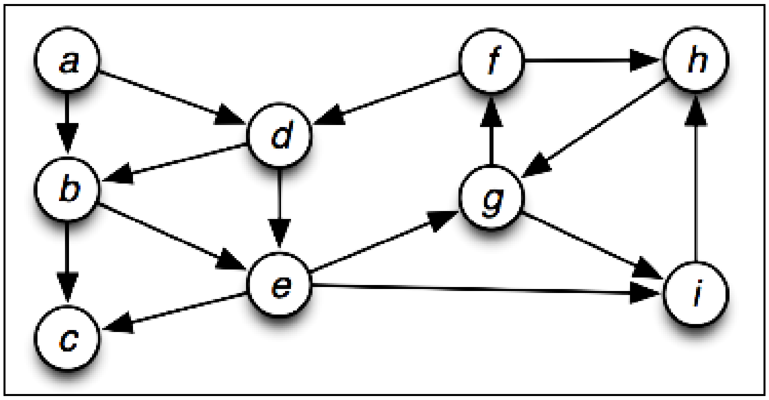
Graph

Graph G is a pair (V, E), where V is a finite set (set of vertices) and E is a finite set of pairs from V (set of edges). We will often denote n = |V|, m = |E|.



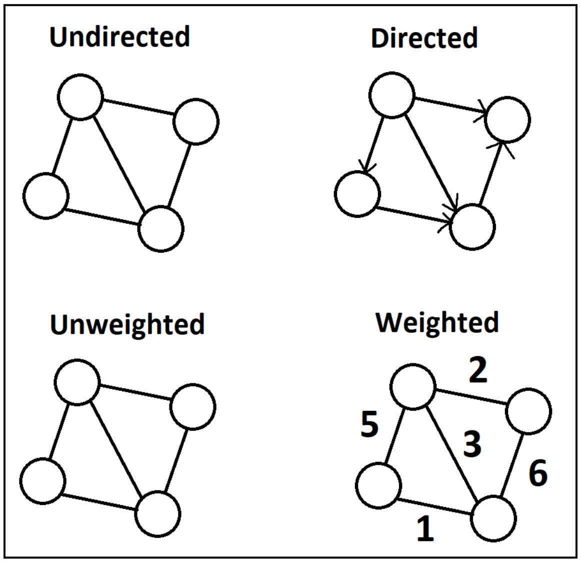
If your problem has data and relationships, you might want to represent it as a graph
How do you choose a representation?

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:

Graph G can be directed, if E consists of ordered pairs, or undirected, if E consists of unordered pairs. If (u, v) ∈ E, then vertices u and v are adjacent.

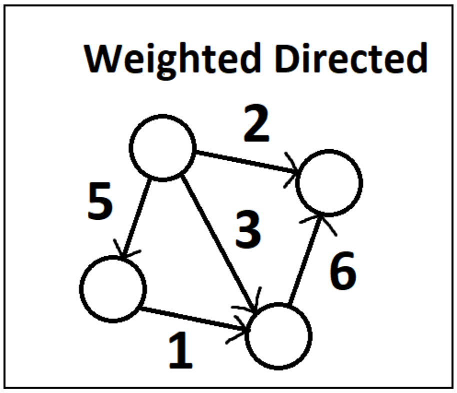
We can assign weight function to the edges: $w^G(e)$ is a weight of edge e ∈ E. The graph which has such a function assigned is called weighted.



Follow-up Question:

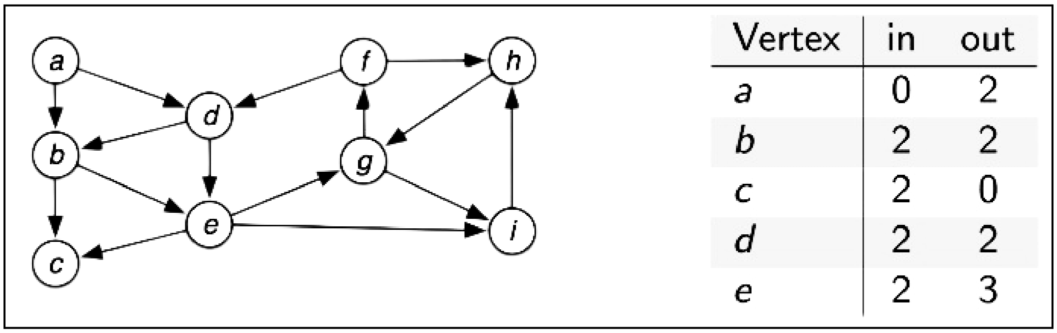
Q) Can you draw a weighted directed graph?

Ans:



Degree of Vertices:

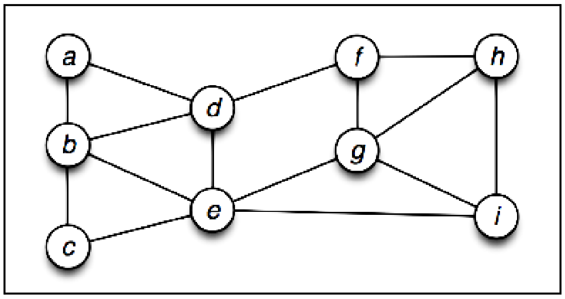
The degree of a vertex is the number of edges connected to that vertex. The number of incoming edges to a vertex v is called in-degree of the vertex. The number of outgoing edges from a vertex is called out-degree.

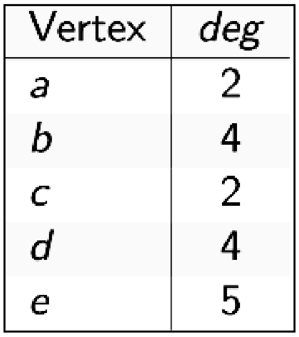


Vertex	in	out
a	0	2
b	2	2
c	2	0
d	2	2
e	2	3

Follow-up Question:

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





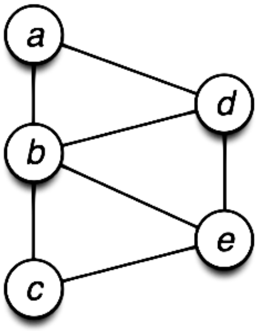
Vertex	deg
a	2
b	4
c	2
d	4
e	5

Graph Representation:

Adjacency Matrix:

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

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

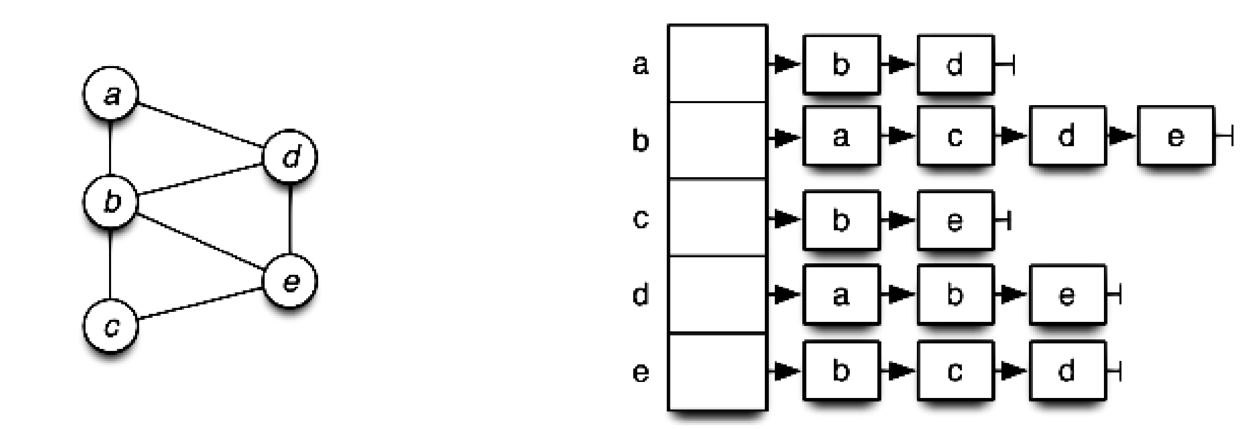


	a	b	c	d	e
a	0	1	0	1	0
b	1	0	1	1	1
c	0	1	0	0	1
d	1	1	0	0	1
e	0	1	1	1	0

Space Complexity: O(V^2), for storing in a matrix of dimension V x V

Adjacency List:

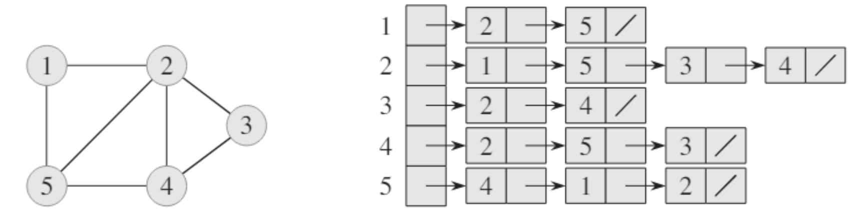
Represent the graph by listing for each vertex v_i its adjacent vertices in a linked list. For each u ∈ V, the adjacency list Adj[u] contains all the vertices v such that there is an edge (u,v) ∈ E.



Space Complexity: O(V+E), V is for storing V vertices in an array, and E is for storing E edges in linked lists.

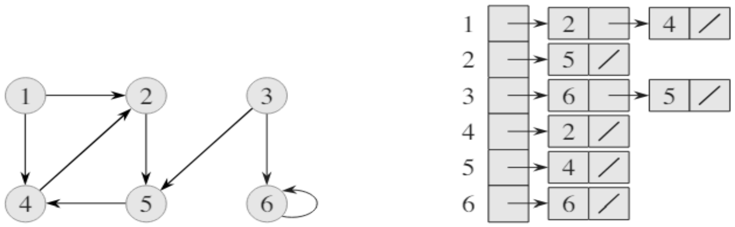
More Examples:

Undirected and Unweighed:



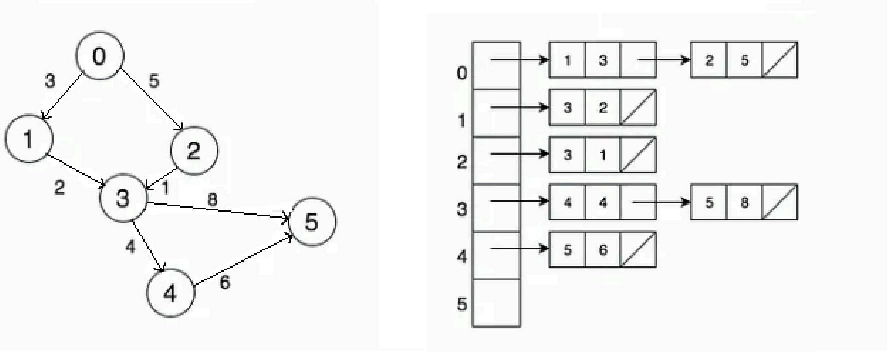
$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0
\end{array}$$

Directed and Unweighed:



$$\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

Directed and Weighed:



$$\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 3 & 5 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 2 & 0 & 0 \\
2 & 0 & 0 & 0 & 1 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 4 & 8 \\
4 & 0 & 0 & 0 & 0 & 0 & 6 \\
5 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Adjacency Matrix Coding:

```
class AdjacencyMatrix:
        def __init__(self, vertices):
                self.vertices = vertices
                self.graph = np.zeros((vertices, vertices), dtype=int)
```

```
def add_unweighted_edge(u, v):
    # Add edge from u to v
    graph[u][v] = 1

    # Add edge from v to u (for undirected graph)
    graph[v][u] = 1
```

```
def add_weighted_edge(u, v, w):
    # Add edge from u to v
    graph[u][v] = w

    # Add edge from v to u (for undirected graph)
    graph[v][u] = w
```

Adjacency List Coding:

```
class Node:
    def __init__(self, vertex, weight = None):
        self.vertex = vertex
        self.weight = weight
        self.next = None
```

```
class AdjacencyList:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = np.array([None] * vertices)
```

```
def add_unweighted_edge(u, v):
    # Add edge from u to v
    n = Node(v)
    if graph[u] is None:
        graph[u] = n
    else:
        current = graph[u]
        while current.next is not None:
            current = current.next
        current.next = n

    # Add edge from v to u (for undirected graph)
    m = Node(u)
    if graph[v] is None:
        graph[v] = m
    else:
        current = graph[v]
        while current.next is not None:
            current = current.next
        current.next = m
```

```
def add_weighted_edge(u, v, w):
    # Add edge from u to v
    n = Node(v, w)
    if graph[u] is None:
        graph[u] = n
    else:
        current = self.graph[u]
        while current.next is not None:
            current = current.next
        current.next = n
```

```
# Add edge from v to u (for undirected graph)
m = Node(u, w)
if graph[v] is None:
    graph[v] = m
else:
    current = self.graph[v]
    while current.next is not None:
        current = current.next
    current.next = m
```