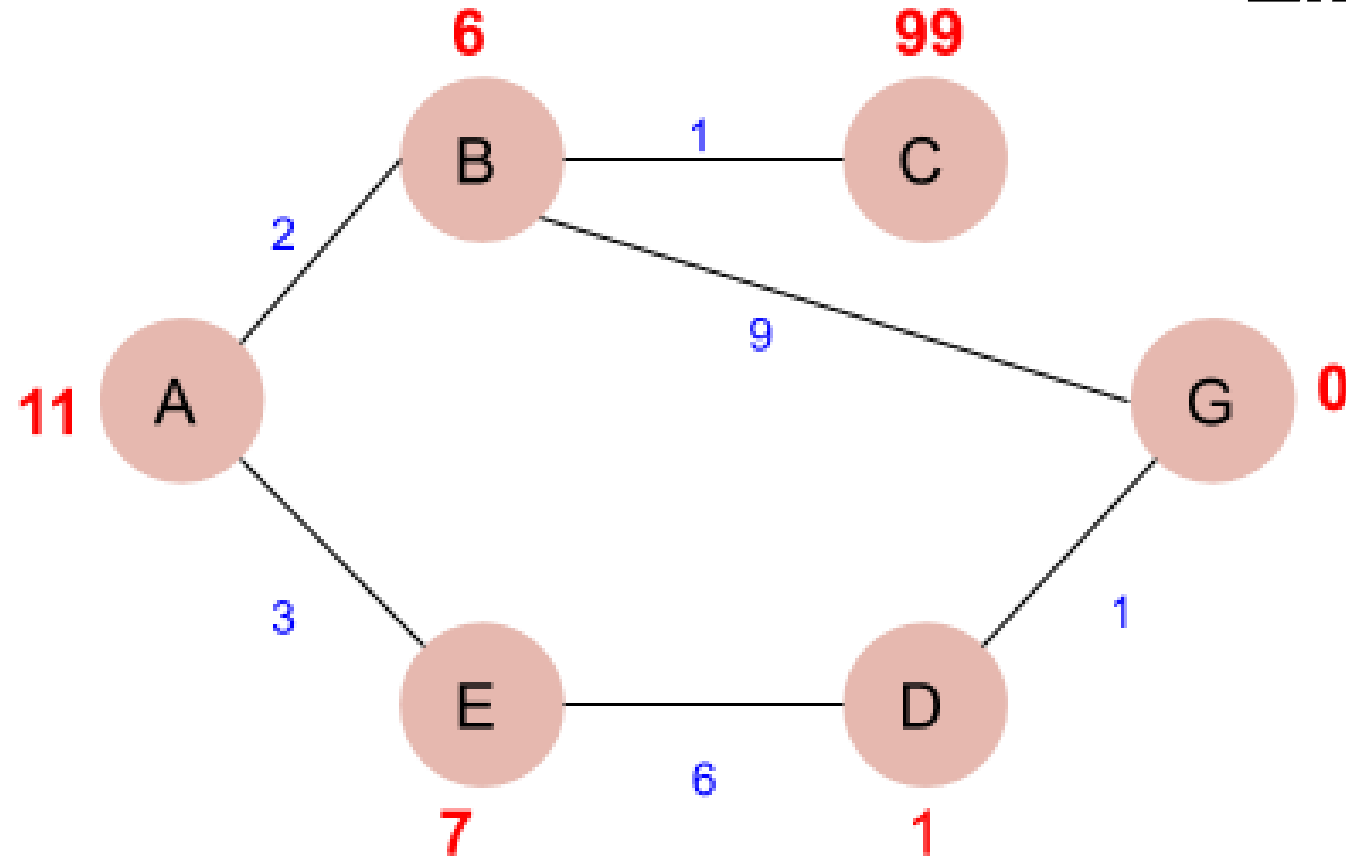
Example 1

State	$h(n)$
S	5
A	3
B	4
C	2
D	6
G	0



Example: 2

Example: 3



Thank You