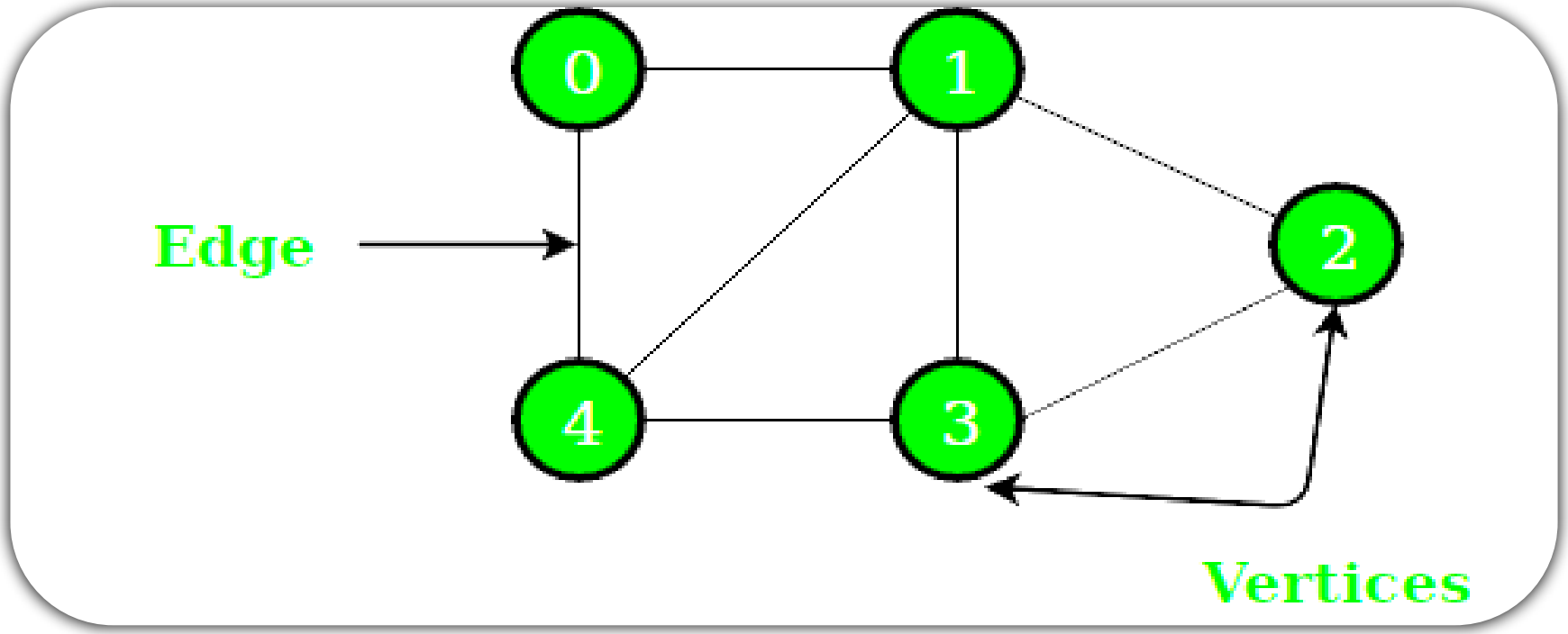
CHAPTER NINE

GRAPHS

Definition of a graph

- Graphs are a type of non-linear data structure which consists of set of nodes and links between those nodes.
- Nodes or vertices are the individual data elements, and a link connecting two nodes is called an edge or an arc.
- So, basically a Graph G is a collection of vertices (V) and edges (E).
- The pair is ordered because (u, v) is not same as (v, u) in case of directed graph (di-graph).
- The pair of form (u, v) indicates that there is an edge from vertex u to vertex v.
- The edges may contain weight/value/cost

In the Graph below; the set of vertices V = {0,1,2,3,4} and the set of edges E = {(0,1), (1,2), (2,3), (3,4), (0,4), (1,4), (1,3)}.

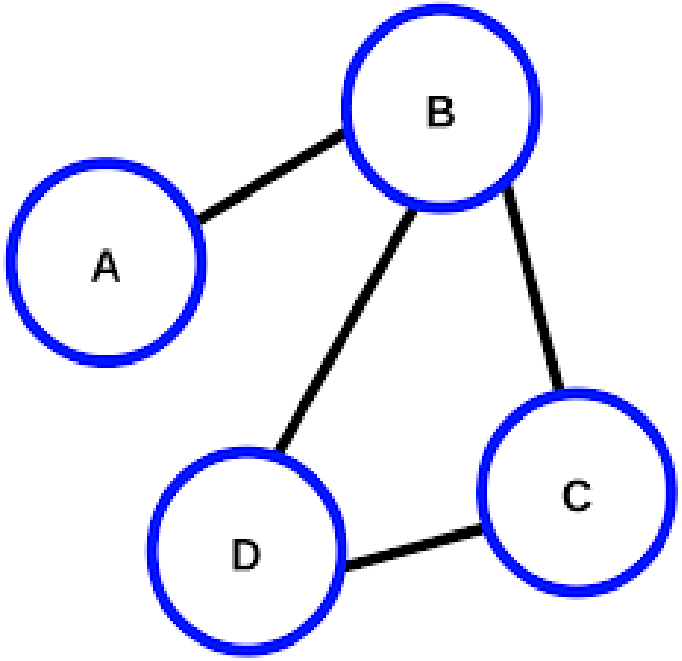


Example of a Graph that Consists of four nodes and edges

- The graph can be defined as

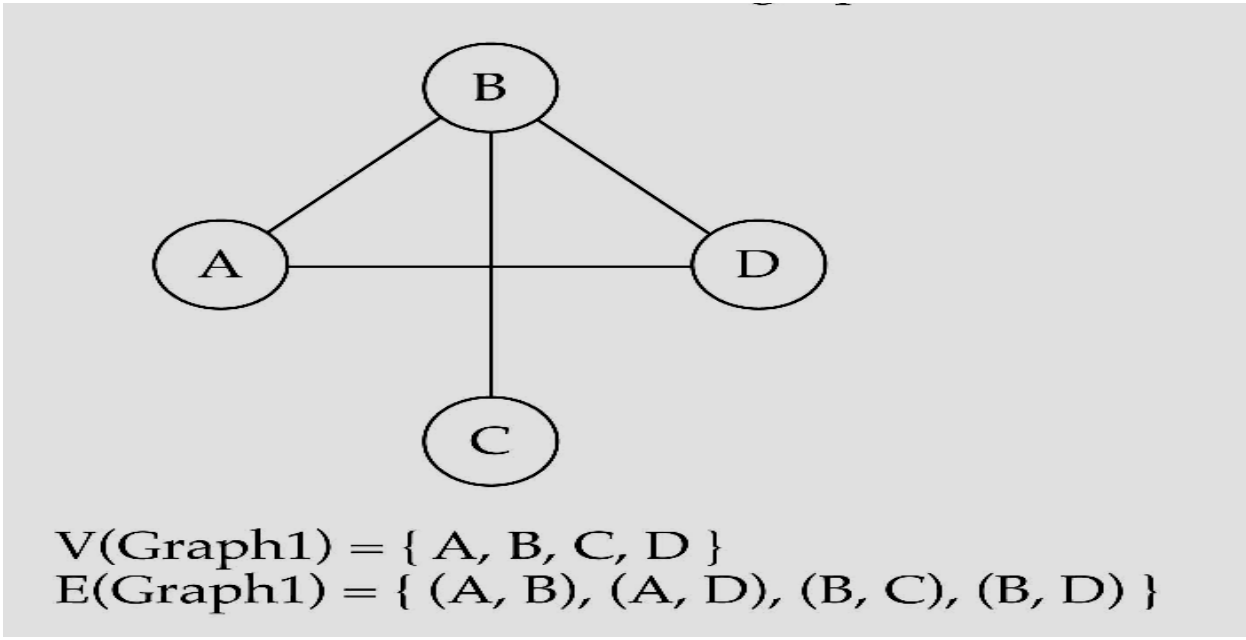
G =(V, E), where V = {A, B, C, D} and E = {(A, B), (B, C), (B, D), (C, D)}.

- A **path** is a collection of edges that connects a sequence of nodes together.
- In our image, {(A, B), (B, C)} is a **path** as it connects nodes A to C. Similarly {(A, B), (B, C), (C, D)} is also a path as it connects node A to D.



Directed vs. undirected graphs

- When the edges in a graph have no direction, the graph is called *undirected*



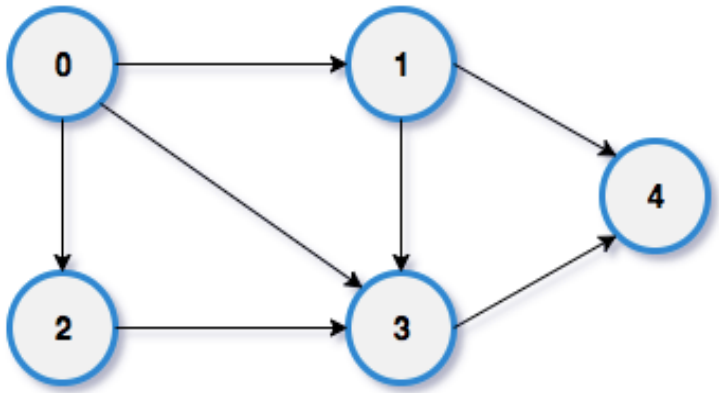
Directed vs. undirected graphs (cont.)

- A **directed graph** is a set of **vertices** (nodes) connected by **edges**, with each node having a direction associated with it.
- Edges are usually represented by arrows pointing in the direction the graph can be traversed.
- In the example on the right, the graph can be traversed from vertex **A** to **B**, but not from vertex **B** to **A**.
- In the undirected its possible to traverse in either direction .

E(Graph2) = {(1,3) (3,1) (5,9) (9,11) (5,7)

More on Directed Graphs

- In the Directed Graph, each edge(E) will be associated with **directions**.
- So, directed Graph have the **ordered pair** of edges.
- Means edge E1 (x, y) and E2 (y, x) are two different edges.
- Edge can only be traversed from the **specified direction**.
- It is also called the **digraph**(**Di**rected **graph**).



V = {0, 1, 2, 3, 4}

E = {<0,1>, <0,2>, <0,3>, <1,3>,<1,4>,<2,3>,<3,4>}

Cyclic Vs. Acyclic Graph

- A cycle, in the context of a graph, occurs when some number of vertices are connected to one another in a closed chain of edges.
- A graph that contains at least one cycle is known as **a cyclic graph**.
- Conversely, a graph that contains zero cycles is known as an **acyclic** graph.

Cyclic Graph

- Cylclic graph is a directed graph that contains a path from at least one node back to itself.
- In the example below; there is path, {(B,C),(C,E), (E,D), (D,B)}

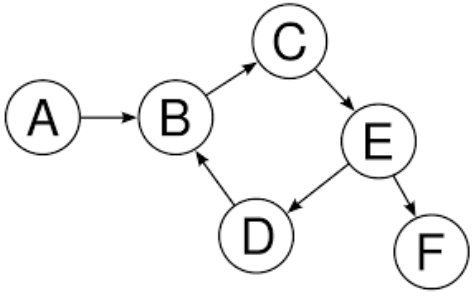
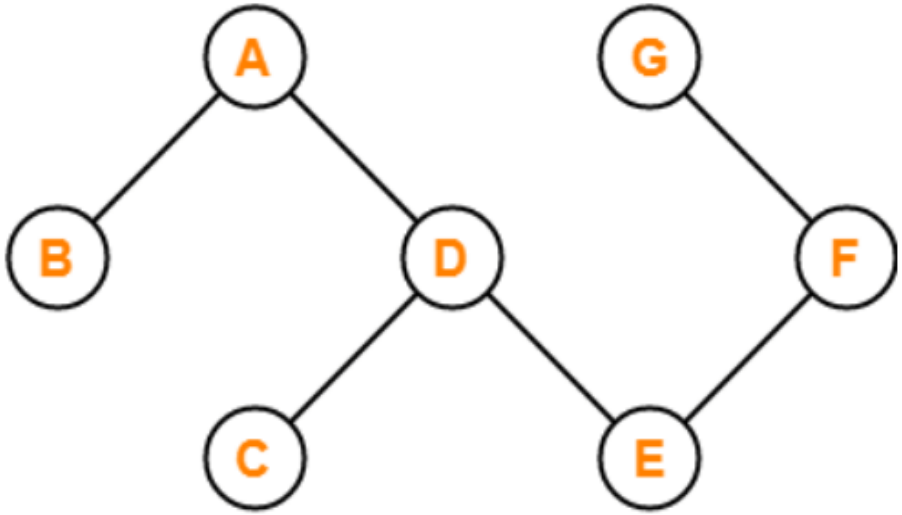


Figure 5 : Cyclic Graph

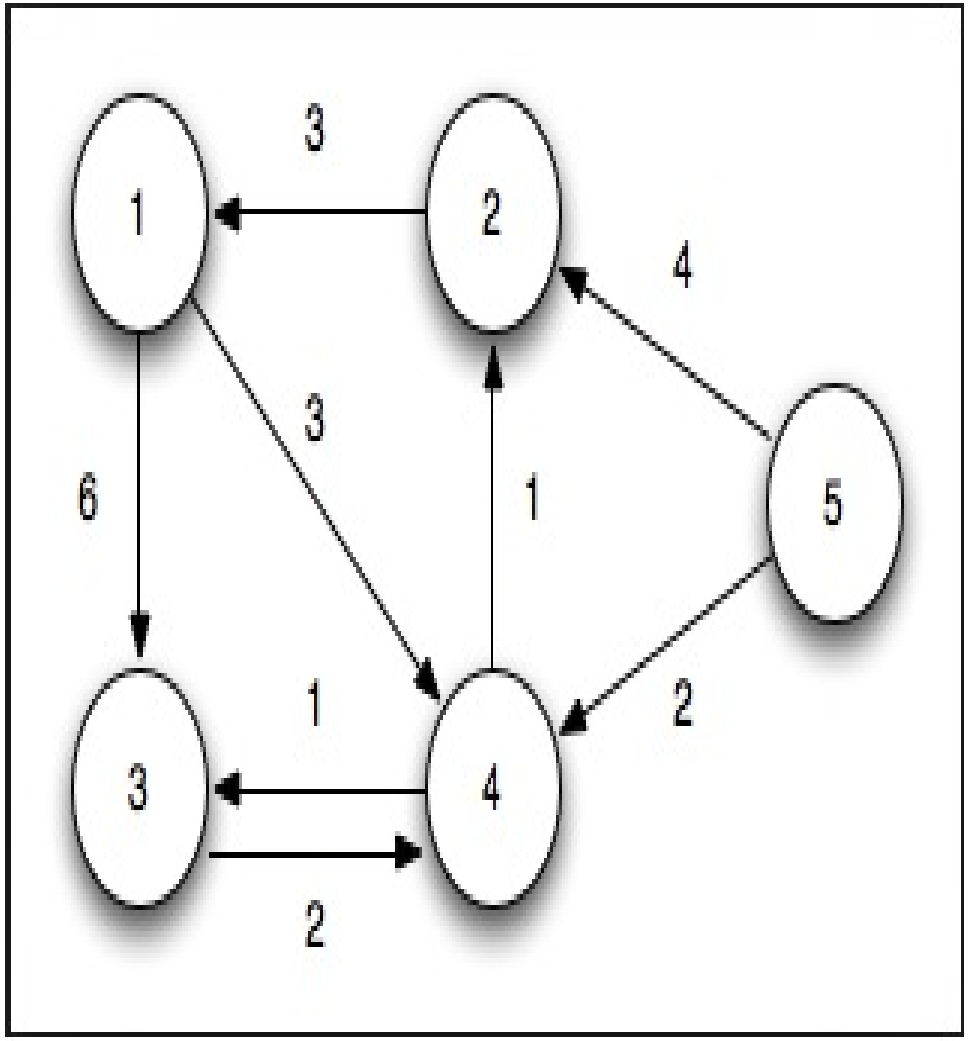
Acyclic Graph

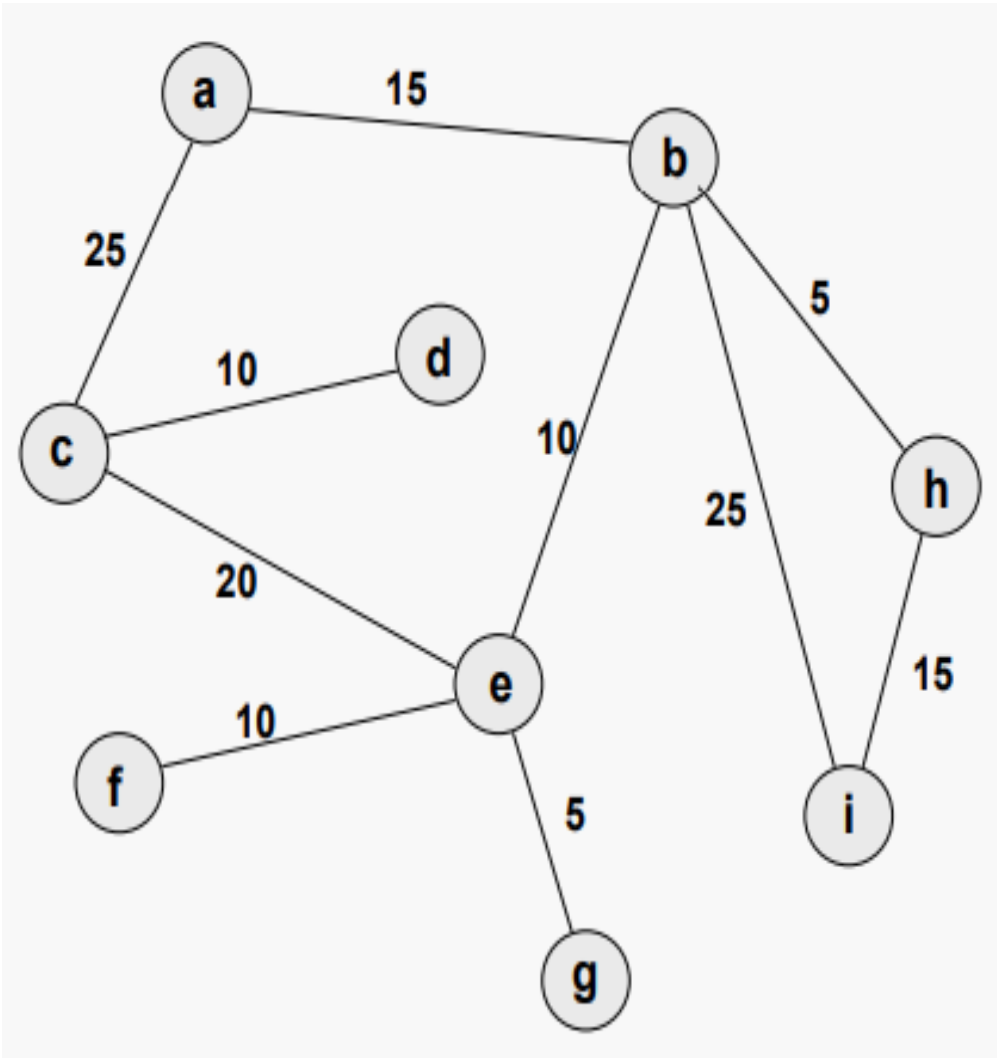
- A graph is called an acyclic graph if zero cycles are present, and an acyclic graph is the complete opposite of a cyclic graph.
- For example your are connected to the celebrity but the celebrity is not connected to you
-



Weighted Vs. Unweighted Edges

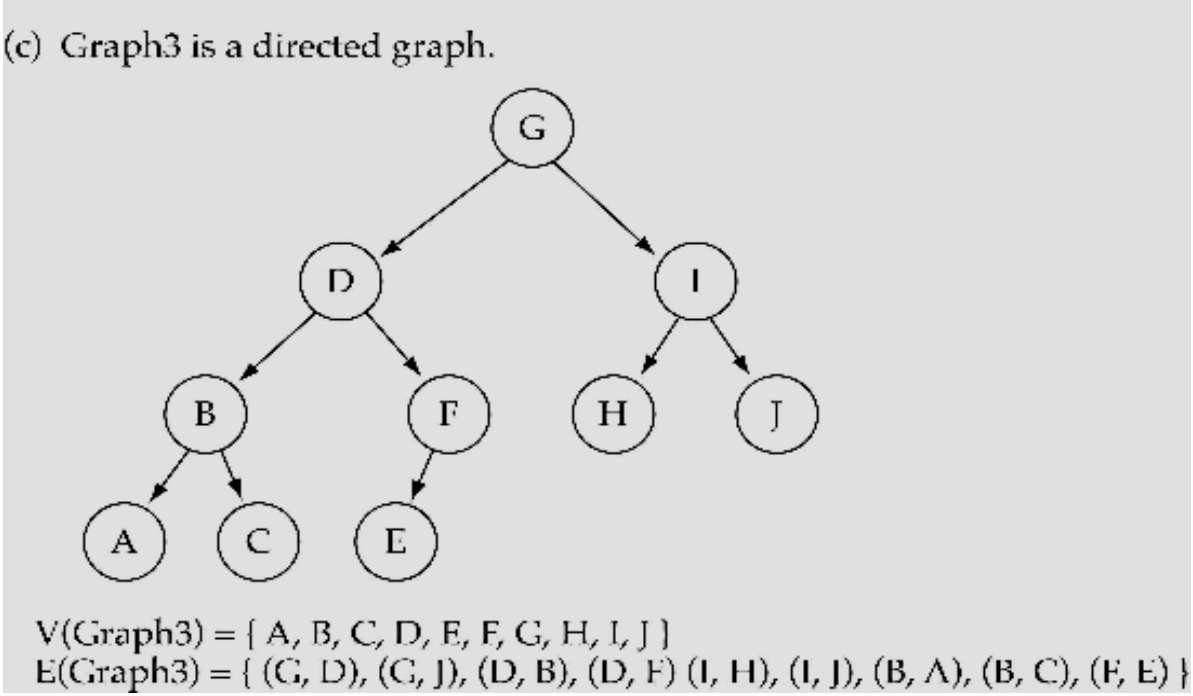
- Each edge of a graph has an associated numerical value, called a **weight**.
- **Weighted Graph:** A weighted graph is one where the edges are associated with a weight. This is generally the cost to traverse the edge. If the edges have no weights then it is unweight.
- The weights are usually used to compute the shortest path in the graph.
- Usually, the edge weights are nonnegative integers.
- The total weight of a path is the sum of the weights of its edges.
- Weighted graphs may be either directed or undirected.
- In applications, the weight may be a measure of the length of a route, the capacity of a line, the energy required to move between locations along a route, etc.


1
2
3
4
5
3
4
3
6
1
1
2
2


a
b
c
d
e
f
g
h
i
15
25
5
10
10
25
20
15
10
5

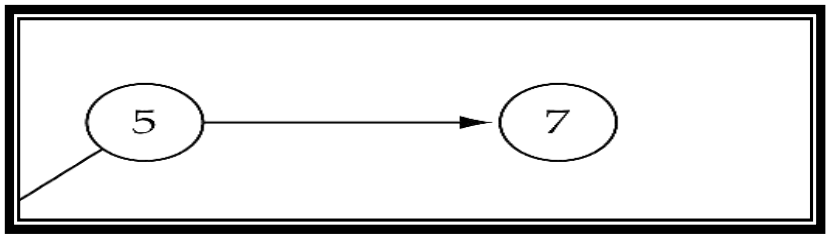
Trees vs graphs

- Trees are special cases of graphs!!



Graph terminology

- Adjacent nodes: two nodes are adjacent if they are connected by an edge



- Path: a sequence of vertices that connect two nodes in a graph
- Complete graph: a graph in which every vertex is directly connected to every other vertex

APPLICATIONS OF GRAPHS

Graphs – examples of graphs

- Road maps (directed, cyclic, weighted)
- Project network (directed, cyclic, weighted)
- Electrical circuits
- Molecules
- Relationships (directed, un-weighted?)
    - family tree
    - students on courses

APPLICATIONS OF GRAPHS

- Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.
- Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

REAL LIFE APPLICATIONS AREAS

1. **Social network graphs:** to tweet or not to tweet. Graphs that represent who knows whom, who communicates with whom, who influences whom or other relationships in social structures.
2. . **Utility graphs.** The power grid, the Internet, and the water network are all examples of graphs where vertices represent connection points, and edges the wires or pipes between them.

REAL LIFE APPLICATIONS AREAS

3. **Transportation networks.** In road networks vertices are intersections and edges are the road segments between them, and for public transportation networks vertices are stops and edges are the links between them. Such networks are used by many map programs such as Google maps, Bing maps and now Apple IOS 6 maps (well perhaps without the public transport) to find the best routes between locations. They are also used for studying traffic patterns, traffic light timings, and many aspects of transportation.

REAL LIFE APPLICATIONS AREAS

- **Document link graphs**. The best known example is the link graph of the web, where each web page is a vertex, and each hyperlink a directed edge. Link graphs are used, for example, to analyze relevance of web pages, the best sources of information, and good link sites.

REAL LIFE APPLICATIONS AREAS

- **Protein-protein interactions graphs.** Vertices represent proteins and edges represent interactions between them that carry out some biological function in the cell. These graphs can be used, for example, to study molecular pathways—chains of molecular interactions in a cellular process. Humans have over 120K proteins with millions of interactions among them.

REAL LIFE APPLICATIONS AREAS

- **Network packet traffic graphs.** Vertices are IP (Internet protocol) addresses and edges are the packets that flow between them. Such graphs are used for analyzing network security, studying the spread of worms, and tracking criminal or non-criminal activity.

REAL LIFE APPLICATIONS AREAS

- **Scene graphs.** In graphics and computer games scene graphs represent the logical or special relationships between objects in a scene. Such graphs are very important in the computer games industry.

REAL LIFE APPLICATIONS AREAS

- **Robot planning.** Vertices represent states the robot can be in and the edges the possible transitions between the states. This requires approximating continuous motion as a sequence of discrete steps. Such graph plans are used, for example, in planning paths for autonomous vehicles.

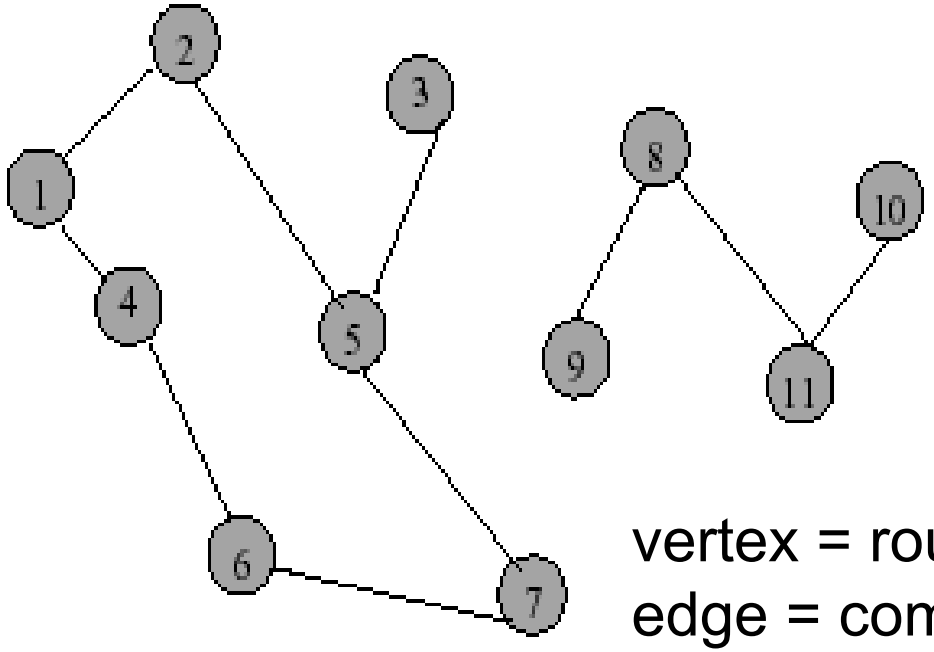
REAL LIFE APPLICATIONS AREAS

- **Neural networks.** Vertices represent neurons and edges the synapses between them. Neural networks are used to understand how our brain works and how connections change when we learn. The human brain has about 1011 neurons and close to 1015 synapses.

REAL LIFE APPLICATIONS AREAS

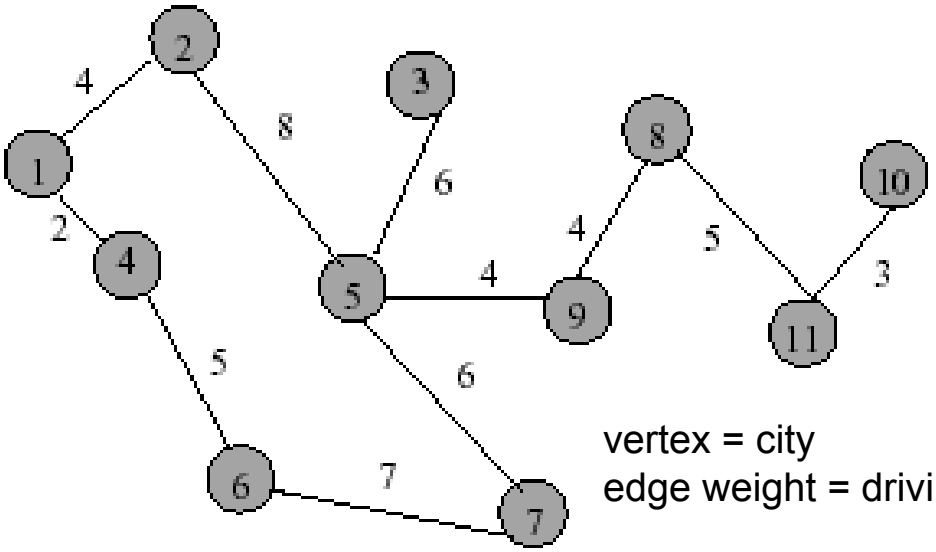
- **Graphs in compilers.** Graphs are used extensively in compilers. They can be used for type inference, for so called data flow analysis, register allocation and many other purposes. They are also used in specialized compilers, such as query optimization in database languages.

Applications – Communication Network



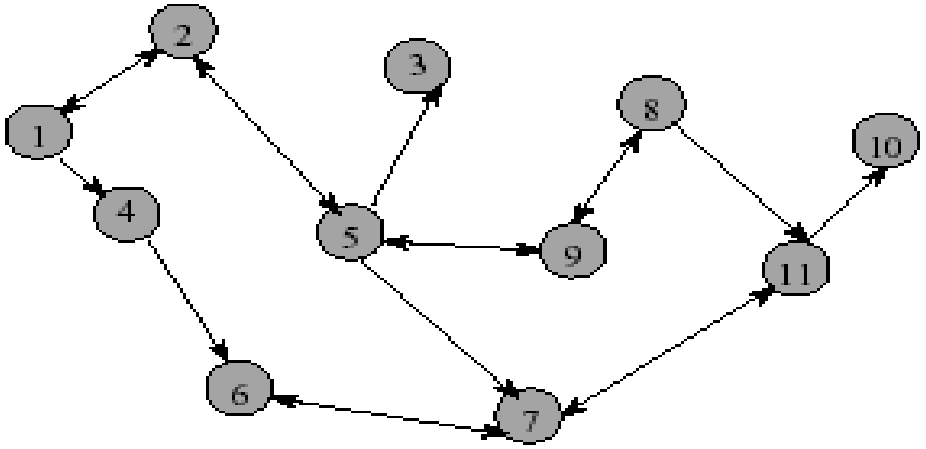
vertex = router
edge = communication link

Applications - Driving Distance/Time Map



vertex = city
edge weight = driving distance/time

Applications - Street Map



- Streets are one- or two-way.
- A single directed edge denotes a one-way street
- A two directed edge denotes a two-way street
- Read Example 16.1 and see Figure 16.2

Example

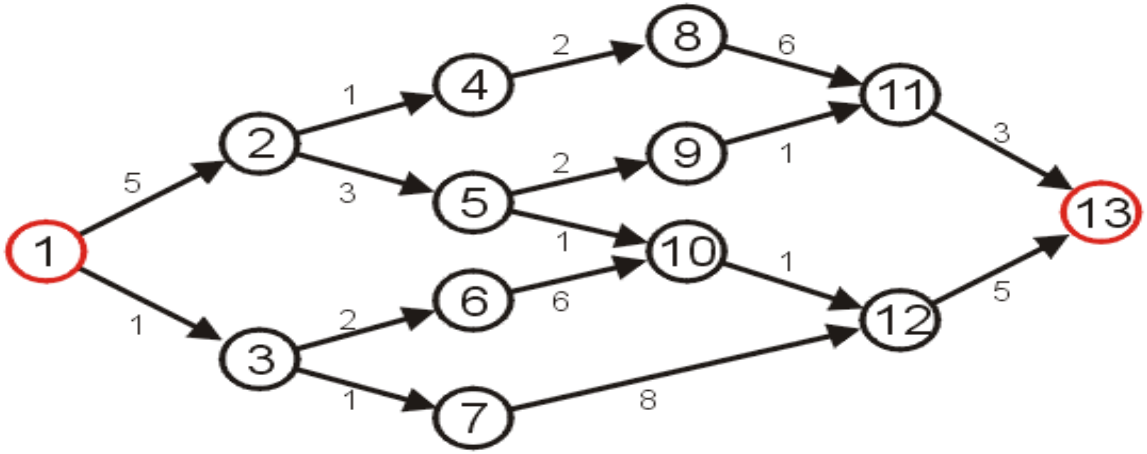
A map of cities and the roads connecting the cities can be represented in a computer using an undirected graph. The cities are vertices and the undirected edges are the roads that connect them. If we want to show the distances between the cities, we can use weighted graphs, in which each edge has a weight that represents the distance between two cities connected by that edge.

Example

Another application of graphs is in computer networks. The vertices can represent the nodes or hubs, the edges can represent the route. Each edge can have a weight that defines the cost of reaching from one hub to an adjacent hub. A router can use graph algorithms to find the shortest path between itself and the final destination of a packet.

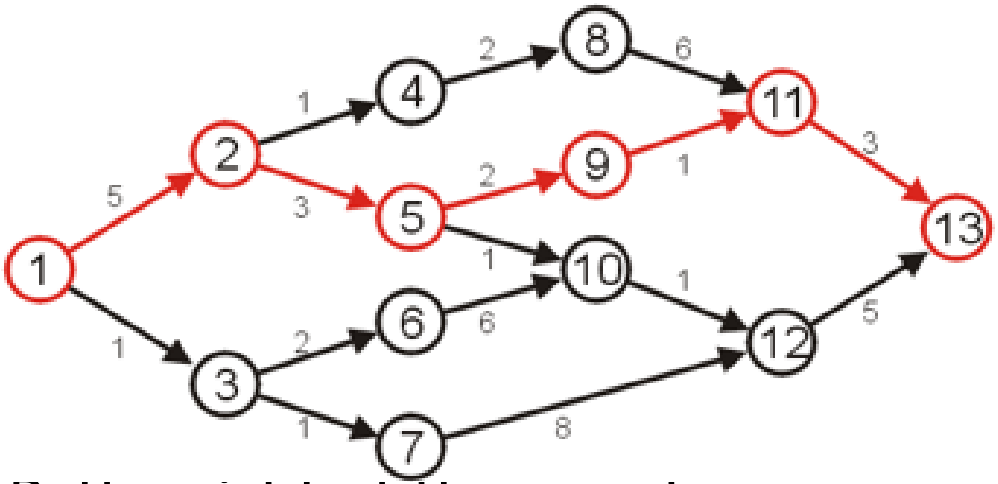
Shortest Path

- Given a weighted directed graph, one common problem is finding the shortest path between two given vertices
- ***Recall that in a weighted graph, the length of a path is the sum of the weights of each of the edges in that path.***
- Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13



Finding a Shortest Path

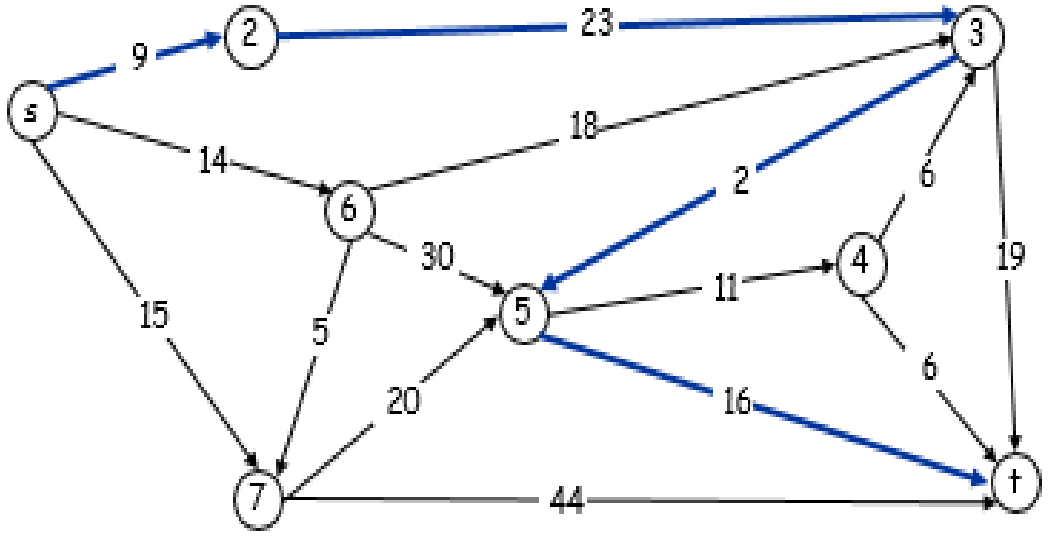
- After some consideration, we may determine that the shortest path is as follows, with length 14



- Other Path exist but they are longer.

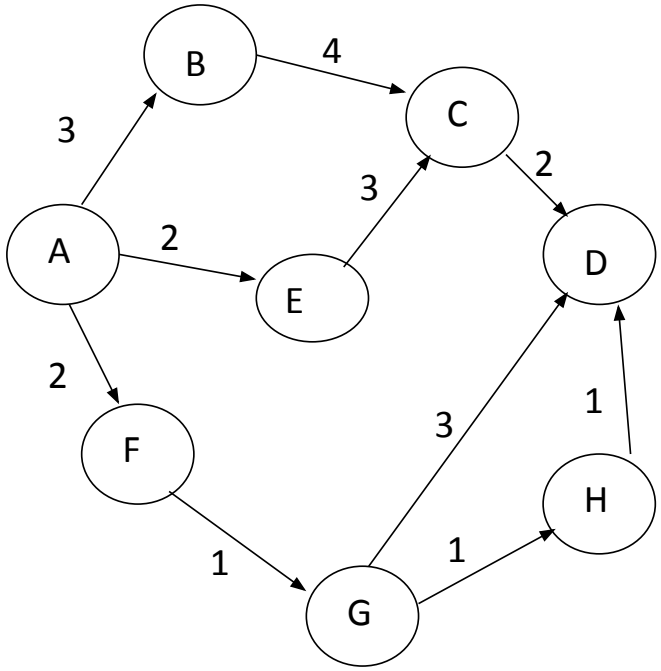
Shortest Path Example

- Given:
  - Weighted Directed graph G = (V, E).
  - Source *s*, destination *t*.
- Find shortest directed path from *s* to *t*.



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Find a shortest path in a graph (say, from A to D)



Applications of shortest path

- One application is circuit design: the time it takes for a change in input to affect an output depends on the shortest path.

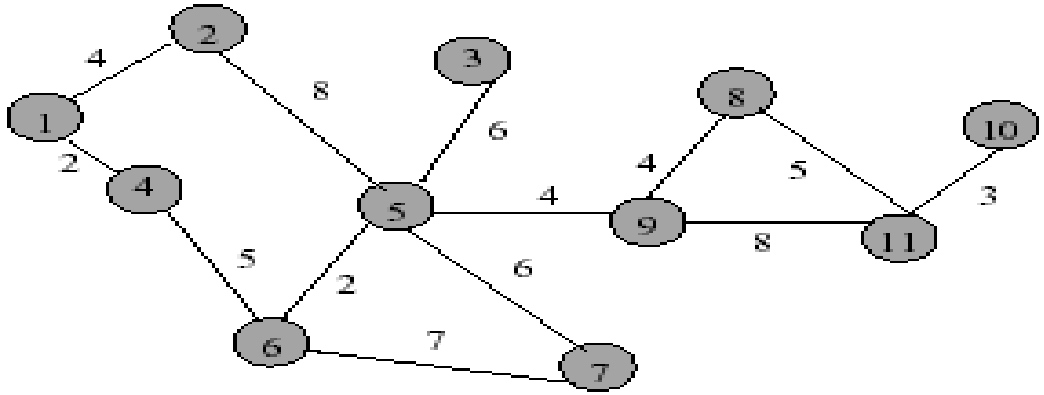
http://www.hp.com/

Simple Path

- A **simple path** is a path in which all vertices, except possibly in the first and last, are different.

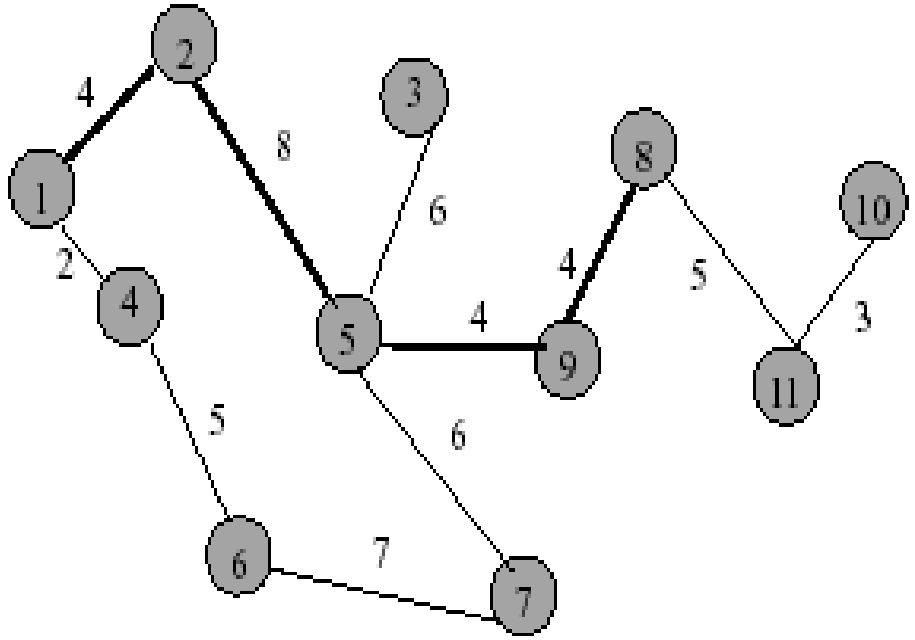
Length (Cost) of a Path

- Each edge in a graph may have an associated **length (or cost).**
- The length of a path is the sum of the lengths of the edges on the path
- What is the length of the path 5, 9, 11, 10 from 1?



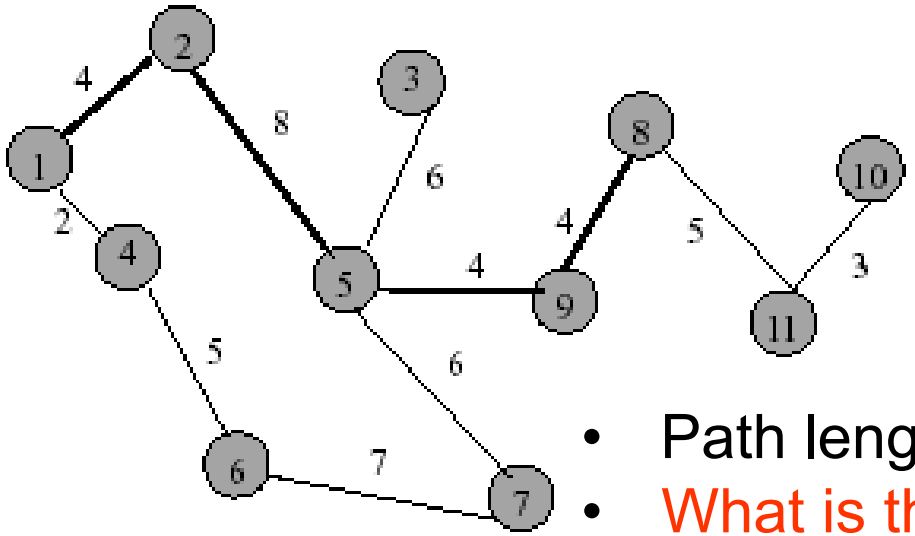
Path Finding

- Path between 1 and 8



- What is a possible path & its length?
- A path is 1, 2, 5, 9, 8 and its length is 20.

Another Path Between 1 and 8



- Path length is 28.
- What is the path?

Graph and its representations

- Following two are the most commonly used representations of graph.

1. Adjacency Matrix
2. Adjacency List

- There are other representations also like, Incidence Matrix and Incidence List.
- The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease.

Adjacency Matrix:

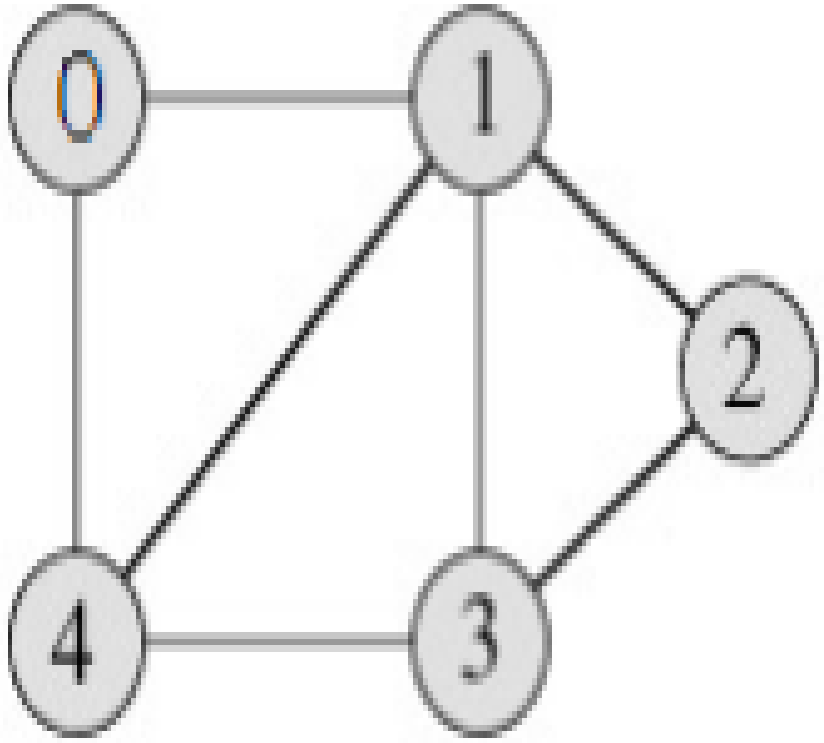
- Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
- Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.
- Adjacency matrix for undirected graph is always symmetric.
- Adjacency Matrix is also used to represent weighted graphs.
- If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.
- The adjacency matrix for the following example graph is presented next slide:

An Example of a Undirected Graph with five Vertices

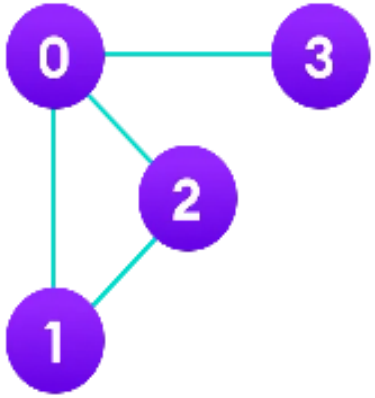
-

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

That is the adjancy matrix for the undirected graph besides here.

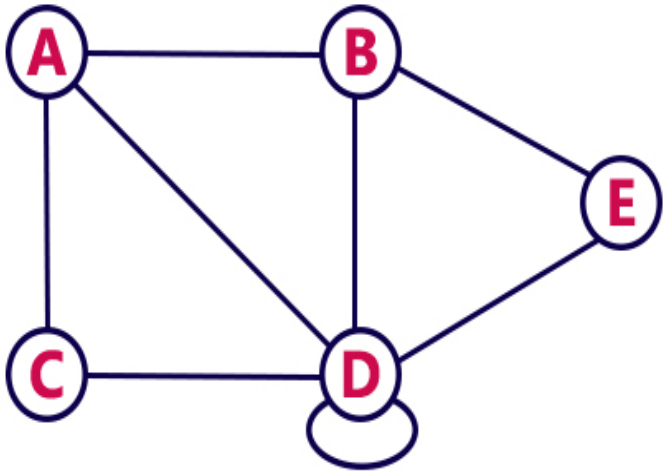


The adjacency matrix for the graph shown



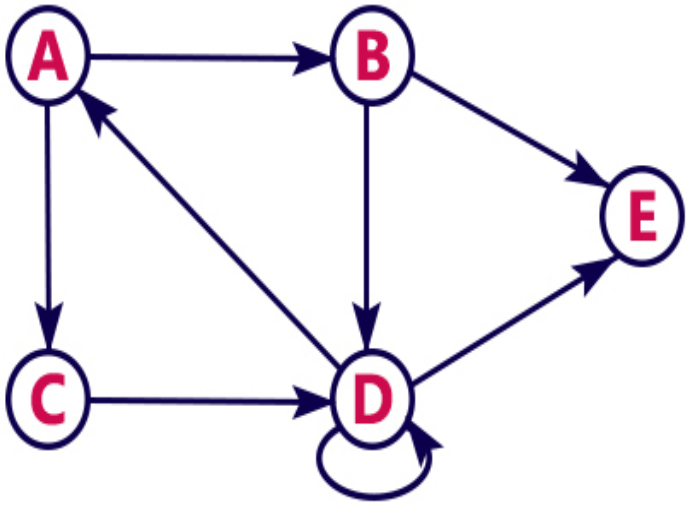
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

For example, consider the following undirected graph representation...



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

Directed graph representation...



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

Pros and Cons

- Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time.
- Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).
- Cons: Consumes more space O(V^2). Even if the graph is sparse (contains less number of edges), it consumes the same space.
- Adding a vertex is O(V^2) time

FINDING ADJACENCY LIST

- This representation is called the adjacency List.
- This representation is based on Linked Lists.
- **In this approach, each Node is holding a list of Nodes, which are Directly connected with that vertices.**
- At the end of list, each node is connected with the null values to tell that it is the end node of that list.

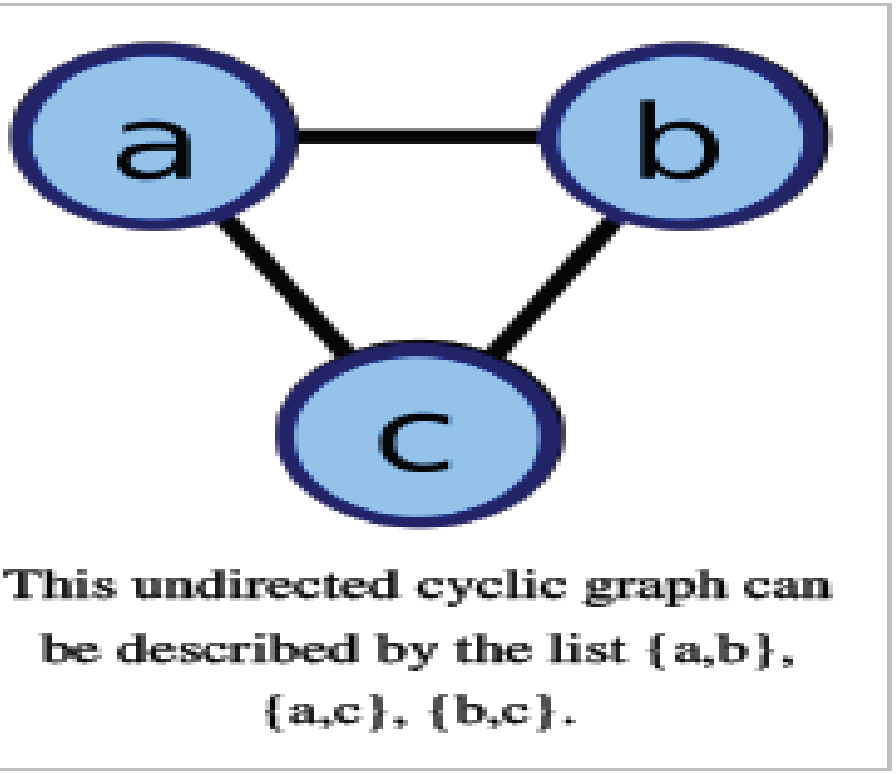
ADJACENCY LIST

- An adjacency list, also called an edge list, is one of the most basic and frequently used representations of a network.
- Each edge in the network is indicated by listing the pair of nodes that are connected.

ADJACENCY LIST

- In computer science, an adjacency list is a data structure for representing graphs.
- ***In an adjacency list representation, we keep, for each vertex in the graph, a list of all other vertices which it has an edge to (that vertex's "adjacency list").***

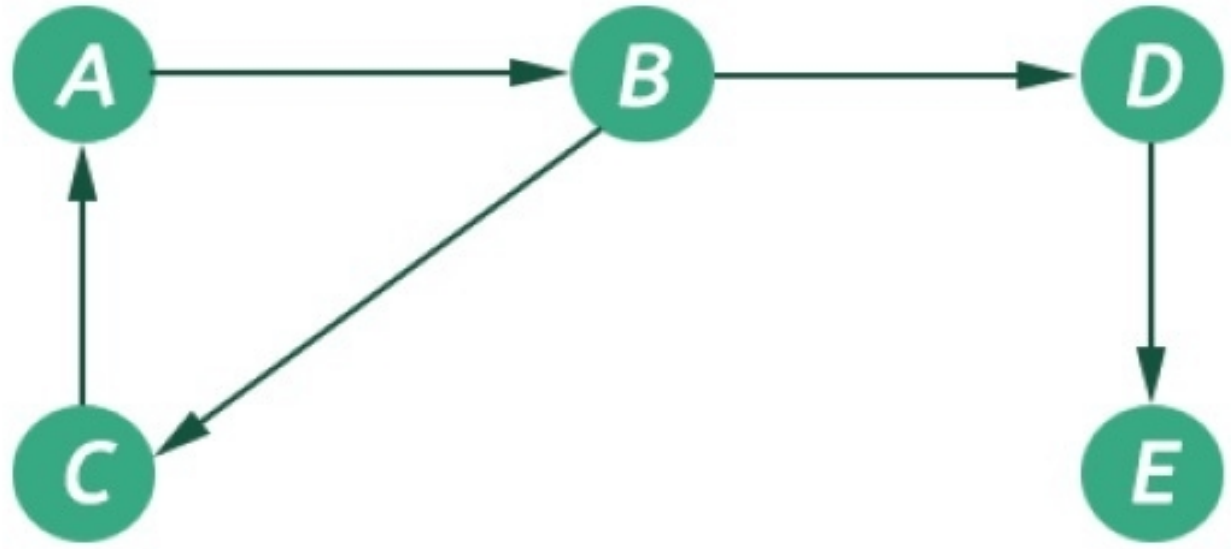
ADJACENCY LIST



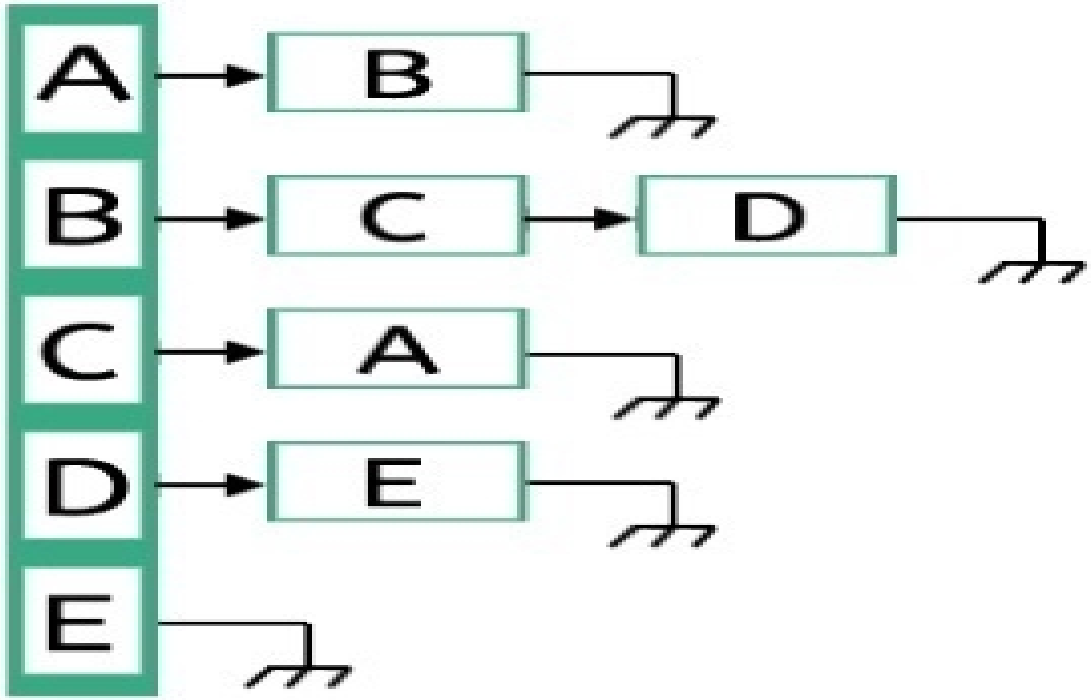
ADJACENCY LIST

The graph pictured above has this adjacency list representation:		
a	adjacent to	b,c
b	adjacent to	a,c
c	adjacent to	a,b

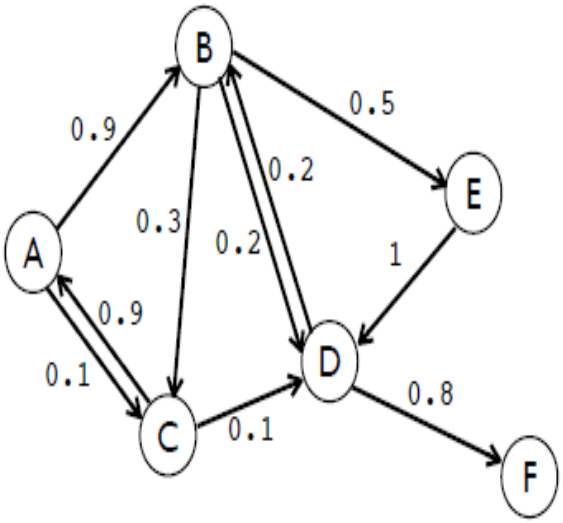
ADJACENCY LIST



ADJACENCY LIST



FINDING ADJACENCY LIST & MATRIX



(a) Graph

Adjacency List

A	(B, 0.9) (C, 0.1)
B	(C, 0.3) (D, 0.2) (E, 0.5)
C	(D, 0.1) (A, 0.9)
D	(B, 0.2) (F, 0.8)
E	(D, 1)
F	

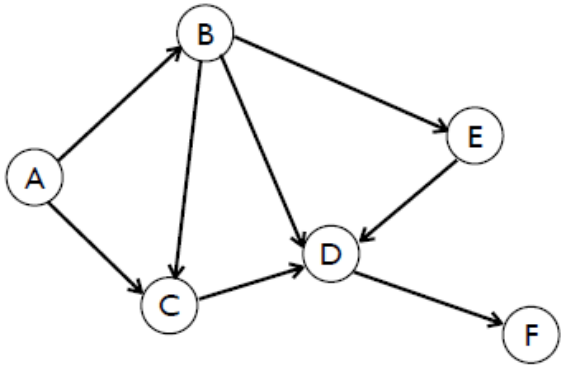
Adjacency Matrix

	A	B	C	D	E	F
A	∞	0.9	0.1	∞	∞	∞
B	∞	∞	0.3	0.2	0.5	∞
C	0.9	∞	∞	0.1	∞	∞
D	∞	0.2	∞	∞	∞	0.8
E	∞	∞	∞	1	∞	∞
F	∞	∞	∞	∞	∞	∞

(b) Representation

Figure 3.11: *Example of a directed, weighted graph*

FINDING ADJACENCY LIST & MATRIX



(a) Graph

Adjacency List

A	B	C		
B	C	D	E	
C	D			
D	F			
E	D			
F				

Adjacency Matrix

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	1	1	1	0
C	0	0	0	1	0	0
D	0	0	0	0	0	1
E	0	0	0	1	0	0
F	0	0	0	0	0	0

(b) Representation

Figure 3.9: *Example of a directed acyclic graph*

CASE STUDY: Modelling Networks With Graphs

- Many things that we experience in the real world are naturally inter-dependent and interconnected.
- A graph has 10 vertices, A to J. A connects to B and I. B connects to D and H. C connects to D. D connects to H. E connects to F and G. F connects to A. G connects to J. H connects to E. I connects to B and J. J connects to A.

a) Represent this graph in adjacency list and adjacency matrix respectively.

Graph Traversal Technique in Data Structure.

- Graph traversal is a technique to visit each node of a graph G. It is also use to calculate the order of vertices in traverse process.
- We visit all the nodes starting from one node which is connected to each other without going into loop.

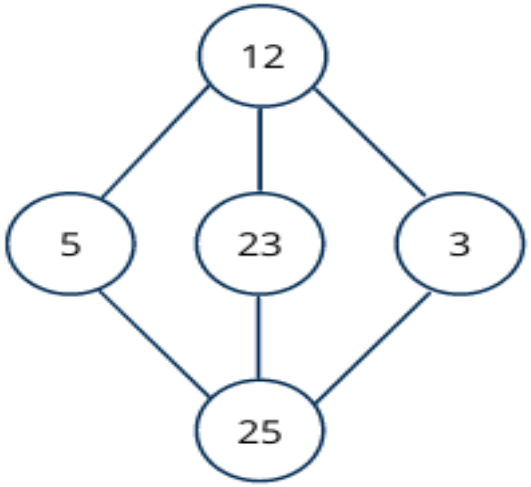
Breadth First Search (BFS) Traversal in Data Structure

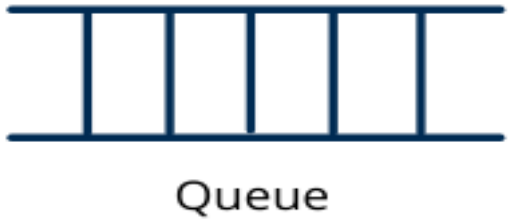
- Breadth-first search graph traversal techniques use a queue data structure as an auxiliary data structure to store nodes for further processing**.(FIFO)**
- The size of the queue will be the maximum total number of vertices in the graph.

Steps to implement BFS traversal

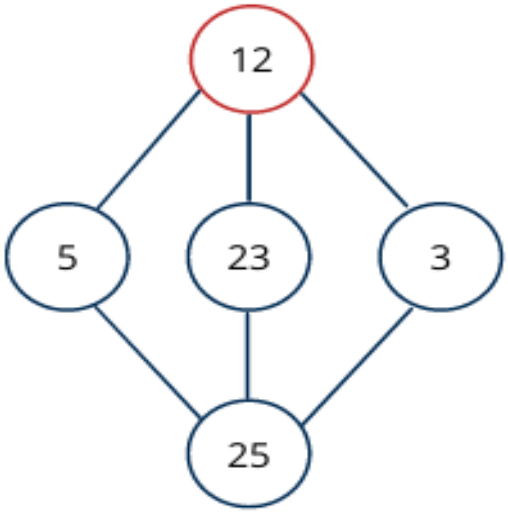
- Step 1 – First define a Queue of size n. Where n is the total number of vertices in the graph.
- Step 2 – Select any vertex which is a starting point from where traversal will start.
- Step 3 – Visit starting vertex and insert it into the Queue.
- Step 4 – Visit all the non-visited adjacent vertices which is connected to it and insert all non-visited vertices into the Queue.
- Step 5 – When there is no new vertex to be visited from the element which is top in a queue, Remove the top element which is the vertex from the queue.

Example Of BFS Graph Traversal
Initialize the queue



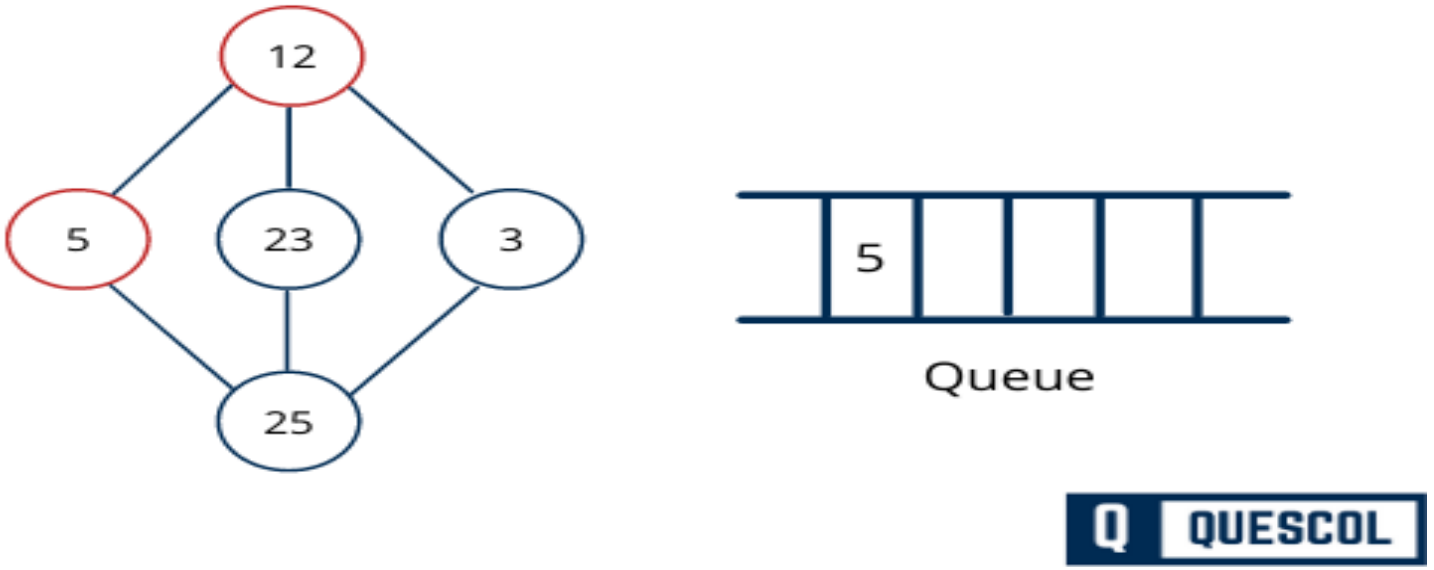


We start visiting from vertex 12 that can be considered as starting node, and mark it as visited.

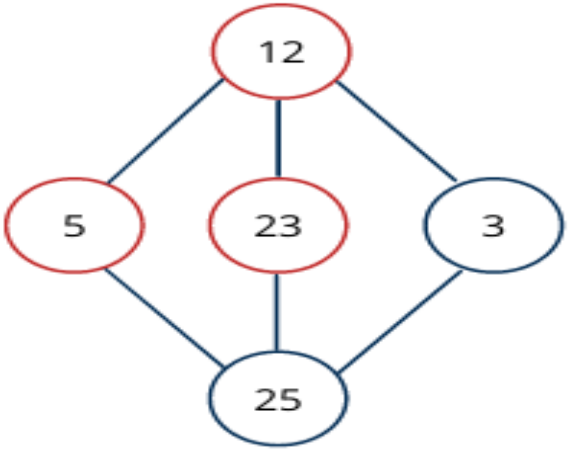


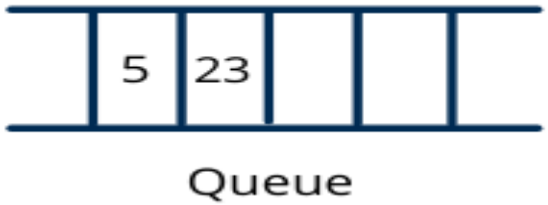
Queue

Now we will see an unvisited adjacent node from 12. In this example, we have three nodes and we can visit anyone. here we are traversing from left to right. We choose 5 and mark it as visited and enqueue it.

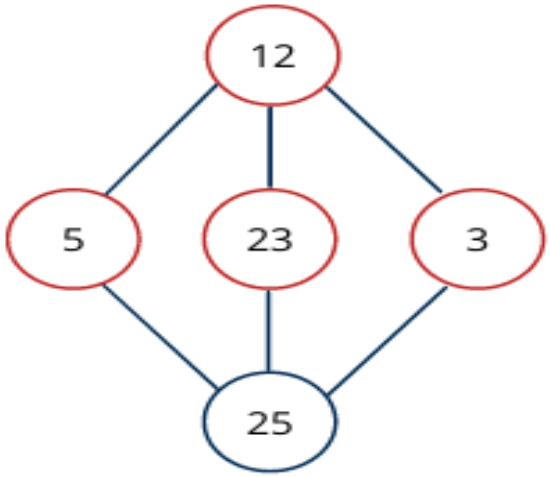


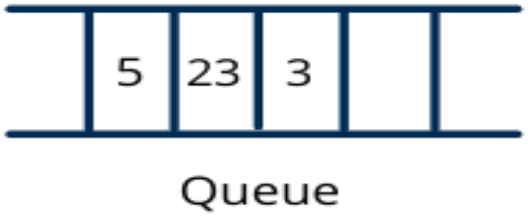
Next, visit the unvisited adjacent node from 12 to 23 . We mark it as visited and enqueue it.





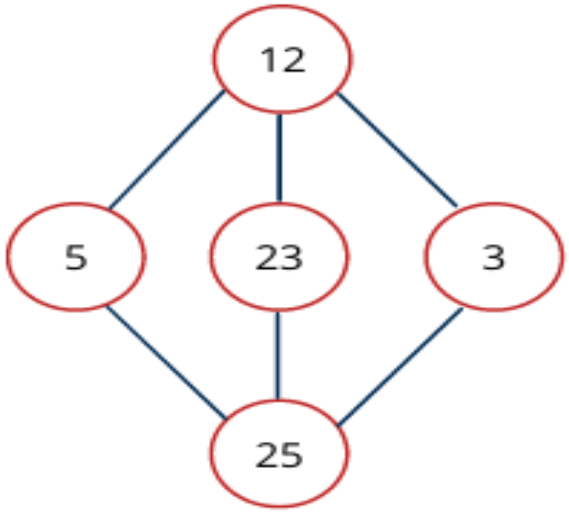
Next, the unvisited adjacent node from 12 is 3. We mark it as visited and enqueue it.

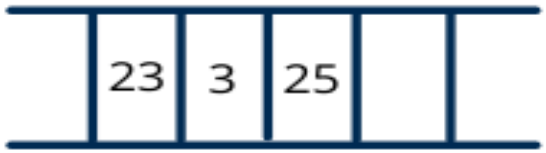




Queue

Now, all the connected adjacent node from 12 is traversed and no unvisited adjacent nodes left. So, we dequeue and find all connect vertex from 5.





Queue

BFS Traversal output

- From 5 we have 25 as unvisited adjacent node. We mark it as visited and enqueue it.
- Now if all nodes visited, we will dequeue all nodes from queue.
- ***So BFS Traversal output is : 12, 5, 23, 3, 25***

Thank You