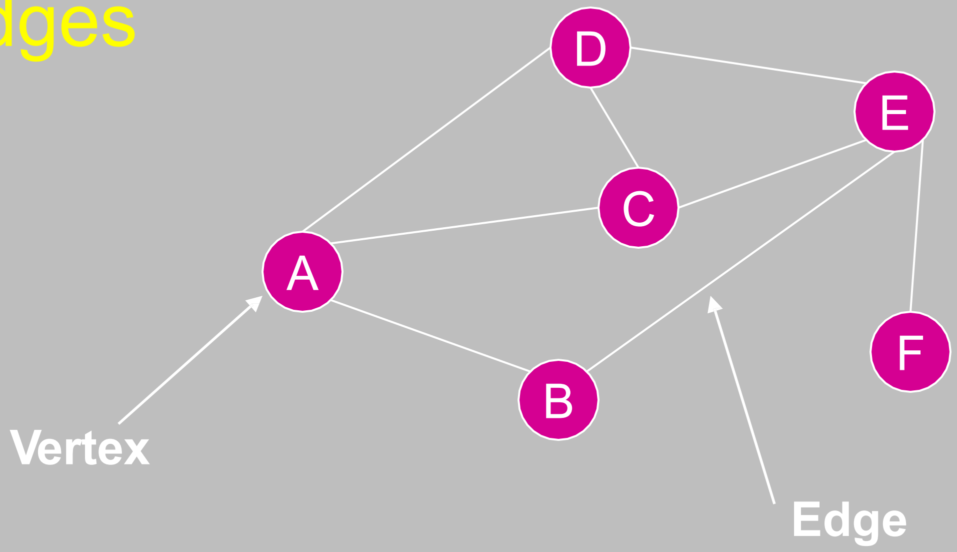
Intro to Graph

Graphs

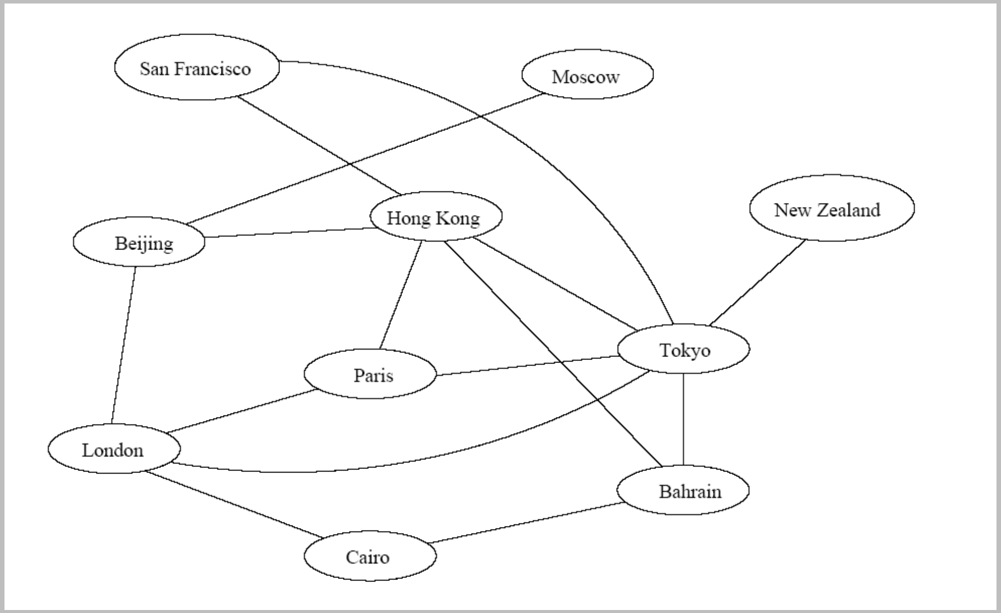
- Extremely useful tool in modeling problems
- Consist of:
  - Vertices
  - Edges



Vertices can be considered "sites" or locations.

Edges represent connections.

Application

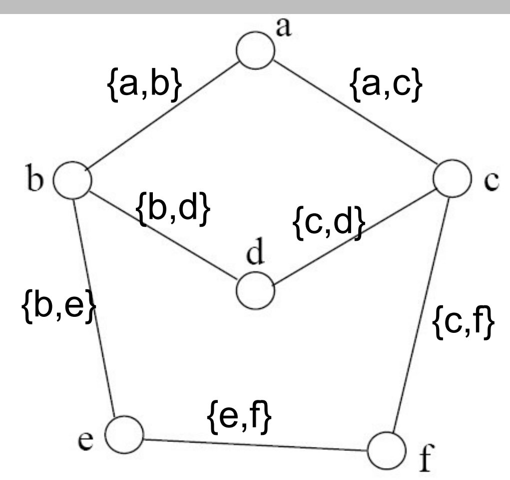


Air flight system

- Each vertex represents a city
- Each edge represents a direct flight between two cities
- A query on direct flights = a query on whether an edge exists
- A query on how to get to a location = does a path exist from A to B
- We can even associate costs to edges (weighted graphs), then ask "what is the cheapest path from A to B"

Definition

- A graph G=(V, E) consists a set of vertices, V, and a set of edges, E.
- Each edge is a pair of *(v, w)*, where v, w belongs to V
- If the pair is unordered, the graph is undirected; otherwise it is directed



$$V = \{a, b, c, d, e, f\}$$

$$E = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, d\}, \{b, e\}, \{c, f\}, \{e, f\}\}$$

An undirected graph

Graph Variations

Variations:

- A *connected graph* has a path from every vertex to every other
- In an *undirected graph:*
  - Edge (u,v) = edge (v,u)
  - No self-loops
- In a *directed* graph:
  - Edge (u,v) goes from vertex u to vertex v, notated $u \rightarrow v$

Graph Variations

- More variations:
  - A *weighted graph* associates weights with either the edges or the vertices
    - E.g., a road map: edges might be weighted w/ distance
  - A *multigraph* allows multiple edges between the same vertices
    - E.g., the call graph in a program (a function can get called from multiple points in another function)

Graphs

- We will typically express running times in terms of |E| and |V| (often dropping the |'s)
  - If $|E| \approx |V|^2$ the graph is *dense*
  - If $|E| \approx |V|$ the graph is *sparse*
- If you know you are dealing with dense or sparse graphs, different data structures may make sense

Path between Vertices

- A path is a sequence of vertices (v_0, v_1, v_2,… v_k) such that:
  - For $0 \leq i < k$, $\{v_i, v_{i+1}\}$ is an edge

Note: a path is allowed to go through the same vertex or the same edge any number of times!

- The length of a path is the number of edges on the path

Types of paths

- A path is simple if and only if it does not contain a vertex more than once.
- A path is a cycle if and only if $v_0 = v_k$
  - The beginning and end are the same vertex!
- A path contains a cycle as its sub-path if some vertex appears twice or more

Example-1

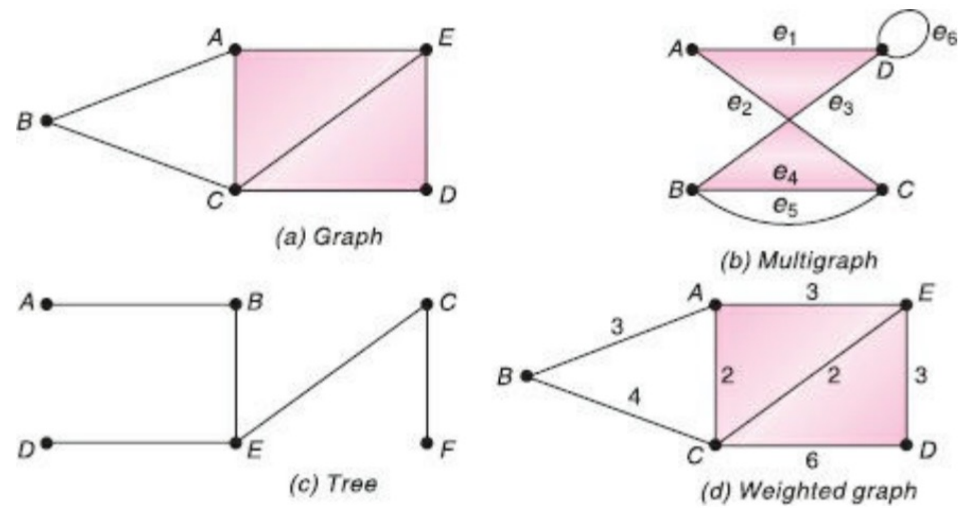


Fig. 8.1

- Figure 8.1(a) is a picture of a connected graph with 5 nodes—A, B, C, D and E— and 7 edges: [A, B], [B, C], [C, D], [D, E], [A, E], [C, E] [A, C]
- There are two simple paths of length 2 from B to E: (B, A, E) and (B, C, E). There is only one simple path of length 2 from B to D: (B, C, D). We note that (B, A, D) is not a path, since [A, D) is not an edge.
- There are two 4-cycles in the graph:[A, B, C, E, A] and [A, C, D, E, A].
- Note that deg(A) = 3, since A belongs to 3 edges. Similarly, deg(C) = 4 and deg(D) = 2.

Example-2

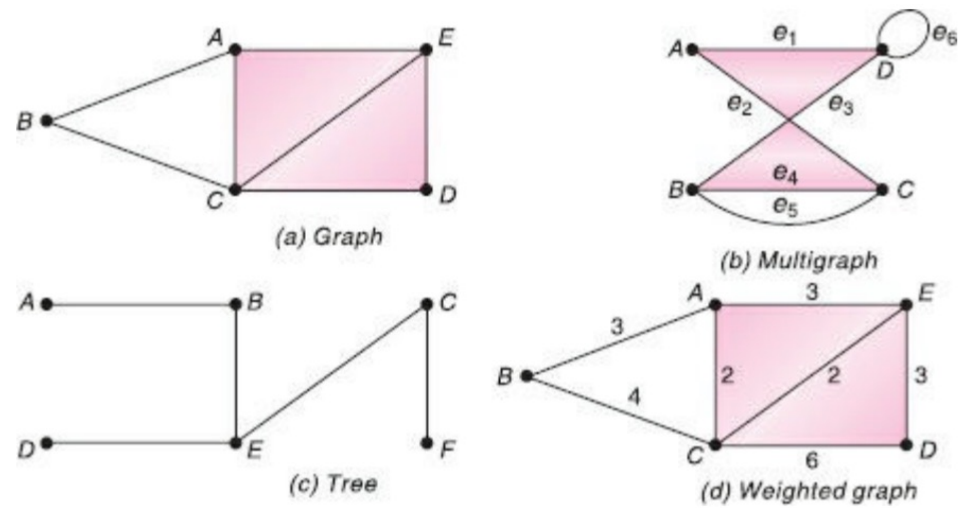


Fig. 8.1

- Figure 8.1(b) is not a graph but a multigraph. The reason is that it has multiple edges—e4 = [B, C] and e5= [B, C]—and it has a loop, e6= [D, D).
- The definition of a graph usually does not allow either multiple edges or loops.
- Figure 8.1(c) is a tree graph with m = 6 nodes and, consequently, m – 1 = 5 edges. There is a unique simple path between any two nodes of the tree graph.

Example-3

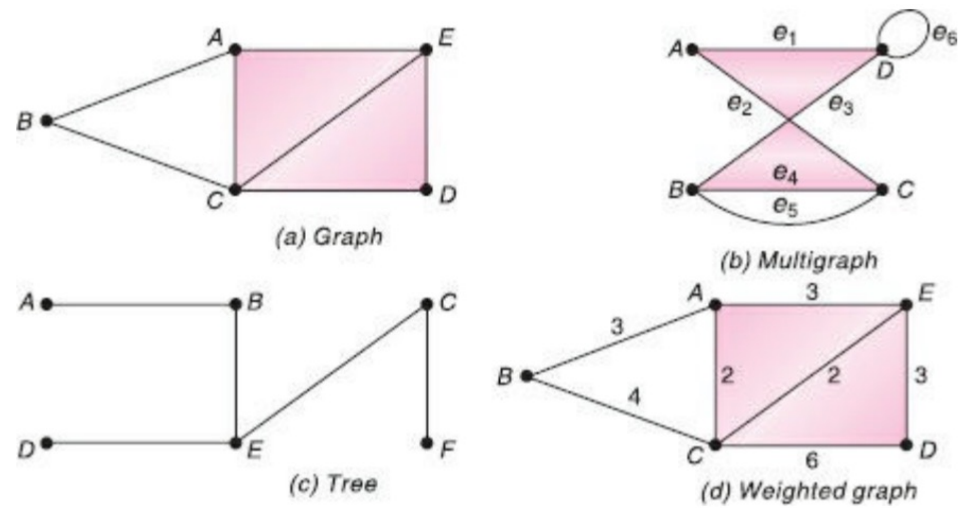


Fig. 8.1

- Figure 8.1(d) is the same graph as in Fig. 8.1(a), except that now the graph is weighted.
- Observe that P1 = (B, C, D) and P2= (B, A, E, D) are both paths from node B to node D.
- Although P2 contains more edges than P1 the weight w(P2) = 9 is less than the weight w(P1) = 10.

Example-4

Consider the (undirected) graph G in Fig. 8.19. (a) Describe G formally in terms of its set V of nodes and its set E of edges. (b) Find the degree of each node.

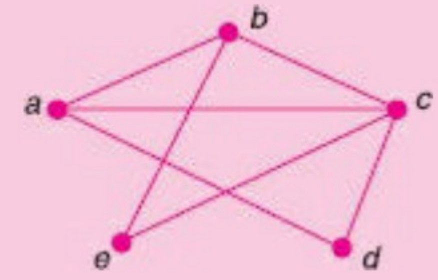


Fig. 8.19

(a) There are 5 nodes, a, b, c, d and e; hence $V = \{a, b, c, d, e\}$. There are 7 pairs [x, y] of nodes such that node x is connected with node y; hence

$$E = \{[a, b], [a, c], [a, d], [b, c], [b, e], [c, d], [c, e]\}$$

(b) The degree of a node is equal to the number of edges to which it belongs; for example, deg(a) = 3, since a belongs to three edges, $[a, b]$, $[a, c]$ and $[a, d]$. Similarly, deg(b) = 3, deg(c) = 4, deg(d) = 2 and deg(e) = 2.

Example-5

Consider the multigraphs in Fig. 8.20. Which of them are (a) connected; (b) loop-free (i.e., without loops); (c) graphs?

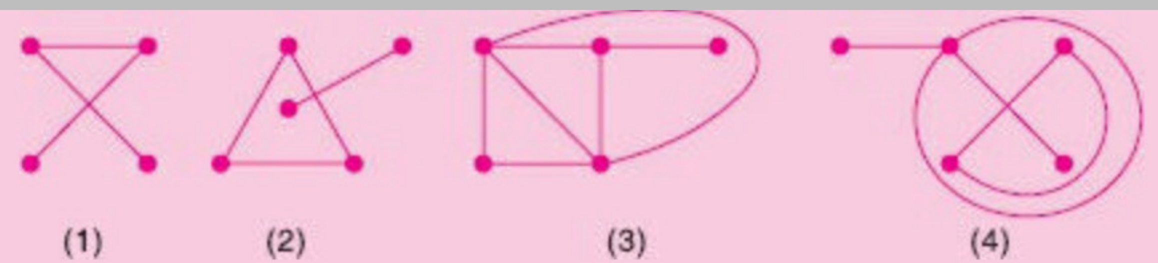


Fig. 8.20

(a) Only multigraphs 1 and 3 are connected.

(b) Only multigraph 4 has a loop (i.e., an edge with the same endpoints).

(c) Only multigraphs 1 and 2 are graphs. Multigraph 3 has multiple edges, and multigraph 4 has multiple edges and a loop.

Example-6

Consider the connected graph G in Fig. 8.21. (a) Find all simple paths from node A to node F. (b) Find the distance between A and F. (c) Find the diameter of G. (The diameter of G is the maximum distance existing between any two of its nodes.)

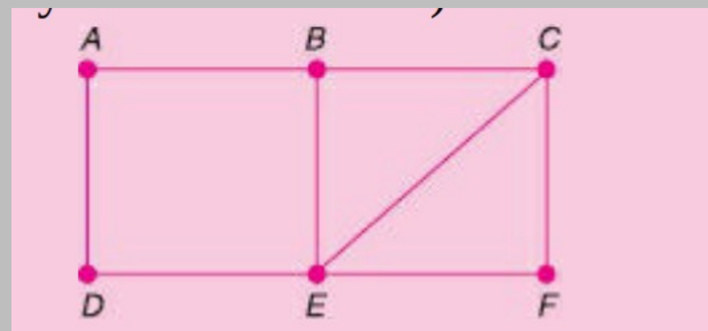


Fig. 8.21

(a) A simple path from A to F is a path such that no node and hence no edge is repeated. There are seven such simple paths:

(A, B, C, F) (A, B, E, F) (A, D, E, F) (A, D, E, C, F)

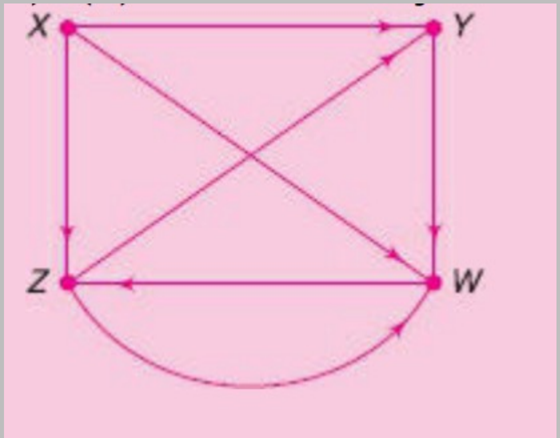
(A, B, C, E, F) (A, B, E, C, F) (A, D, E, B, C, F)

(b) The distance from A to F equals 3, since there is a simple path, (A, B, C, F), from A to F of length 3 and there is no shorter path from A to F.

(c) The distance between A and F equals 3, and the distance between any two nodes does not exceed 3; hence the diameter of the graph G equals 3.

Example-7

Consider the (directed) graph G in Fig. 8.22. (a) Find all the simple paths from X to Z. (b) Find all the simple paths from Y to Z. (c) Find indeg(Y) and outdeg(Y). (d) Are there any sources or sinks?



(a) There are three simple paths from X to Z: (X, Z), (X, W, Z) *and* (X, Y, W, Z).

(b) There is only one simple path from Y to Z: (Y, W, Z).

(c) Since two edges enter Y (i.e., end at Y), we have indeg(Y) = 2. Since only one edge leaves Y (i.e., begins at Y), outdeg(Y) = 1.

(d) X is a source, since no edge enters X (i.e., indeg(X) = 0) but some edges leave X (i. e., outdeg(X) > 0). There are no sinks, since each node has a nonzero outdegree (i.e., each node is the initial point of some edge).

Graph Representation

- Two popular computer representations of a graph. Both represent the vertex set and the edge set, but in different ways.

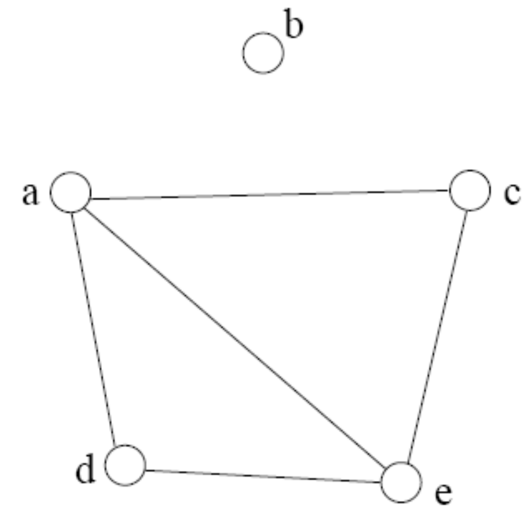
1. Adjacency Matrix

   Use a 2D matrix to represent the graph

2. Adjacency List

   Use a 1D array of linked lists

Adjacency Matrix



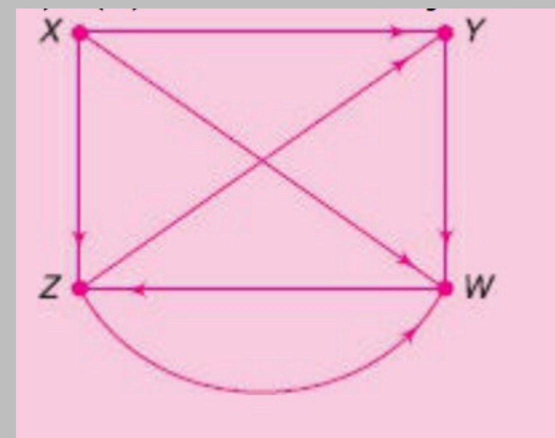
	a	b	c	d	e
a	0	0	1	1	1
b	0	0	0	0	0
c	1	0	0	0	1
d	1	0	0	0	1
e	1	0	1	1	0

- 2D array A[0..n-1, 0..n-1], where ***n*** is the number of vertices in the graph
- Each row and column is indexed by the vertex id
  - e,g a=0, b=1, c=2, d=3, e=4
- *A[i][j]=1* if there is an edge connecting vertices *i* and *j*; otherwise, *A[i][j]=0*
- The storage requirement is $\Theta(n^2)$. It is not efficient if the graph has few edges. An adjacency matrix is an appropriate representation if the graph is dense: $|E|=\Theta(|V|^2)$
- We can detect in O(1) time whether two vertices are connected.

Example

Consider the graph G in Fig. 8.22. Suppose the nodes are stored in memory in an array DATA as follows: DATA: X, Y, Z, W

(a) Find the adjacency matrix A of the graph G.

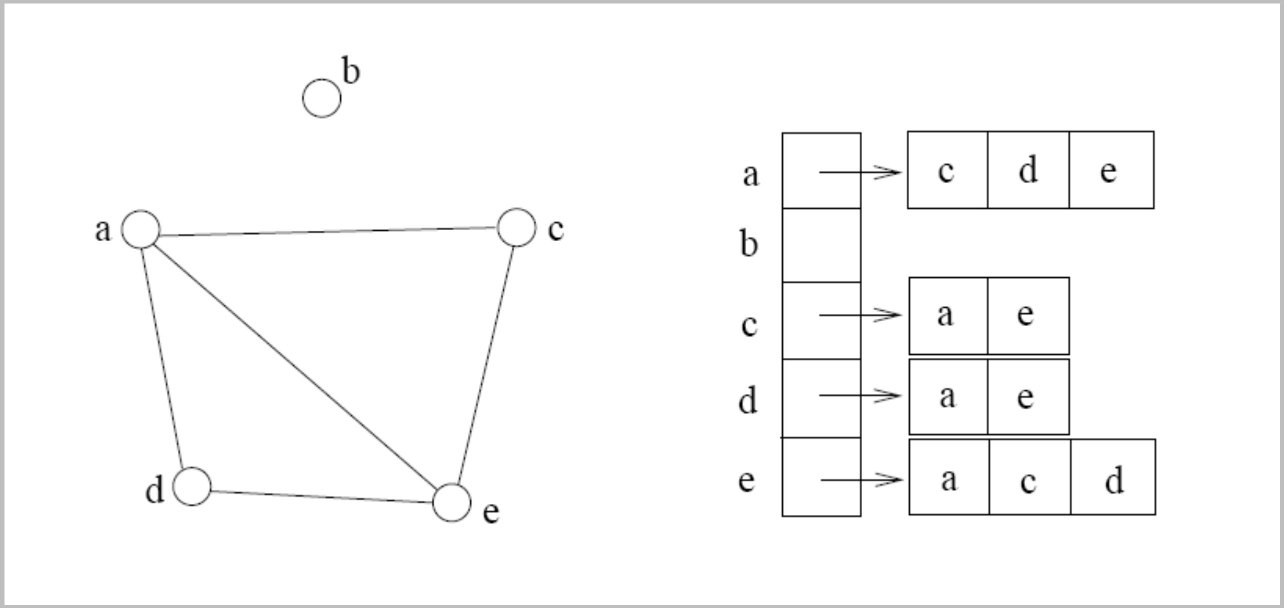


(a) The nodes are normally ordered according to the way they appear in memory; that is, we assume v_1 = X, v_2 = Y v_3 = Z and v_4 = W. The adjacency matrix A of G follows:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

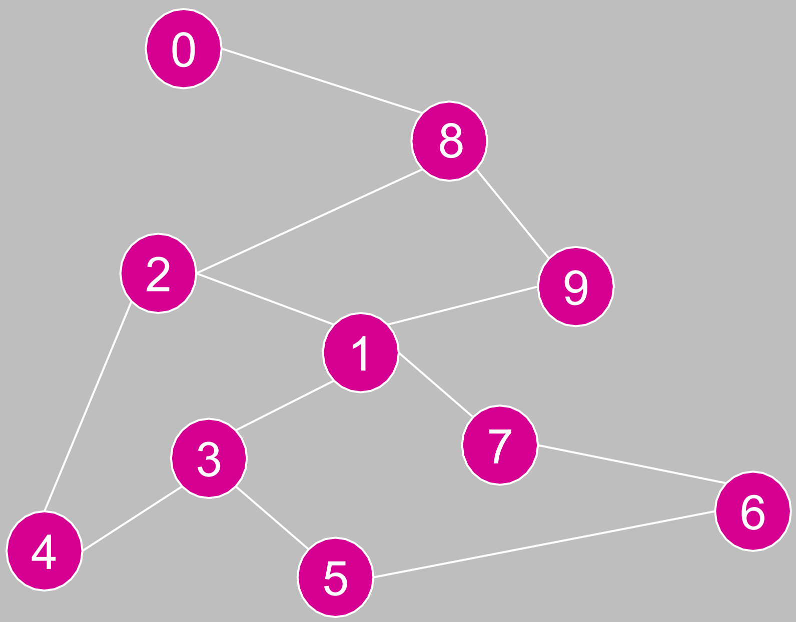
Here $a_{ij} = 1$ if there is a node from v_i to v_j otherwise, $a_{ij} = 0$.

Adjacency List



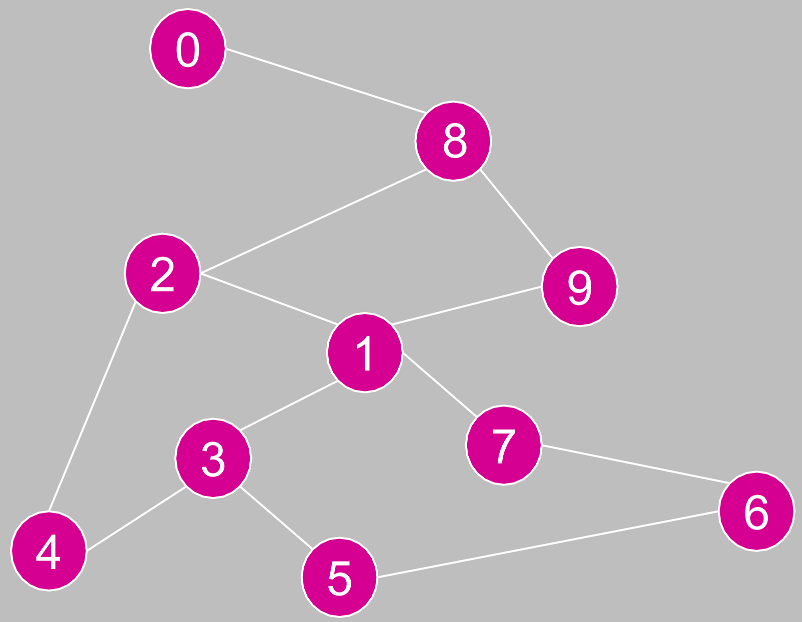
- If the graph is not dense, in other words, sparse, a better solution is an adjacency list
- The adjacency list is an array A[0..n-1] of lists, where n is the number of vertices in the graph.
- Each array entry is indexed by the vertex id
- Each list *A[i]* stores the ids of the vertices adjacent to vertex *i*

Adjacency Matrix Example



	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0	1	0	1
2	0	1	0	0	1	0	0	0	1	0
3	0	1	0	0	1	1	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0
5	0	0	0	1	0	0	1	0	0	0
6	0	0	0	0	0	1	0	1	0	0
7	0	1	0	0	0	0	1	0	0	0
8	1	0	1	0	0	0	0	0	0	1
9	0	1	0	0	0	0	0	0	1	0

Adjacency List Example



0	→	8			
1	→	2	3	7	9
2	→	1	4	8	
3	→	1	4	5	
4	→	2	3		
5	→	3	6		
6	→	5	7		
7	→	1	6		
8	→	0	2	9	
9	→	1	8		

Example

Consider the graph G in Fig. 8.14(a). (The adjacency lists of the nodes appear in Fig. 8.14(b).) Suppose G represents the daily flights between cities of some airline, and suppose we want to fly from city A to city J with the minimum number of stops. In other words, we want the minimum path P from A to J (where each edge has length 1).

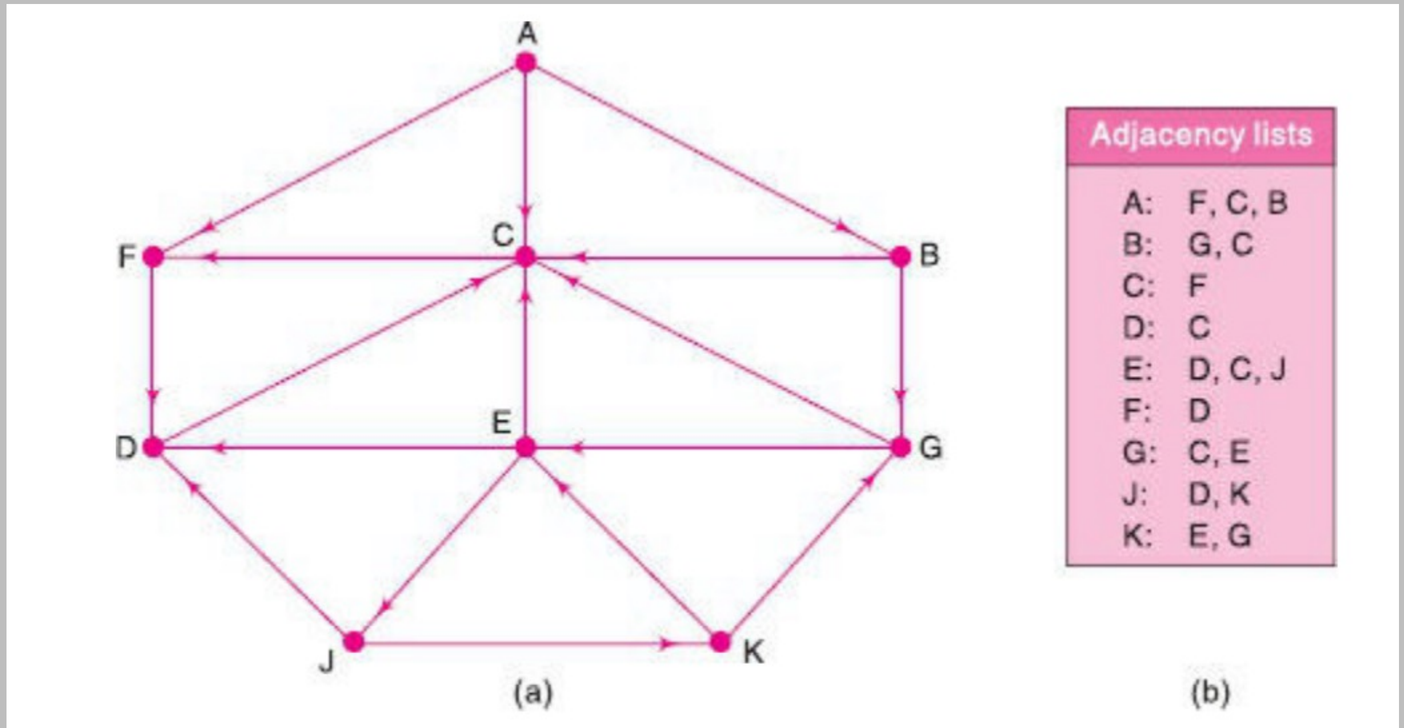


Fig. 8.14