Lecture 15.2

Topics:

1. Graph – Introduction

____

1. Graph – Introduction

1.1 Definition

In general, a graph is a collection of vertices or nodes, where pairs of which are joined by lines or edges. Formally, a graph `G = (V, E)` is an ordered pair of finite sets `V` and `E`.

The elements of `V` are called vertices (also called nodes or points). The elements of `E` are called edges (also called lines or arcs). Each edge in `E` joins two different vertices of `V` and is denoted as the tuple `(i, j),` where `i` and `j` are two vertices joined by `E`.

A graph can be displayed with vertices represented as circles and edges by lines. If the edges are oriented then they are called **directed edges** (thus, a **directed graphed** or **digraph**). If the edges are not directed then the graph is called an **undirected graph**. In an undirected graph, the edges `(i, j)` and `(j, i)` are the same, while the directed graph does differentiate the pair of `(i, j)` and `(j, i)` as different. **Figure 1** depicts the graphs.

In a graph, vertices `i` and `j` are adjacent vertices if and only if `(i, j)` is an edge in the graph. The edge `(i, j)` is incident on the vertices `i` and `j`. For a digraph, the directed edge `(i, j)` is incident to vertex `j` and incident from vertex `i`. Vertex `i` is adjacent to vertex `j`, and vertex `j` is adjacent from vertex `i`. For an undirected graph, the "to" and "from" are synonymous.

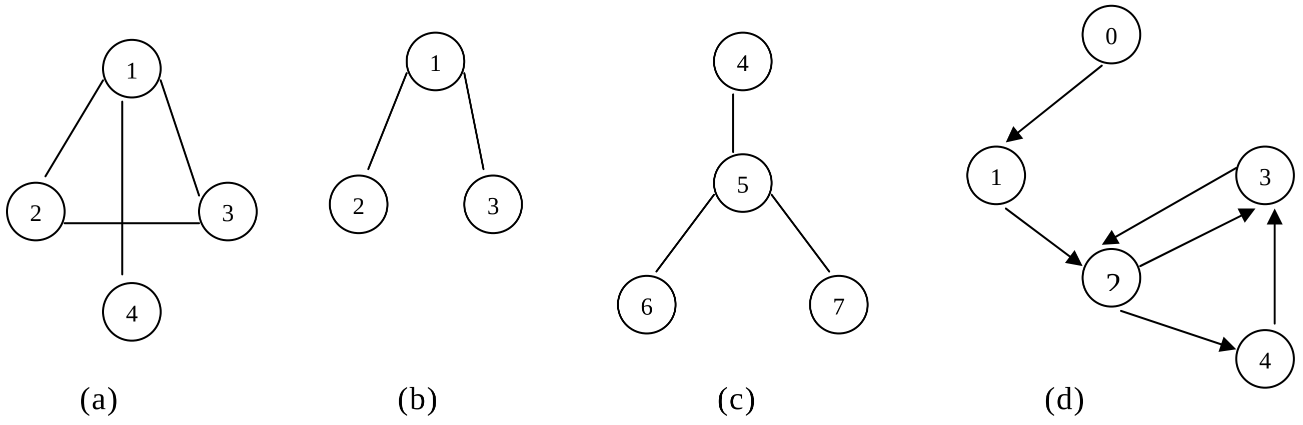

Figure 1 Graphs

Using the set notation, the graphs of **Figure 1** may be specified as `G1 = (V1, E1), G2 = (V2, E2),` and `G3 = (V3, E3)` where,

```
V1 = {1, 2, 3, 4}
E1 = { (1, 2), (1, 3), (2, 3), (1, 4), (3, 4) }

V2 = { 1, 2, 3, 4, 5, 6, 7}
E2 = { (1, 2), (1, 3), (4, 5), (5, 6), (5, 7), (6, 7) }

V3 = { 0, 1, 2, 3, 4}
E3 = { (0, 1), (1, 2), (2, 3), (3, 2), (2, 4), (4, 3) }
```

By definition, a graph does not contain multiple copies of the same edge. Therefore, an undirected graph can have at most one edge between any pair of vertices, and a directed graph can have at most one edge from vertex **i** to vertex **j** and one edge from vertex **j** to vertex **i**. A graph cannot have any self-edges, which are the edges from (i, i) are not allowed (a self-edge is also called a loop).

Weights can be assigned to edges. They represent the cost or priority within a scheme. These weighted graphs do represent the real applications in electrical engineering (linear and nonlinear systems), molecular representation, traffic networks (such as city streets, air routes, etc), and many others.

A graph can be represented in several ways. The two common ways are called the adjacency matrices and adjacency lists. The discussion using matrices would require mathematical explanation and will not be presented here. The adjacency lists are given here for directed graphs (or digraphs).

1.2 Digraphs – Adjacency Lists

Let **G** be a digraph with **n** vertices, where $n > 0$. Let $V(G) = \{ v_1, v_2, v_3, v_n \}$. In the adjacency list representation, corresponding to each vertex, v, there is a linked list such that each node of the linked list contains the vertex, u, such that $(v, u) \varepsilon E(G)$.

Since there are **n** nodes, one can use an array **A** of size v such that A[i] is a pointer to the linked list containing the vertices to which $\mathbf{v_i}$ is adjacent. Each node in this linked list has the data value which is the vertex information or the index of the vertex adjacent to vertex **i**.

1.3 Examples

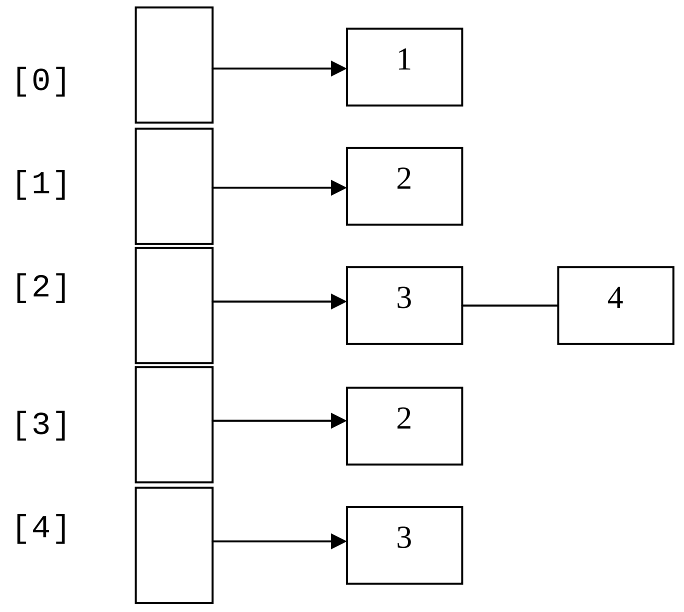

Figure 2 Adjacency list of graph from Figure 1(d)

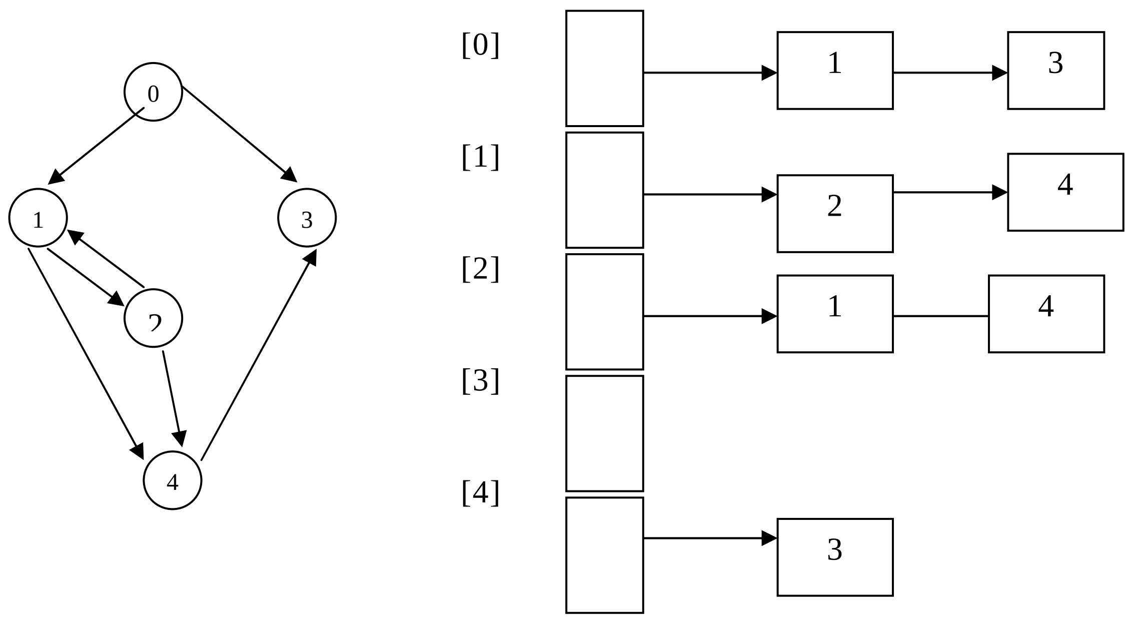

Figure 3 Digraph and its adjacency list

The operations commonly performed on a graph would include creating graph, clearing graph, printing graph, and traversing graph. Specific applications will have additional operations to handle the required tasks.