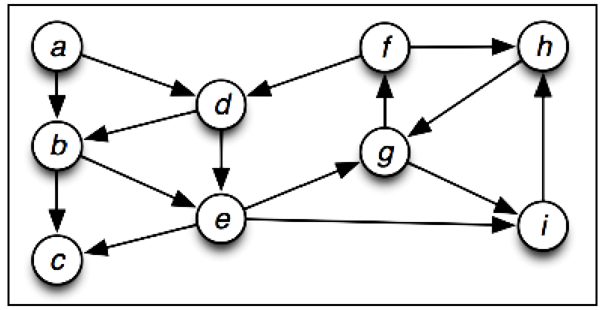
Graph Basics

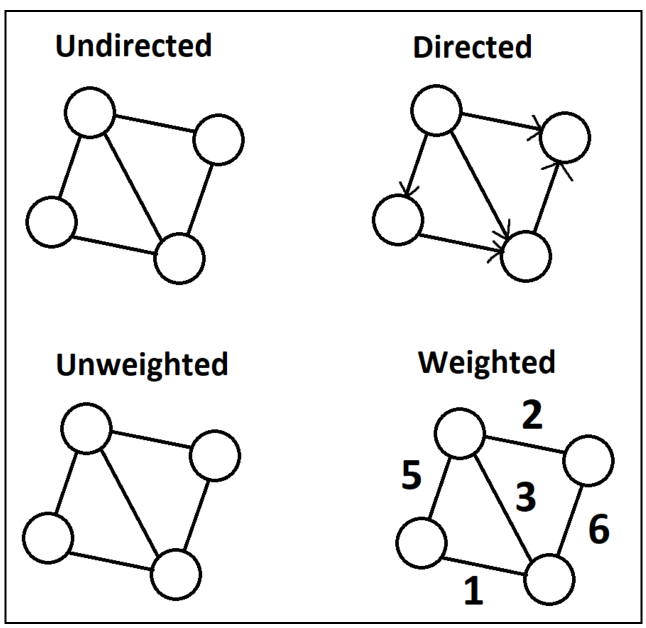


A graph is a pair: G = (V, E)
V: a set of vertices/nodes
E: a set of edges
Each edge is a pair of vertices.

If your problem has data and relationships, you might want to represent it as a graph
How do you choose a representation?

Usually:
Think about what your “fundamental” objects are
Those become your vertices.
Then think about how they’re related
Those become your edges.

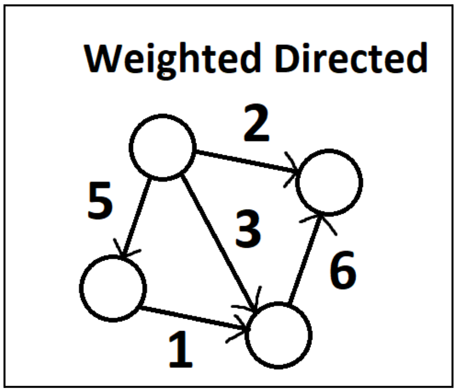
Types of Graphs:



Follow-up Question:

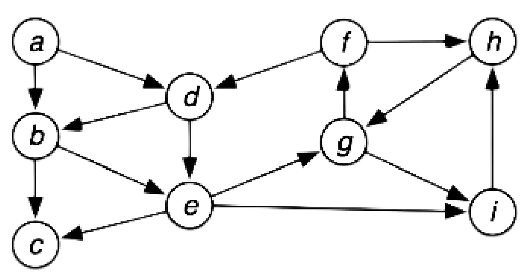
Q) Can you draw a weighted directed graph?

Ans:



Degree of Vertices:

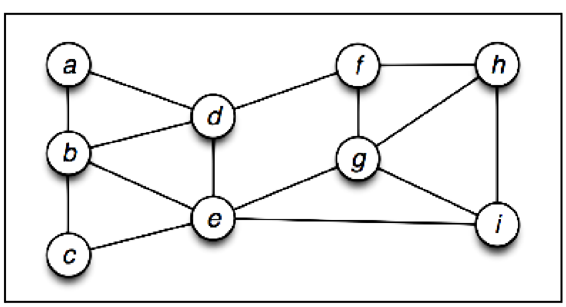
The degree of a vertex is the number of edges connected to that vertex. The number of incoming edges to a vertex v is called in-degree of the vertex. The number of outgoing edges from a vertex is called out-degree.



Vertex	in	out
a	0	2
b	2	2
c	2	0
d	2	2
e	2	3

Follow-up Question:

Q) What are the degrees of vertices of the following undirected graph?



Ans:

Vertex	deg
a	2
b	4
c	2
d	4
e	5

Few Graph Relationships:

If G is a graph with m edges, then

$$\sum_{v \in G} deg(v) = 2m.$$

If G is a directed graph (digraph) with m edges, then

$$\sum_{v \in G} indeg(v) = \sum_{v \in G} outdeg(v) = m.$$

If G is a simple ***undirected*** *graph with n vertices and m edges, then $m \leq n(n-1)/2$.*

If G is a simple ***directed*** *graph with n vertices and m edges, then $m \leq n(n-1)$.*

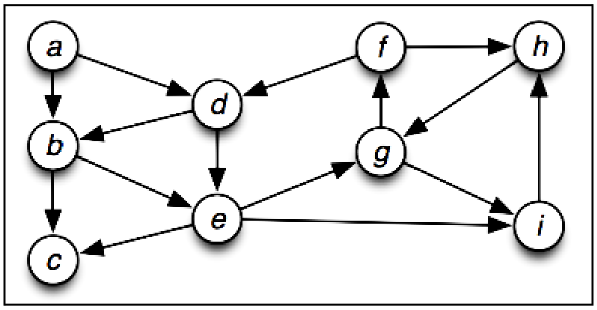
A **complete** graph is a graph in which each vertex is connected to every other vertex. That is, a complete graph is an undirected graph where every pair of distinct vertices is connected by a unique edge.

The formula for the number of edges m in a complete graph with n nodes is: m = n(n−1)/2:

- Each of the n nodes can be connected to n−1 other nodes.
- However, this counts each edge twice (once from each endpoint), so we divide by 2 to get the total number of unique edges.

Paths and Connectivity:

A path is a trail of vertices/nodes. It is a sequence of nodes in which each node is connected by an edge to the next.

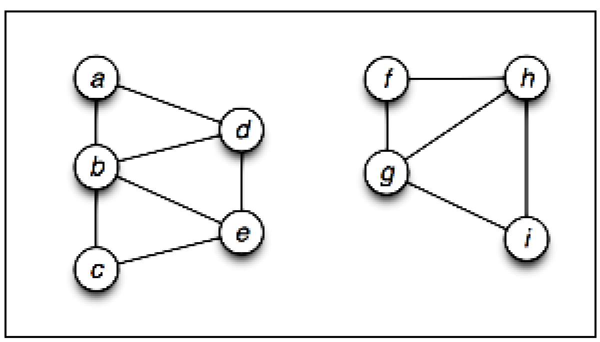


Here, ‘adeg’ is a path.
‘abeih’ is another path.
And so on.

A path is simple if all vertices are distinct. For example: ‘adeg’
A cycle is a path in which only the first and last vertices are equal. For example: ‘gihg’

Connected VS Unconnected Graphs:

The undirected graph G is connected, if for every pair of vertices u, v, there exists a path from u to v. If a graph is not connected, the vertices of the graph can be divided into connected components. Two vertices are in the same connected component iff they are connected by a path.

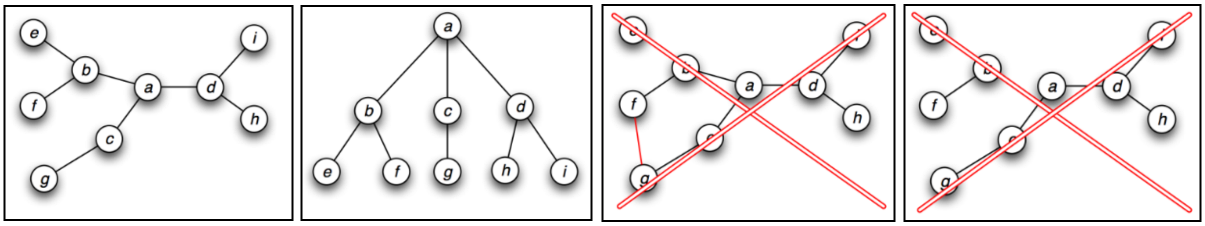


Trees:

An undirected graph is a tree if it is connected and does not contain a cycle (implying that every n-node tree has exactly n − 1 edges).

For an undirected graph G, any two of the following imply the third.

- G is connected
- G does not contain a cycle
- G has n − 1 edges

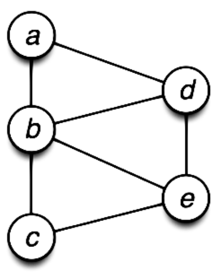


To represent Graphs, we use an Adjacency Matrix or Adjacency List.

Adjacency Matrix:

Represents the graph as an $n \times n$ matrix $A = (a_{i,j})$, where

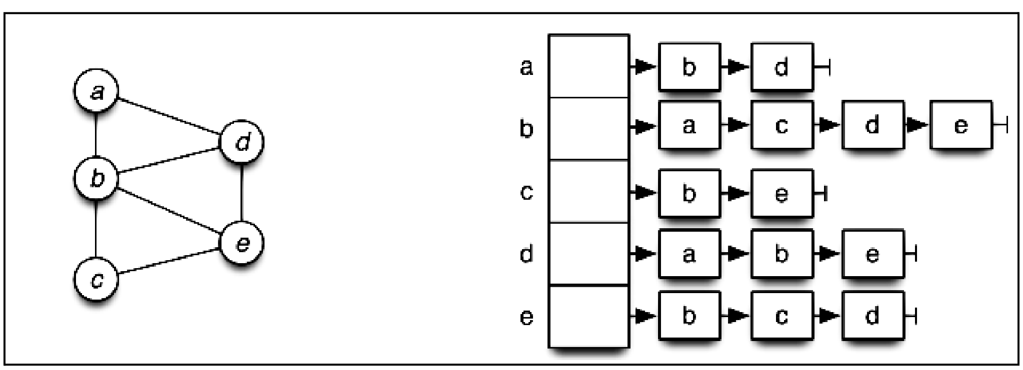
$$a_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

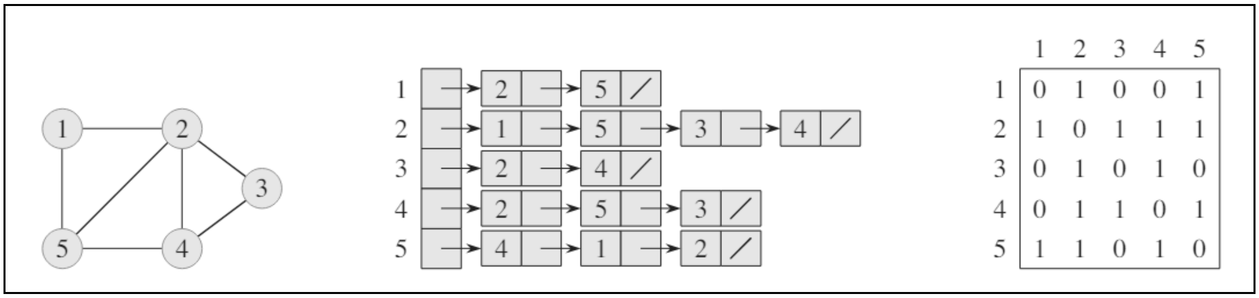


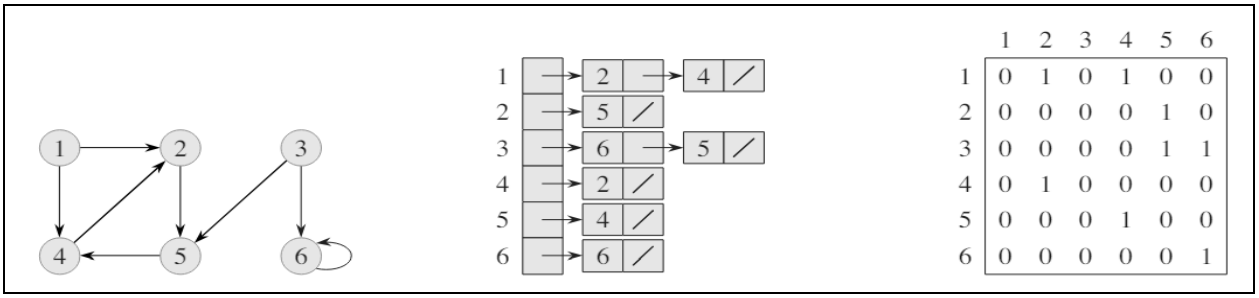
	a	b	c	d	e
a	0	1	0	1	0
b	1	0	1	1	1
c	0	1	0	0	1
d	1	1	0	0	1
e	0	1	1	1	0

Adjacency List:

Represent the graph by listing each vertex v_i its adjacent vertices in a list. (Representation can be linked list or another appropriate structure) For each u ∈ V, the adjacency list Adj[u] contains all the vertices v such that there is an edge (u,v) ∈ E.







If G is a directed graph, the sum of the lengths of all the adjacency lists is |E| since an edge of the form (u,v) is represented by having v appear in Adj[u].

If G is an undirected graph, the sum of the lengths of all the adjacency lists is 2 |E|, since if (u,v) is an undirected edge, then u appears in v's adjacency list and vice versa.

Differences between Adjacency List and Adjacency Matrix:

Operations	Adjacency Matrix	Adjacency List
Space Complexity	Use of VxV matrix, so space required in worst case is $O(\|V\|^2)$.	For each vertex, we store its neighbors. This requires O(V) for the vertices and O(E) for the edges, resulting in an overall space complexity of O(V + E).
Adding a vertex	Adding a new vertex requires increasing the matrix size to $(\|V\|+1)^2$, requiring a full matrix copy. Thus, the complexity is $O(\|V\|^2)$.	To add a new vertex, add a key to hashmap or a head node which takes O(1)

Adding an edge	To add an edge say from i to j, matrix[i][j] = 1 which requires O(1) time.	To add a new edge, add a value to the key of the hashmap or add a node to the tail which takes O(1)
Removing a vertex	Removing a vertex requires reducing the matrix size to $\|V\|^2$ from $(\|V\|+1)^2$, requiring a full matrix copy. Thus, the complexity is $O(\|V\|^2)$.	Removing a vertex involves O(|V|) for searching and O(|E|) for edge traversal, resulting in a time complexity of O(|V| + |E|).
Removing an edge	To remove an edge say from i to j, matrix[i][j] = 0 which requires O(1) time.	Removing an edge requires traversing through all the edges, resulting in a time complexity of O(|E|).
Finding an edge of a vertex	Checking for an existing edge involves directly accessing matrix[i][j], resulting in O(1) time complexity.	Checking for an existing edge, we need to examine the adjacent vertices of a given vertex. A,s a vertex can have at most O(|V| - 1) neighbors, the worst-case time complexity is O(|V|).

Adjacency List representation provides a compact way to represent sparse graphs - for which $|E|$ is much less than $|V|^2$ - it is usually the method of choice. You might prefer an adjacency-matrix representation, however, when the graph is dense - $|E|$ is close to $|V|^2$

We will be using adjacency lists to represent graphs unless stated otherwise.

Follow-up Question:

Q) When to use the Adjacency Matrix over the Adjacency List?

Ans: Adjacency Matrix representation is used when the graph is dense - $|E|$ is close to $|V|^2$ whereas Adjacency List representation provides a compact way to represent sparse graphs - for which $|E|$ is much less than $|V|^2$

Q) How to store weights in an adjacency matrix?

Ans: Add weight values in the matrix instead of ‘1’s.

Q) How to store weights in an adjacency list?

Ans: Add weight values along with neighbors in an array or linked list.