UCS1302:
DATA STRUCTURES

Graphs

Session Meta Data

Author	**Dr. B. Bharathi**
Reviewer	
Version Number	1.2
Release Date	20 August 2019

Revision History

Revision Date	Details	Version no.
22 September 2017	1. New SSN template applied	1.2

Session Objectives

- To learn about graph and its representations

Session Outcomes

- At the end of this session, participants will be able to
  - Understand graph terminologies
  - Represent the graphs using different methods

Agenda

- Graph introduction
- Terminologies
- Representation of graphs

Graphs

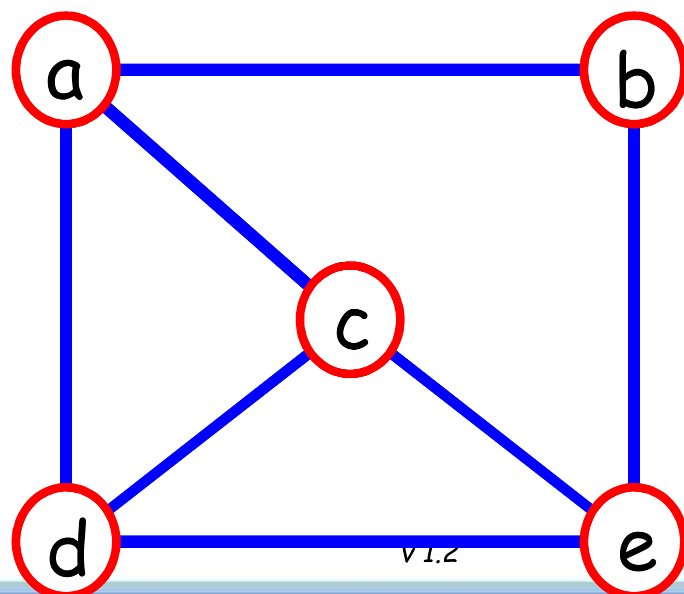
Dr. B. Bharathi
SSNCE

August 20, 2019

What is a Graph?

- A graph G = (V,E) is composed of:
  - V: set of vertices
  - E: set of edges or arcs connecting the vertices in V
- An edge e = (u,v) is a pair of vertices belonging to E
- weight or cost

Example:

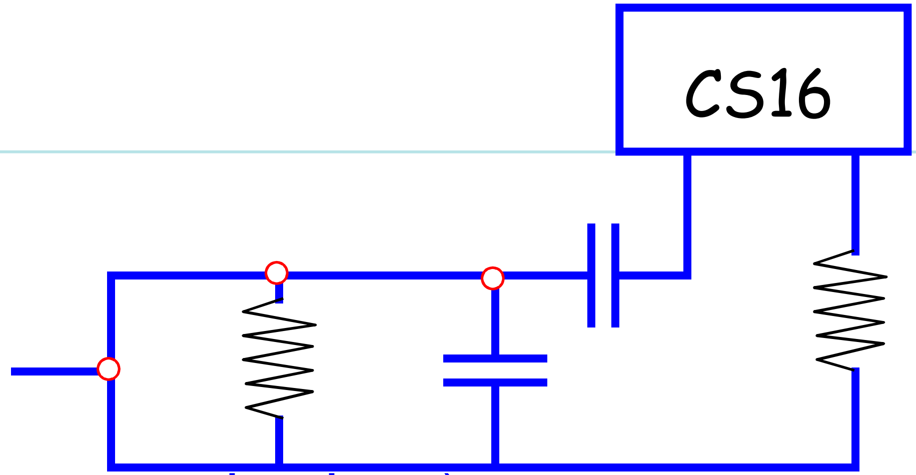


V= {a,b,c,d,e}
E= {(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

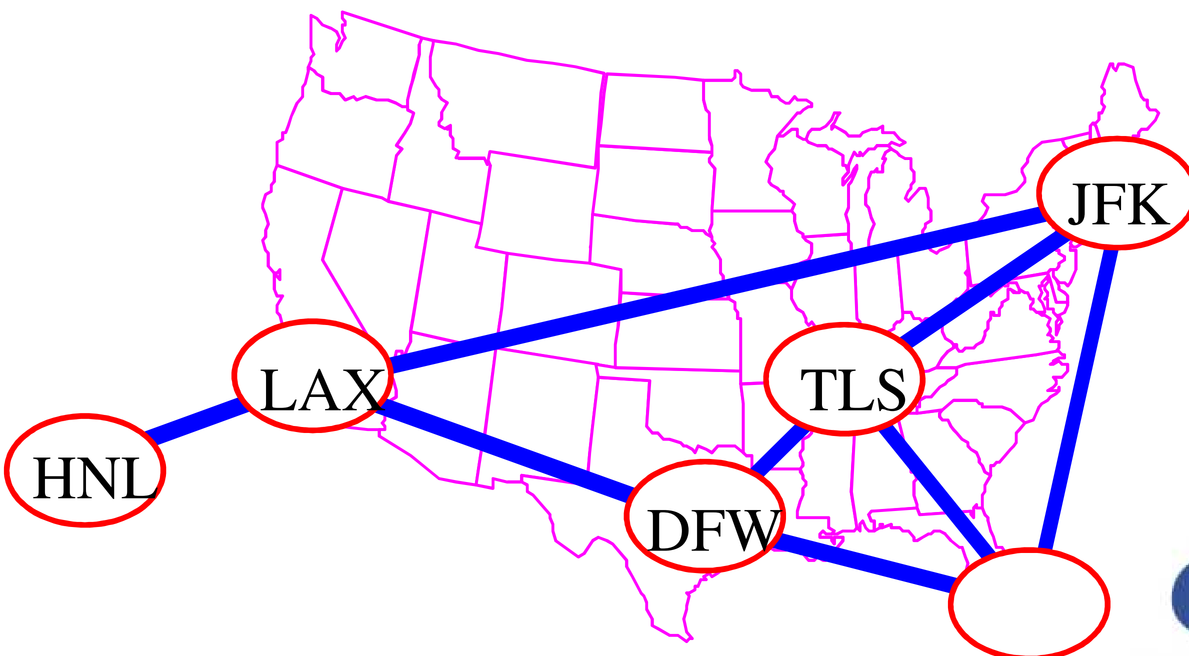
Directed vs. Undirected Graph

- An undirected graph is one in which the pair of vertices in a edge is unordered, $(v_0, v_1) = (v_1,v_0)$
- A directed graph is one in which each edge is a directed pair of vertices, $<v_0, v_1> != <v_1,v_0>$

Applications



- electronic circuits
- networks (roads, flights, communications)

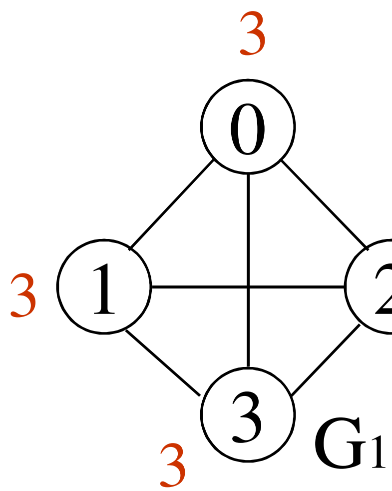


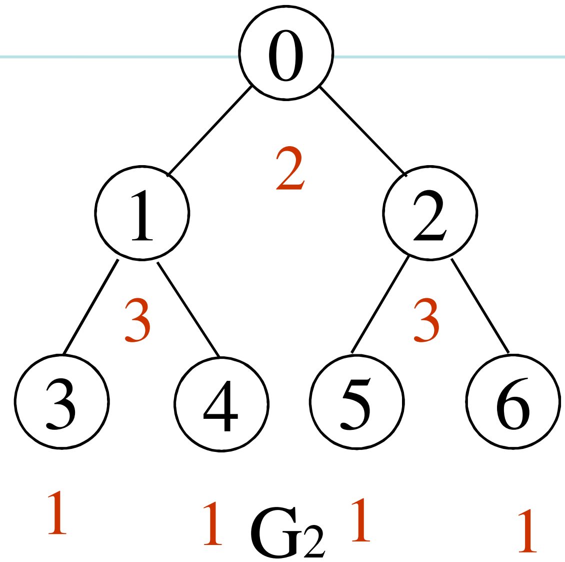
Terminology: Degree of a Vertex

- The degree of a vertex is the number of edges incident to that vertex
- For directed graph,
  - the in-degree of a vertex *v* is the number of edges that have *v* as the head
  - the out-degree of a vertex *v* is the number of edges that have *v* as the tail
  - if *di* is the degree of a vertex *i* in a graph *G* with *n* vertices and *e* edges, the number of edges is

$$e = \left(\sum_{0}^{n-1} d_i\right) / 2$$

Examples

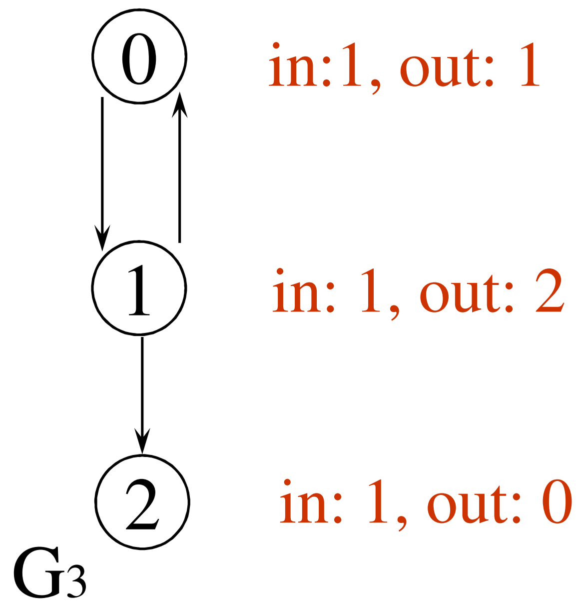




directed graph

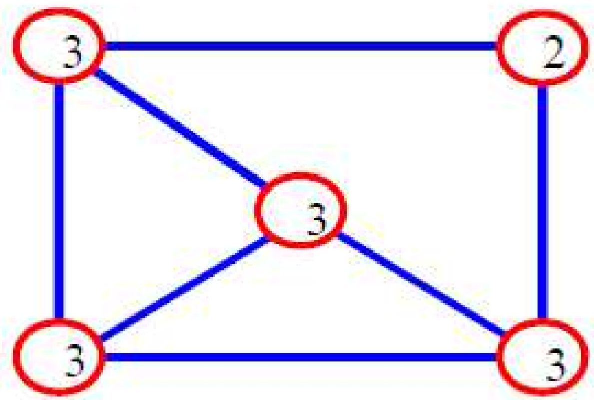
in-degree

out-degree

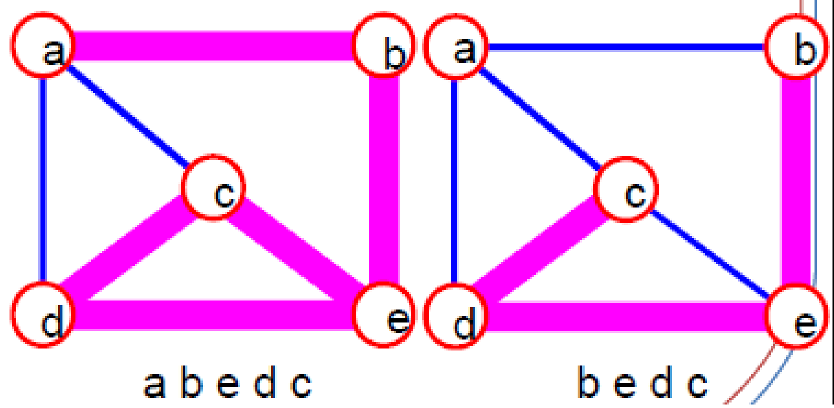


Path

- path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.

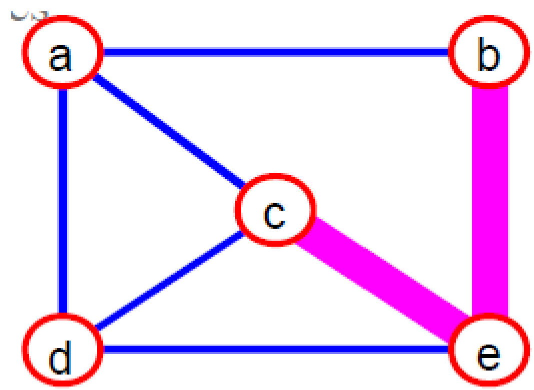


The **length** of the path is the number of edges along the path



Terminology

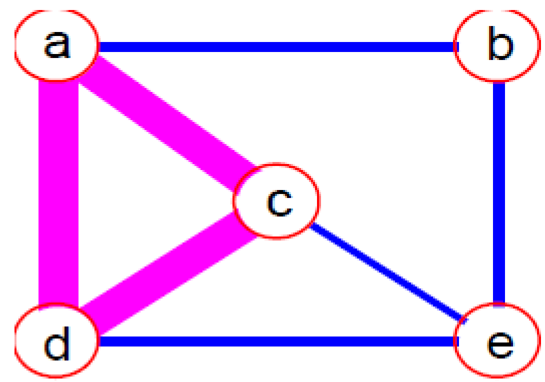
Simple path: No repeated vertices



b e c

Cycle: simple path, except that the last vertex is the same as the first vertex

a c d a



Terminology

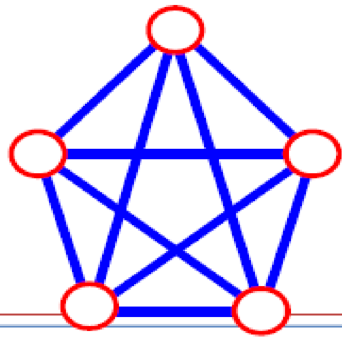
- A directed graph that has *no* cyclic paths is called a **DAG** (a Directed Acyclic Graph).
- An undirected graph that has an edge between every pair of vertices is called a **complete** graph.

Let **n** = no. of vertices, and **m** = no. of edges

- *How many total edges in a complete graph?*
  - Each of the n vertices is incident to **n**-1 edges, however, we would have counted each edge twice! Therefore, intuitively, m = **n**(**n** -1)/2.
- Therefore, if a graph is not complete, m < **n**(**n** -1)/2

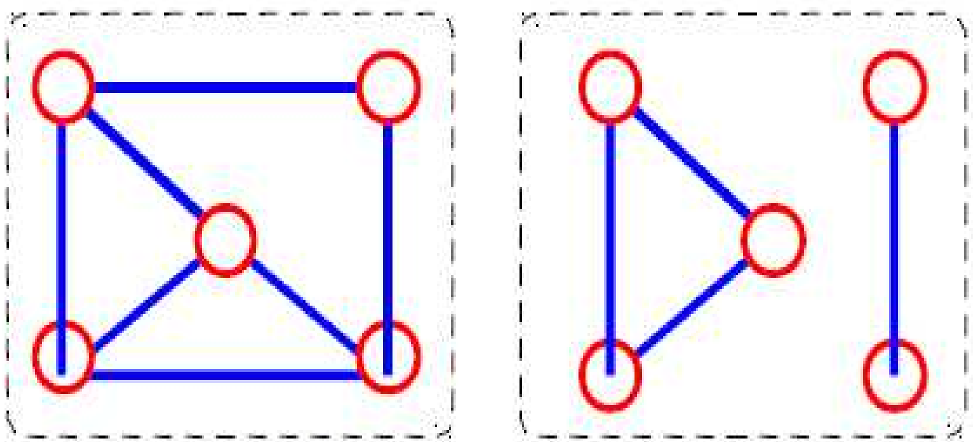
Note: A directed graph can also be a complete graph; in that case, there must be an edge from every vertex to every other vertex.

Terminology



$\mathbf{n} = 5$
$\mathbf{m} = (5 * 4)/2 = 10$

- connected graph: any two vertices are connected by some path



connected not connected

- subgraph: subset of vertices and edges forming a graph

Terminology

- An undirected graph **is connected** if a path exists from every vertex to every other vertex
- A directed graph is **strongly connected** if a path exists from every vertex to every other vertex
- A directed graph is **weakly connected** if a path exists from every vertex to every other vertex, disregarding the direction of the edge

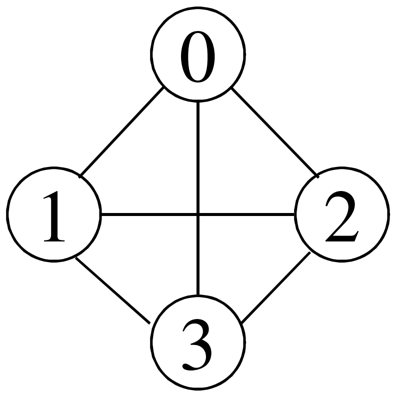
Graph representations

- Adjacency matrix - graph can be represented using a matrix of size total number of vertices by total number of vertices.
- Adjacency lists - every vertex of graph contains list of its adjacent vertices.

Adjacency matrix

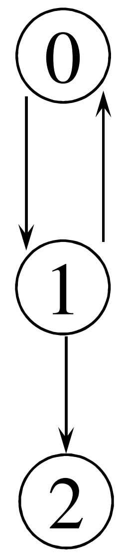
- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (vi, vj) is in E(G), adj_mat[i][j]=1
- If there is no such edge in E(G), adj_mat[i][j]=0
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric

Examples for Adjacency Matrix



$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

G_1



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

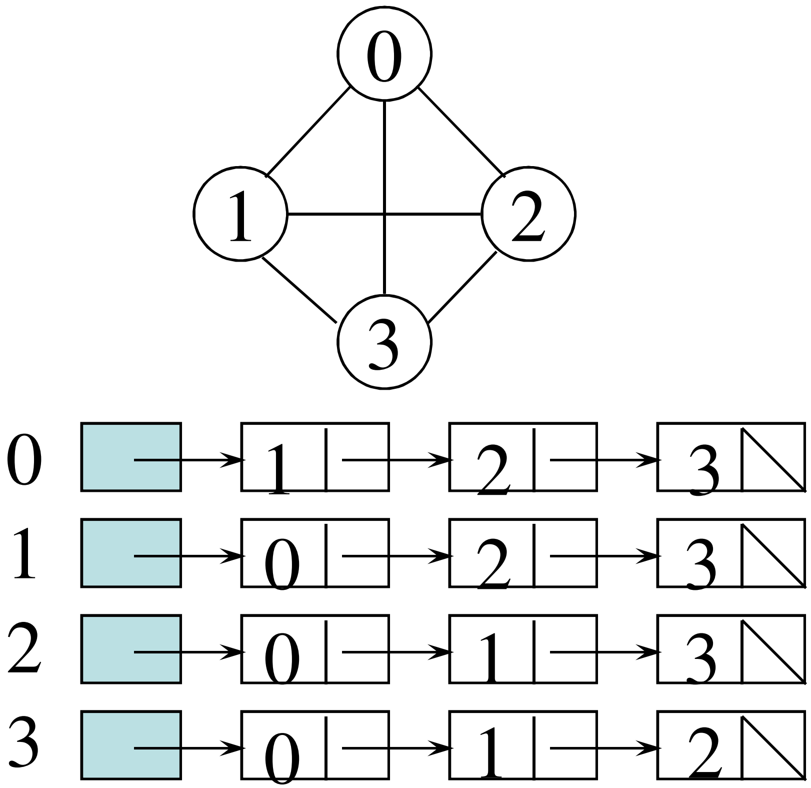
G_2

Adjacency Lists

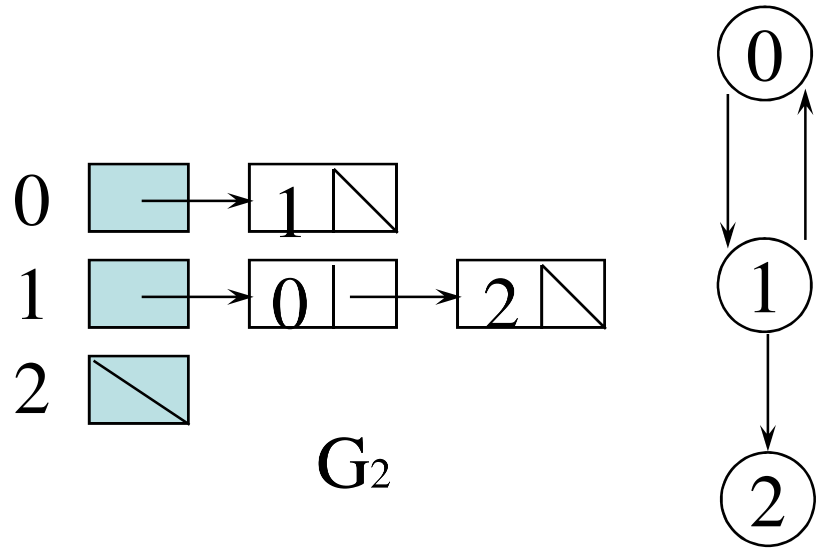
Each row in adjacency matrix is represented as an adjacency list

```
#define MAX_VERTICES 50
typedef struct node *node_pointer;
typedef struct node
{
   int vertex;
   struct node *link;
};
node_pointer graph[MAX_VERTICES];
```

Examples for Adjacency Lists



G_1



G_2

Summary

- Introduction to graph
- Graph terminologies
- Representation of graph
  - Adjacency matrix
  - Adjacency list