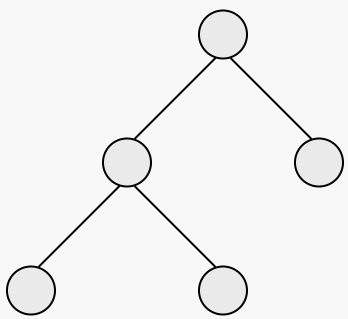
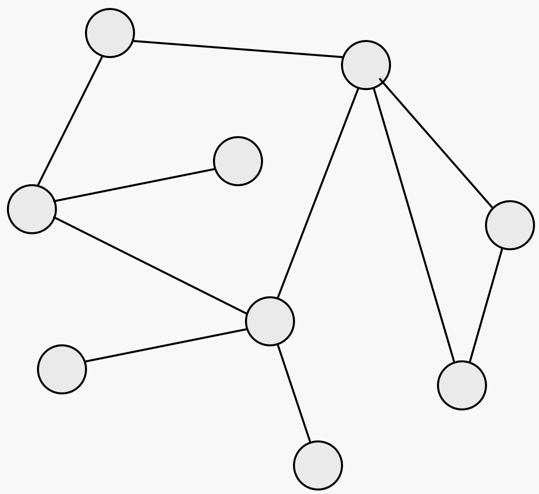
Graphs

A graph G consists of a set V of vertices and a set E of pairs of distinct vertices from V. These pairs of vertices are called edges.

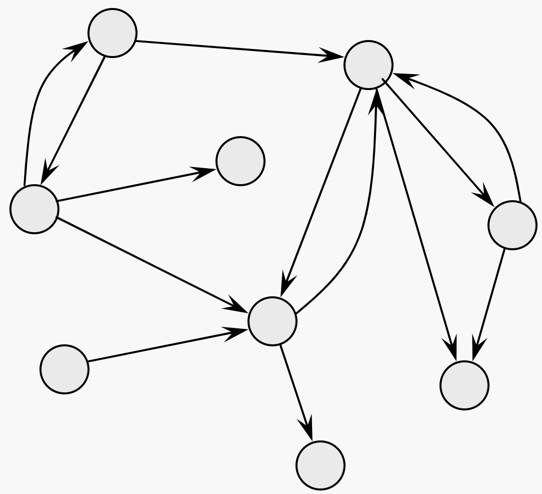
If the pairs of vertices are unordered, G is an undirected graph. If the pairs of vertices are ordered, G is a directed graph or digraph.



A tree is a graph.



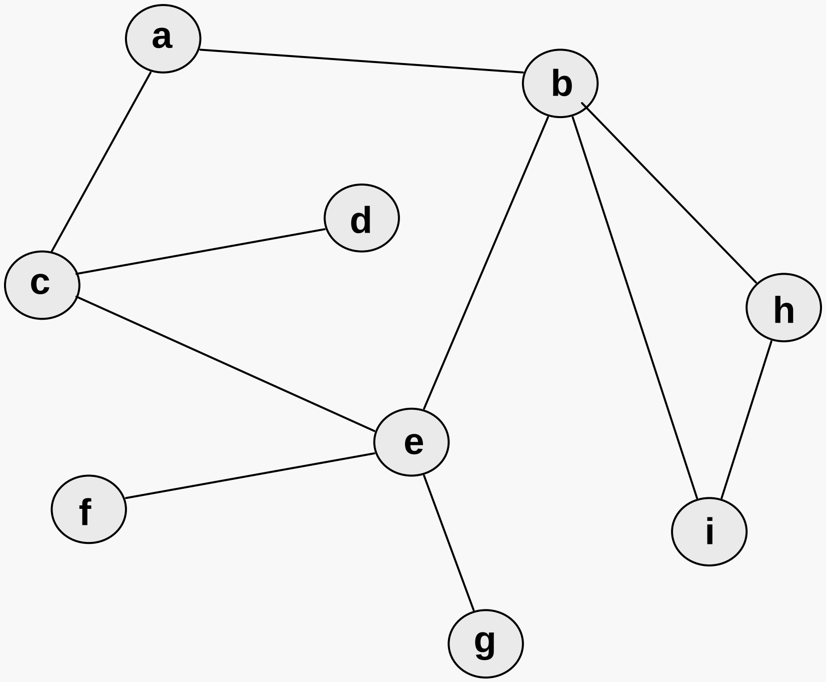
An undirected graph.



A directed graph.

Undirected Graph Terminology

An undirected graph G, where:



V = {a, b, c, d, e, f, g, h, i}

E = { {a, b}, {a, c}, {b, e}, {b, h}, {b, i} ,

{c, d} , {c, e} , {e, f} , {e, g} , {h, i} }

e = {c, d} **is an edge, incident upon the vertices c and d**

Two vertices, x and y, are adjacent if {x, y} is an edge (in E).

A path in G is a sequence of distinct vertices, each adjacent to the next.

A path is simple if no vertex occurs twice in the path.

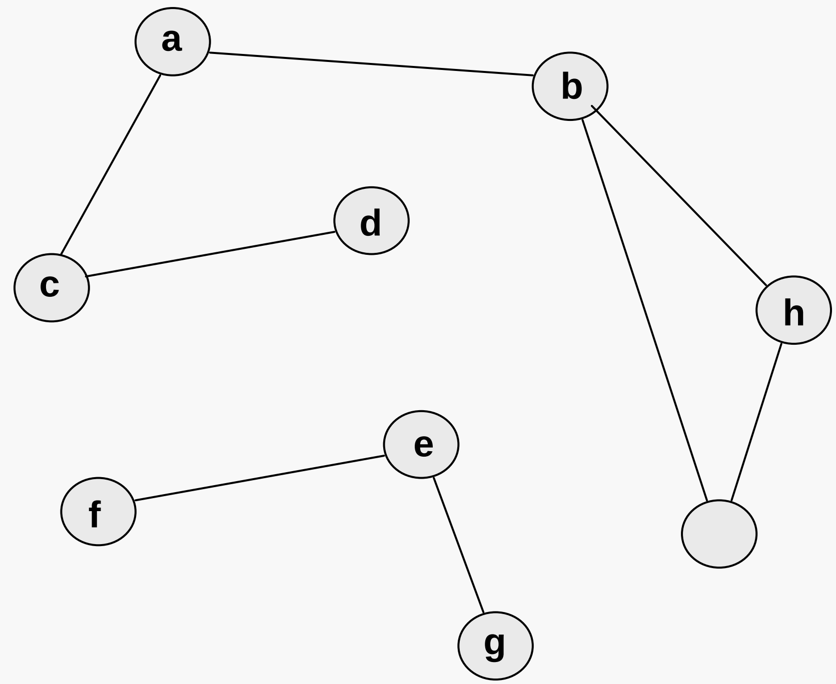
A cycle in G is a path in G, containing at least three vertices, such that the last vertex in the sequence is adjacent to the first vertex in the sequence.

Undirected Graph Terminology

A graph G is <u>connected</u> if, given any two vertices x and y in G, there is a path in G with first vertex x and last vertex y.

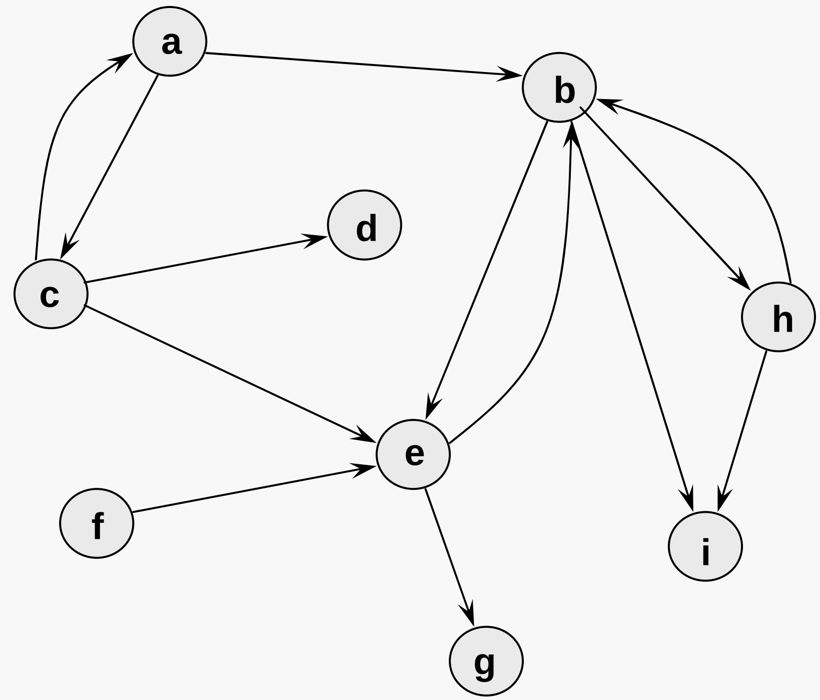
The graph on the previous slide is connected.

If a graph G is not connected, then we say that a maximal connected set of vertices is a <u>component</u> of G.



Directed Graph Terminology

The terminology for directed graphs is only slightly different…



e = (c, d) is an edge, <u>from</u> c <u>to</u> d

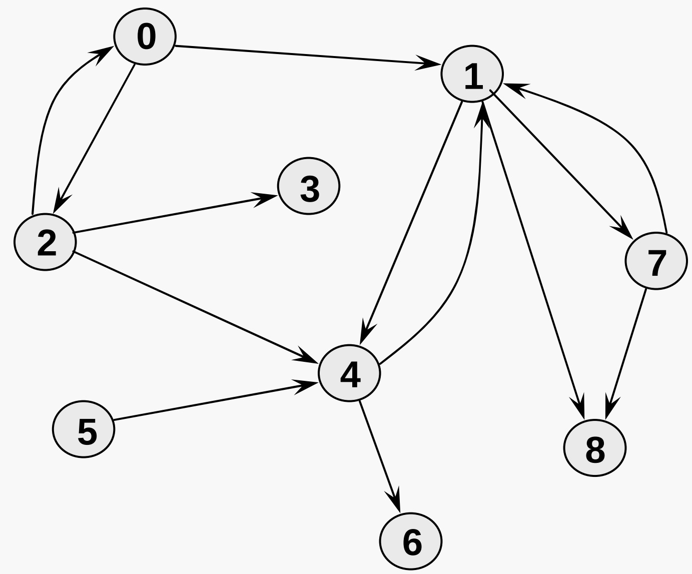
A <u>directed path</u> in a directed graph G is a sequence of distinct vertices, such that there is an edge from each vertex in the sequence to the next.

A directed graph G is <u>weakly connected</u> if, the undirected graph obtained by suppressing the directions on the edges of G is connected (according to the previous definition).

A directed graph G is <u>strongly connected</u> if, given any two vertices x and y in G, there is a directed path in G from x to y.

Adjacency Matrix Representation

A graph may be represented by a two-dimensional adjacency matrix:



If G has n = |V| vertices, let M be an n by n matrix whose entries are defined by

$$m_{ij} = \begin{cases} 1 & \text{if (i, j) is an edge} \\ 0 & \text{otherwise} \end{cases}$$

$$M(G) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Adjacency Matrix Representation

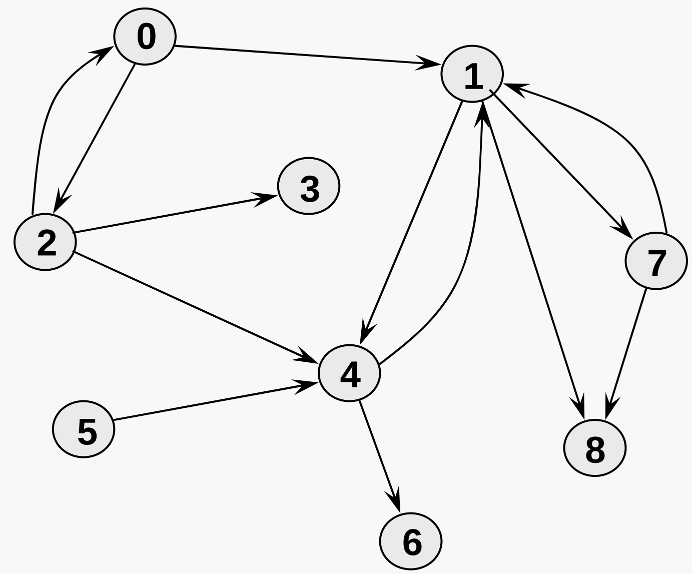
The adjacency table:

- $\Theta(1)$ to determine existence of a specific edge
- $\Theta(|V|^2)$ storage cost (cut cost by 75% or more by changing types)
- $\Theta(|V|)$ for finding all vertices accessible from a specific vertex
- $\Theta(1)$ to add or delete an edge
- Not easy to add or delete a vertex; better for static graph structure.
- Symmetric matrix for undirected graph; so half is redundant then.

$$M(G) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Adjacency Table Representation

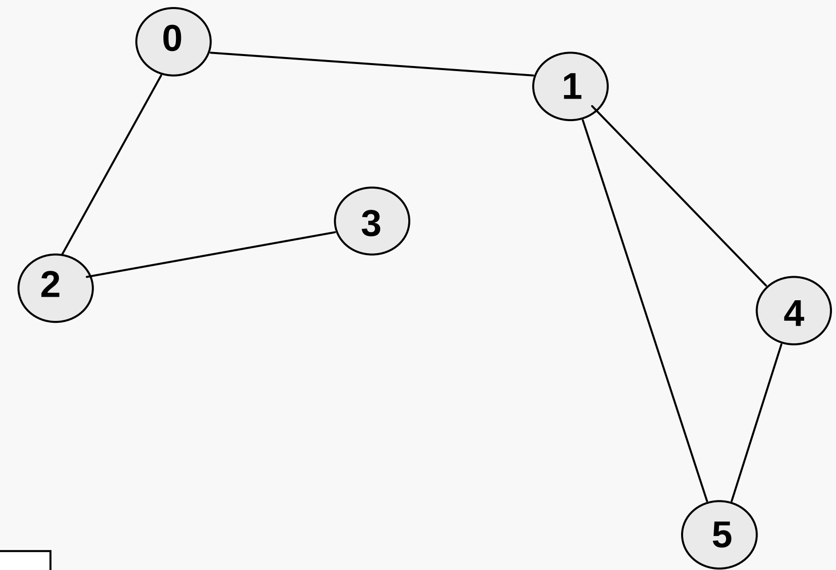
A slightly different approach is to represent only the adjacent nodes in the structure:

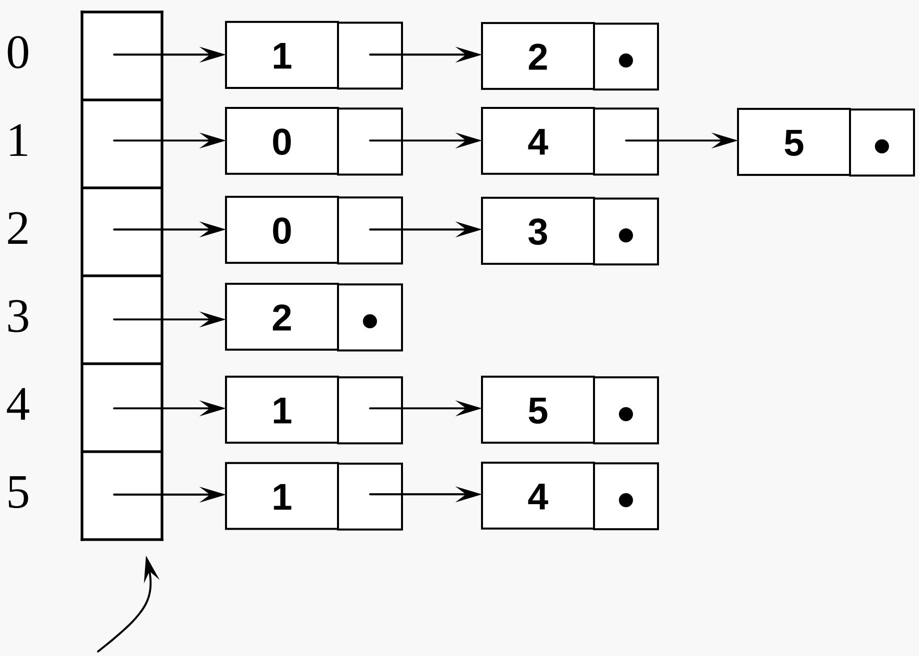



```
0 |  1  2
1 |  4  7  8
2 |  0  3  4
3 |
4 |  1  6
5 |  4
6 |
7 |  1  8
8 |
```

Adjacency List Representation

The adjacency list structure is simply a linked version of the adjacency table:





Array of linked lists, where list nodes store node labels for neighbors.

Adjacency List Representation

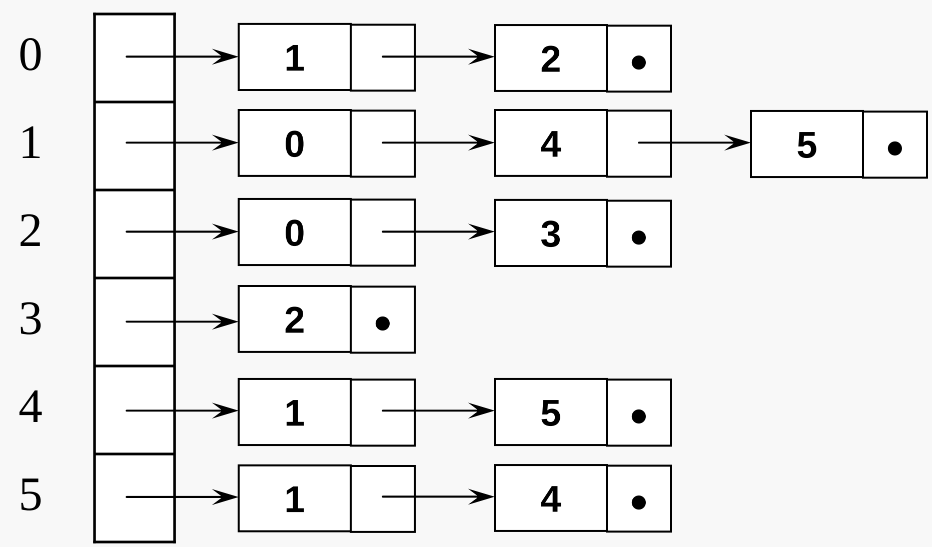
The adjacency list structure:

- Worst case: Θ(|V|) to determine existence of a specific edge
- Θ(|V| + |E|) storage cost
- Worst case: Θ(|V|) for finding all neighbors of a specific vertex
- Worst case: Θ(|V|) to add or delete an edge
- Still not easy to add or delete a vertex; however, we can use a linked list in place of the array.

Note, for an undirected graph, the upper bound on the number of edges is:

$$|E| \leq |V|*(|V|-1)$$

So, the space comparison with the adjacency matrix scheme is not trivial.



An Adjacency Matrix Class


```
public class AdjMatrix {

   private int numVertices;
   private boolean[] Marker;  // used for vertex marking
   private int[][] Edge;      // Edge[i][j] == 1 iff (i,j) exists

   public AdjMatrix(int numV) {...}

   public boolean addEdge(int Src, int Trm) {...}
   public boolean delEdge(int Src, int Trm) {...}
   public boolean hasEdge(int Src, int Trm) {...}

   public int firstNeighbor(int Src) {...}
   public int nextNeighbor(int Src, int Prev) {...}

   public boolean isMarked(int V) {...}
   public boolean Mark(int V) {...}
   public boolean unMark(int V) {...}

   public void Clear() {...}    // erase edges and vertex marks
   public void Display() {...}
}
```

`firstNeighbor()` **returns the first vertex adjacent to** `Src`**.**

`nextNeighbor()` **returns the next vertex, after** `Prev`**, which is adjacent to** `Src`**.**