

Elementary Graph Algorithms

Graphs: a graph is a collection of vertices (at least one) and edges.

- **Vertices:** points or dots called vertices (singular is vertex)
- **Edges:** lines connecting vertices called edges
- **Adjacent:** If there is an edge between vertices A and B, then A & B are adjacent.

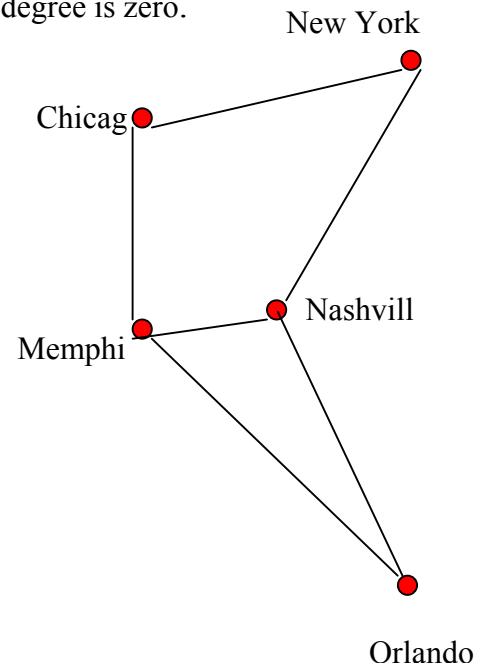
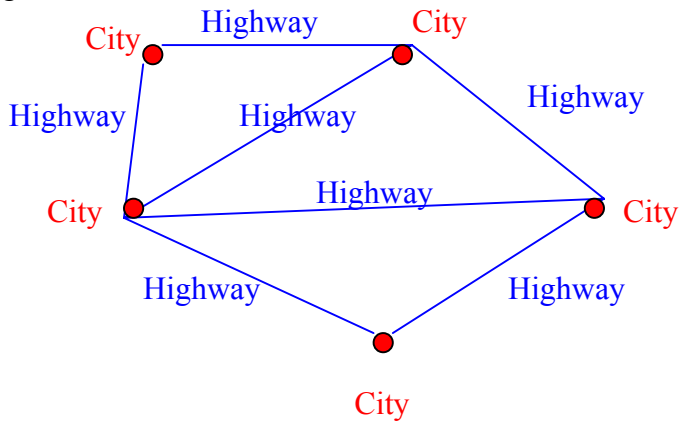
Simple Graphs: Graphs in which there is no more than one edge between any two vertices

Degree of a vertex: the number of edges joined to that vertex.

If a vertex has no edges, its degree is zero.

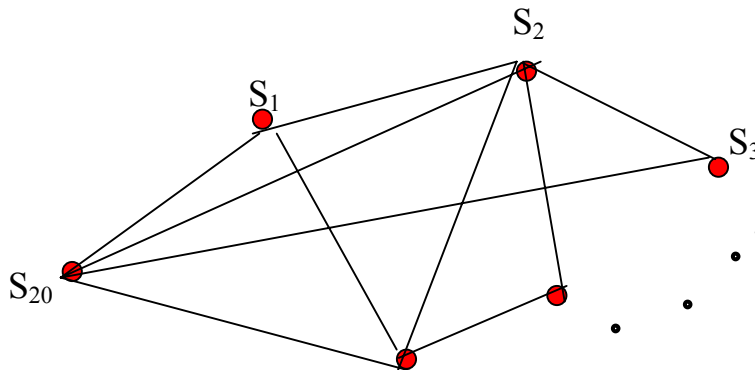
Examples of graphs:

e.g. 1:



e.g. 2: Computer Network.

e.g. 3 Students know each other.



- **Directed Graphs:** e.g. S_{20} know S_2 , but S_2 does not know S_{20} .
e.g. Road in one-way.
- **Undirected Graphs.**
- **Weighted Graphs:**
 - Weighted on **edges:** e.g. numbers of traffic per hour
e.g. 80 miles/hour, 120 miles/hours on highway
 - Weighted on **vertices:** e.g. population of city.
- Two ways to represent a graph:
 - **Adjacent List:** List all adjacent vertices in the queue. (There is no exact order.)
 - **Adjacent Matrix:** If there is an edge, then enter 1, otherwise enter 0

Undirected Graphs: (See your textbook as following)

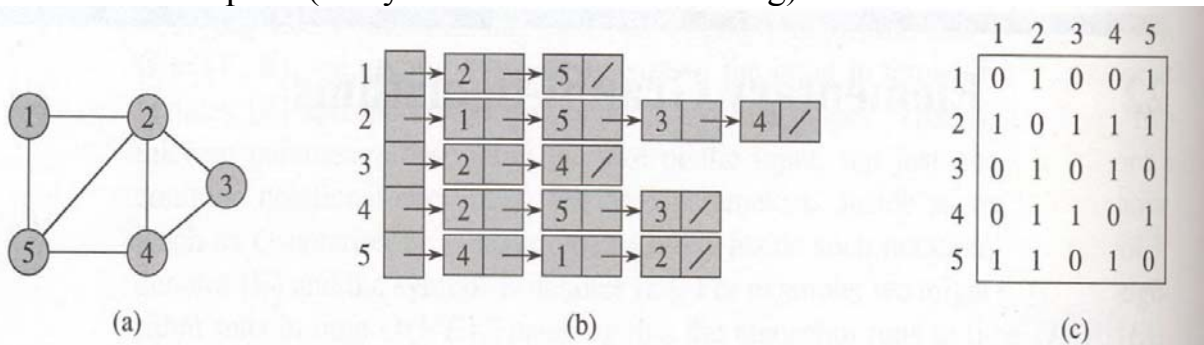


Figure 22.1 Two representations of an undirected graph. (a) An undirected graph G having five vertices and seven edges. (b) An adjacency-list representation of G . (c) The adjacency-matrix representation of G .

Directed Graphs: (See your textbook as following)

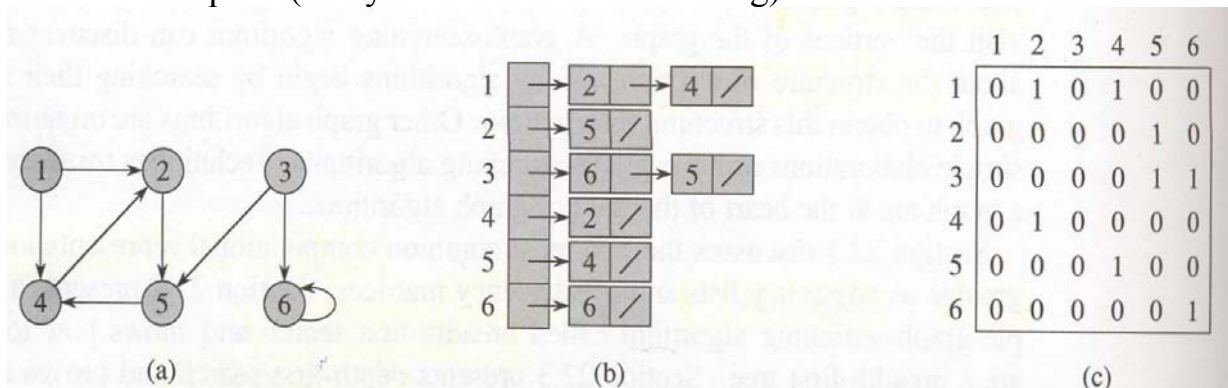


Figure 22.2 Two representations of a directed graph. (a) A directed graph G having six vertices and eight edges. (b) An adjacency-list representation of G . (c) The adjacency-matrix representation of G .

How many memory space using: (Assume n vertices, m edges)

- Adjacent Matrix: $n^2 = O(n^2)$
- Adjacent List: $n + 2m = O(n + m)$
- Adjacent List is better than Adjacent Matrix if compare the using spaces.

Searching a graph:

Visit every vertex and edge of the graph in systematic manner (Do not miss any of vertex and edge).

Two important graph exploration techniques are

- **breadth-first search:** like breadth-first search in a tree, we search as broadly as possible by visiting a node, and then immediately visiting all nodes adjacent to that node.
- **depth-first search:** like depth-first search in a tree, we search as deeply as possible by visiting a node, and then recursively performing depth-first search on each adjacent node.