Advanced Data Structures and Algorithms

Associate Professor Dr. Raed Ibraheem Hamed

University of Human Development, College of Science and Technology Computer Science Department

2015 – 2016

What this Lecture is about:

- ✓ Graph Concepts
- ✓ Graphs
- ✓ Directed and Undirected Graphs
- ✓ Undirected Graph
- ✓ Directed Graph (Digraph)
- ✓ complete graph
- ✓ incomplete graph

Graph Concepts

- Graph terminology: vertex, edge, adjacent, incident, degree, cycle, path, connected component
- Types of graphs: undirected, directed, weighted
- Graph representations: adjacency matrix, array adjacency lists, linked adjacency lists
- Graph search methods: breath-first, depth-first search

Graphs

- G = (V,E)
- V is the **vertex** set.
- Vertices are also called nodes and points.
- E is the **edge** set.
- Each edge connects two vertices.
- Edges are also called arcs and lines.
- Vertices *i* and *j* are adjacent vertices iff (*i*, *j*) has an edge in the graph

Directed and Undirected Graphs

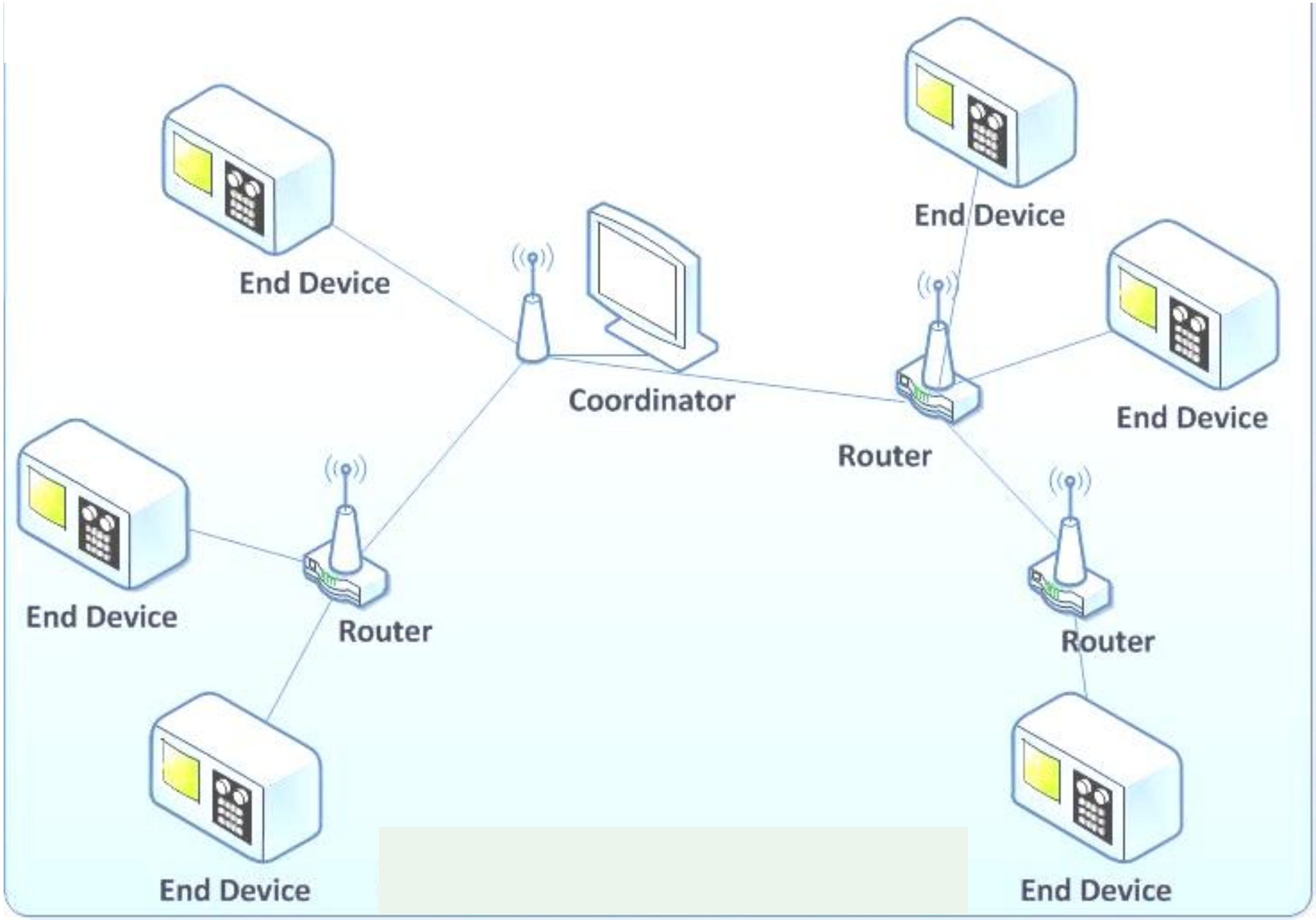
- Directed graph:
  - < v1, v2 > in E is ordered, i.e., a relation (v1,v2)
- Undirected graph:
  - < v1, v2 > in E is un-ordered, i.e., a set { v1, v2 }
- Degree of a node X:
  - Out-degree: number of edges < X, v2 >
  - In-degree: number of edges < v1, X >
  - Degree: In-degree + Out-degree

Graphs

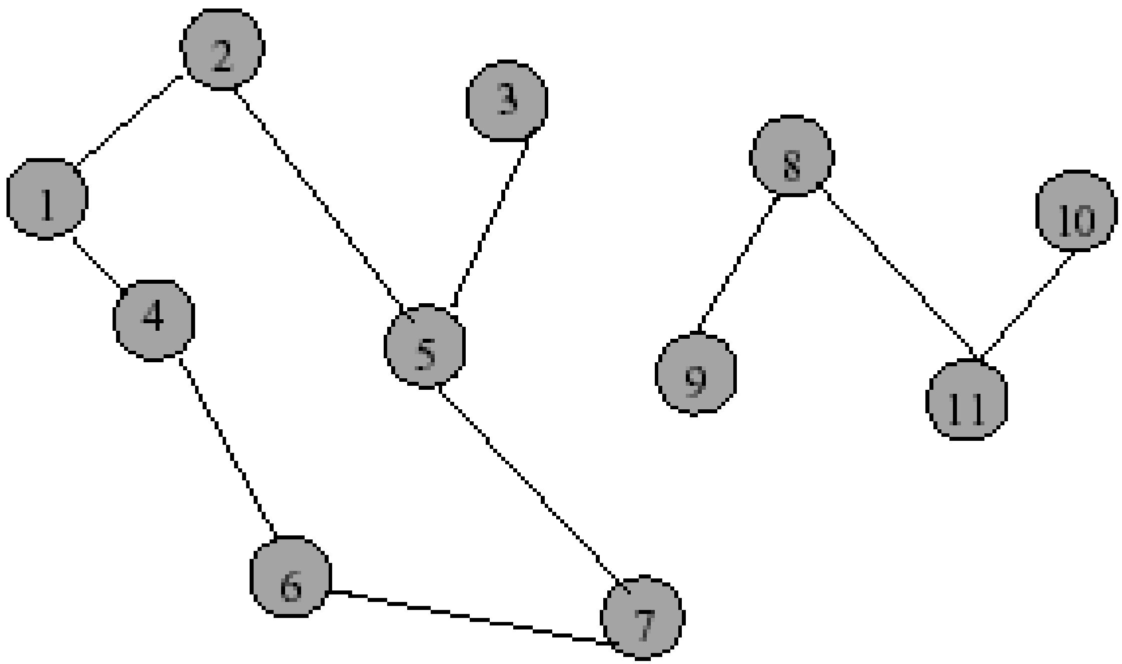
- Undirected edge has no orientation (no arrow head)
- Directed edge has an orientation (has an arrow head)
- Undirected graph – all edges are undirected
- Directed graph – all edges are directed

undirected edge **directed edge**

Applications – Communication Network



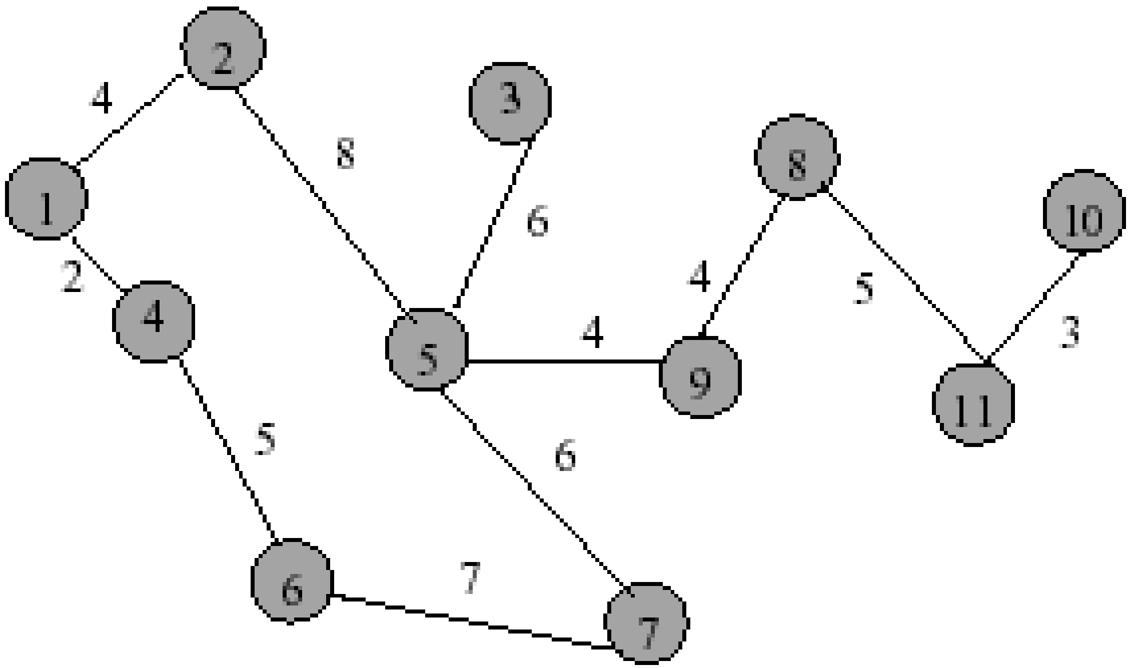
Applications – Communication Network



vertex = **Router**

edge = **Communication link**

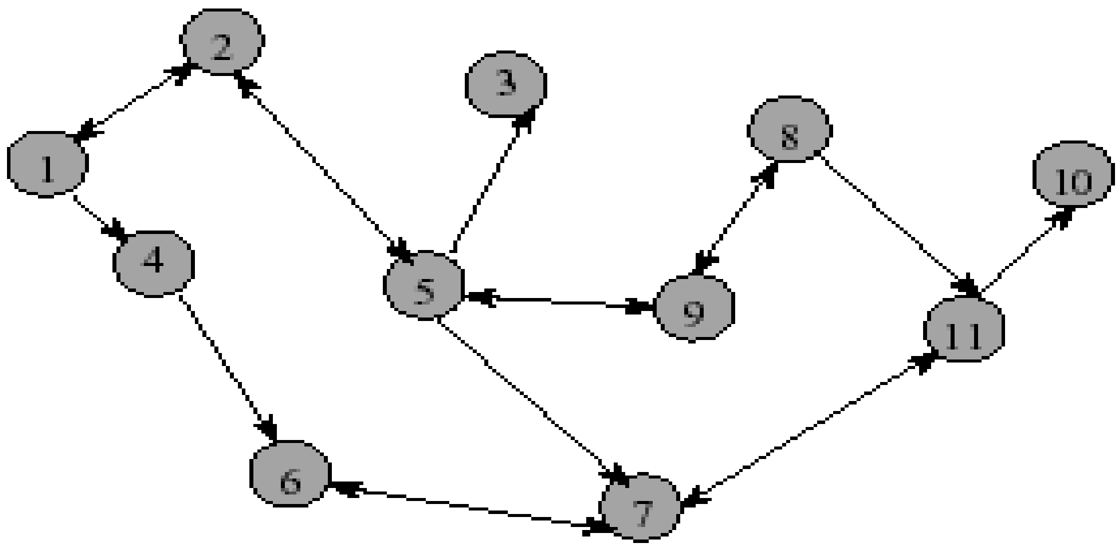
Applications - Driving Distance/Time Map



vertex = **City**

edge weight = **Driving Distance/Time**

Applications - Street Map



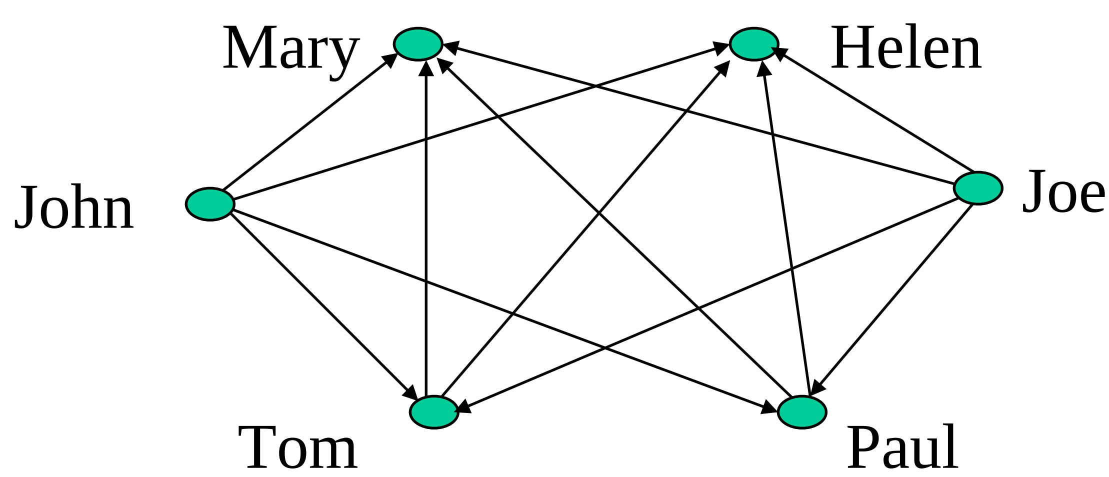
vertex = **Places**

edge = **Street**

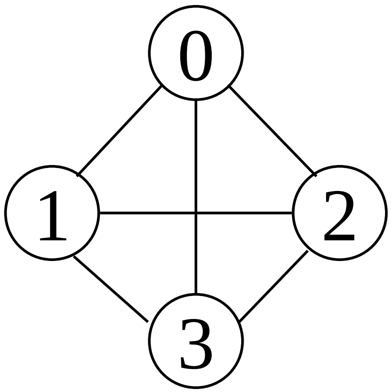
- **Streets are one- or two-way.**
- A single directed edge denotes a one-way street
- A two directed edge denotes a two-way street

A “Real-life” Example of a Graph

- V=set of 6 people: John, Mary, Joe, Helen, Tom, and Paul, of ages 12, 15, 12, 15, 13, and 13, respectively.
- E ={(x, y) | if x is **younger** than y}

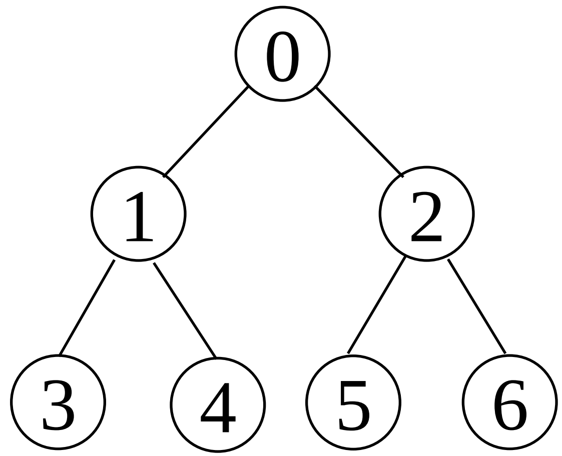


Examples for Graph



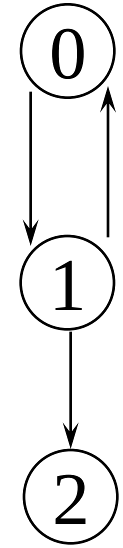
G_1

complete graph



G_2

incomplete graph



G_3

$V(G_1)=\{0,1,2,3\}$ $E(G_1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$

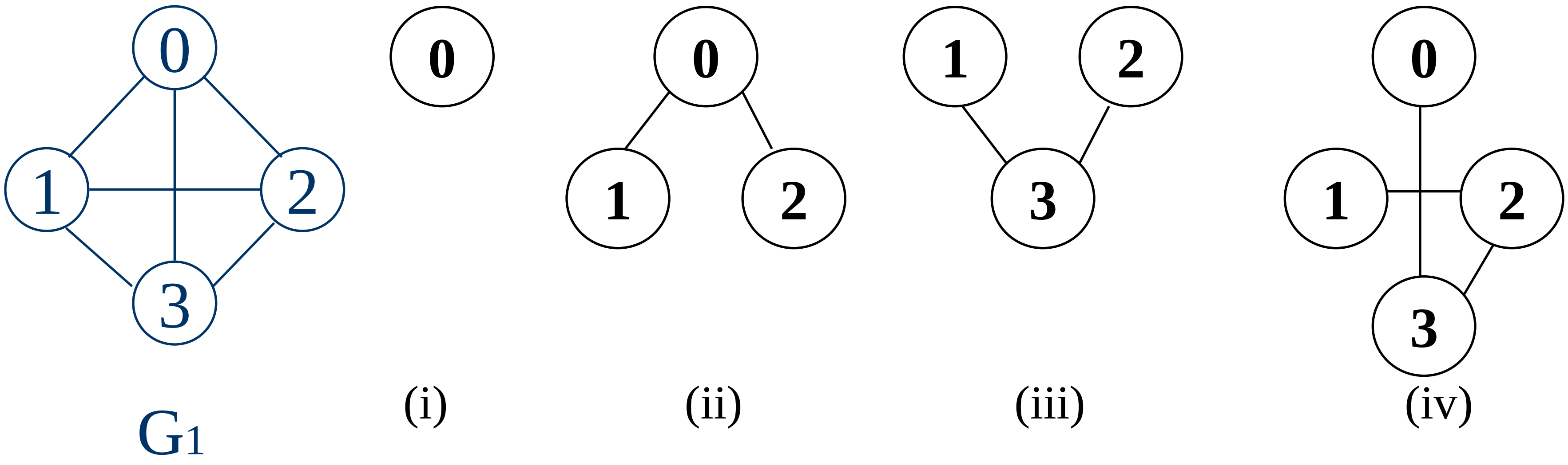
$V(G_2)=\{0,1,2,3,4,5,6\}$ $E(G_2)=\{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$

$V(G_3)=\{0,1,2\}$ $E(G_3)=\{<0,1>,<1,0>,<1,2>\}$

complete undirected graph: **n(n-1)/2** edges

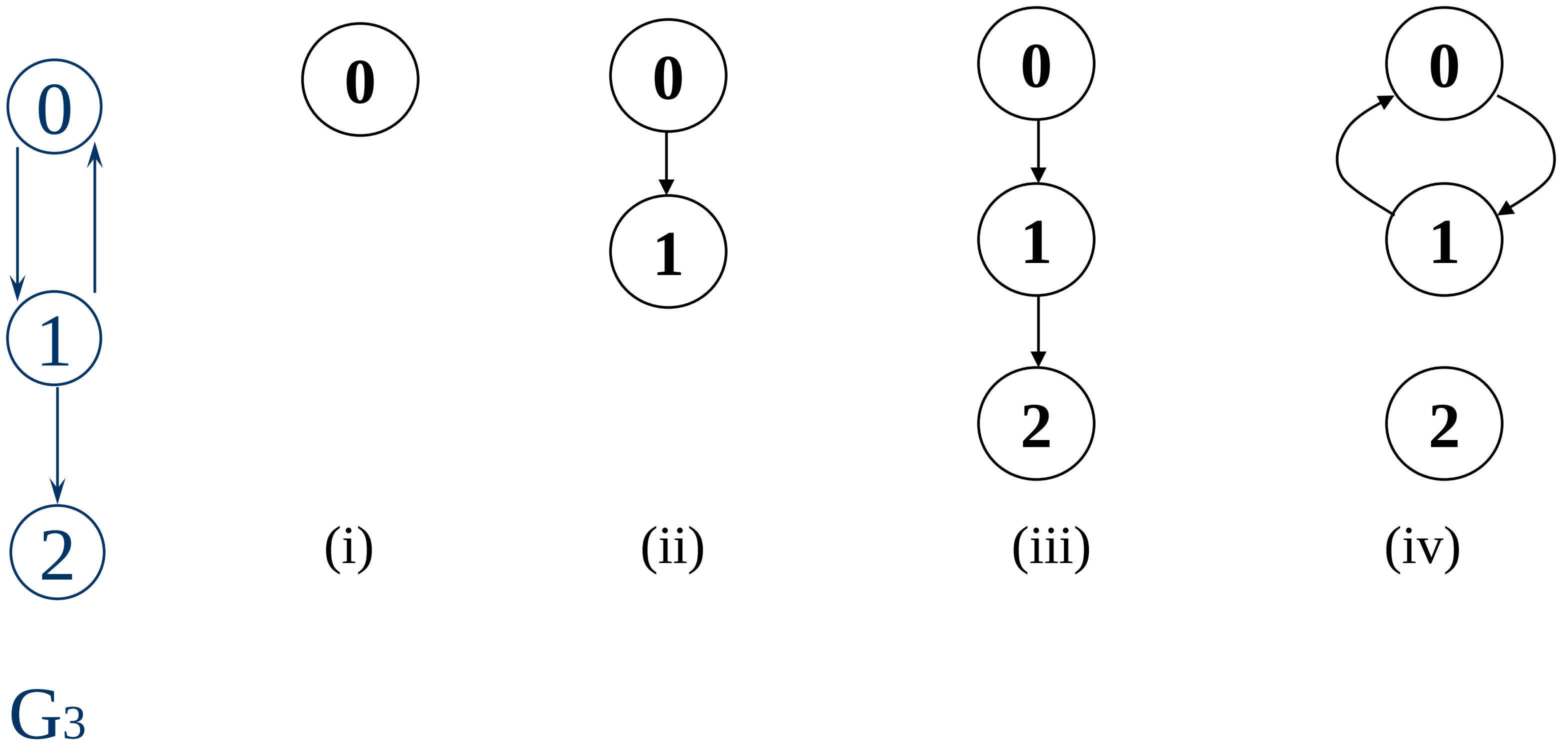
complete directed graph: **n(n-1)** edges

Subgraphs of G_1



(a) Some of the subgraph of G_1

Subgraphs of G_3



(b) Some of the subgraph of G_3

Intuition Behind Graphs

- The nodes represent entities (such as people, cities, computers, words, etc.)
- Edges (x,y) represent relationships between entities x and y, such as:
  - "x loves y"
  - "x hates y"
  - "x is a friend of y" (note that this not necessarily reciprocal)
  - "x considers y a friend"
  - "x is a child of y"
  - "x is a half-sibling of y"
  - "x is a full-sibling of y"
- In those examples, each relationship is a different graph

Graph Representation

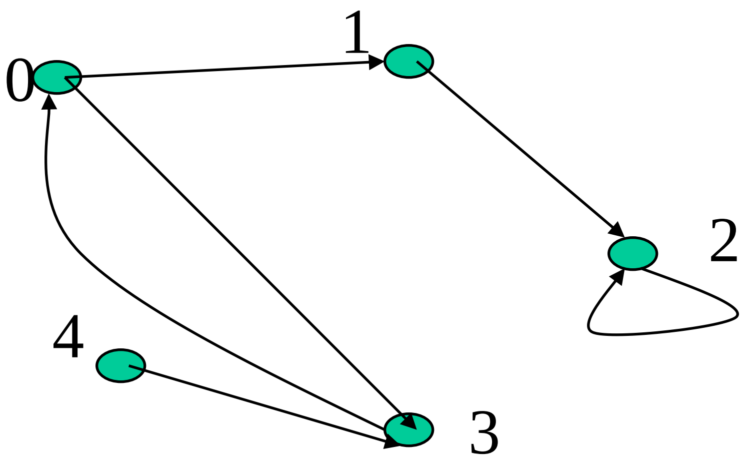
- For graphs to be computationally useful, they have to be conveniently represented in programs
- There are two computer representations of graphs:
  - Adjacency matrix representation
  - Adjacency lists representation

Adjacency Matrix Representation

- In this representation, each graph of n nodes is represented by an n x n matrix A, that is, a two-dimensional array A
- The nodes are (re)-labeled 1,2,…,n
- A[i][j] = 1 if (i, j) is an edge
- A[i][j] = 0 if (i, j) is not an edge

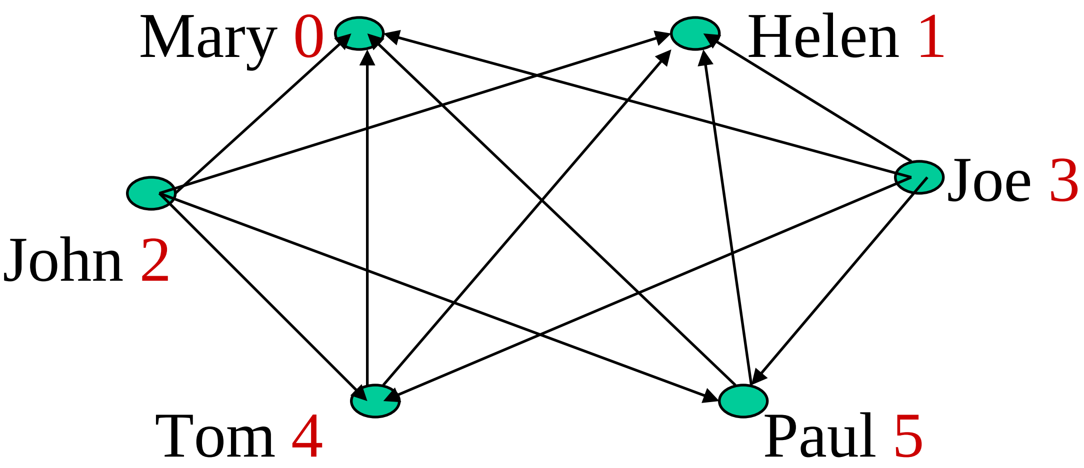
Example of Adjacency Matrix

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



Another Example of Adj. Matrix

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

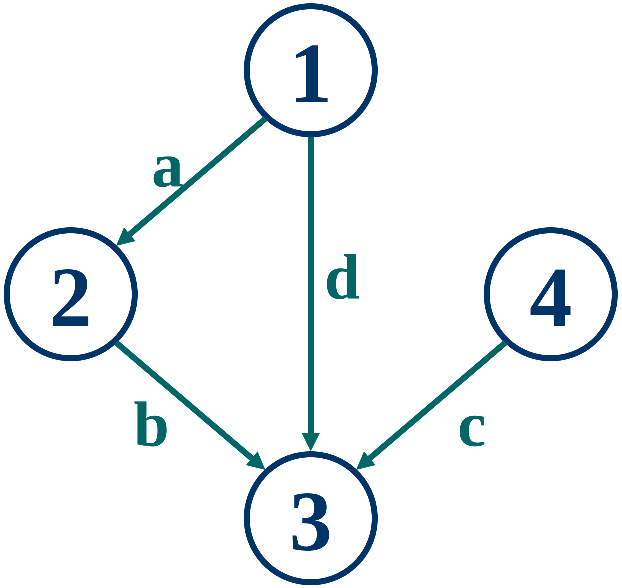


Representing Graphs

- Assume V = {1, 2, …, *n*}
- An *adjacency matrix* represents the graph as a *n* x *n* matrix A:
  - A[*i*, *j*] = 1 if edge (*i*, *j*) $\in$ E (or weight of edge)
    = 0 if edge (*i*, *j*) $\notin$ E

Graphs: Adjacency Matrix

- Example:



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

Pros and Cons of Adjacency Matrices

- Pros:
  - Simple to implement
  - Easy and fast to tell if a pair (i, j) is an edge: simply check if A[i][j] is 1 or 0
- Cons:
  - No matter how few edges the graph has, the matrix takes $O(n^2)$ in memory

Adjacency Lists Representation

- A graph of *n* nodes is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent from node i.
  - The nodes in the list L[i] are in no particular order

Example of Linked Representation

L[0]: empty

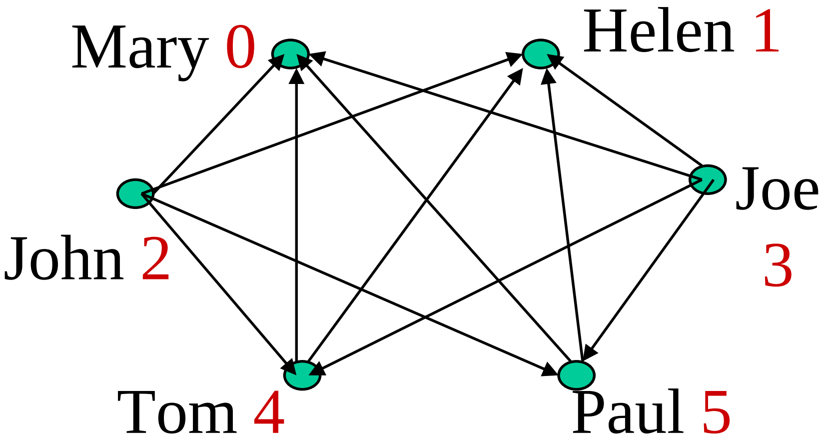
L[1]: empty

L[2]: 0, 1, 4, 5

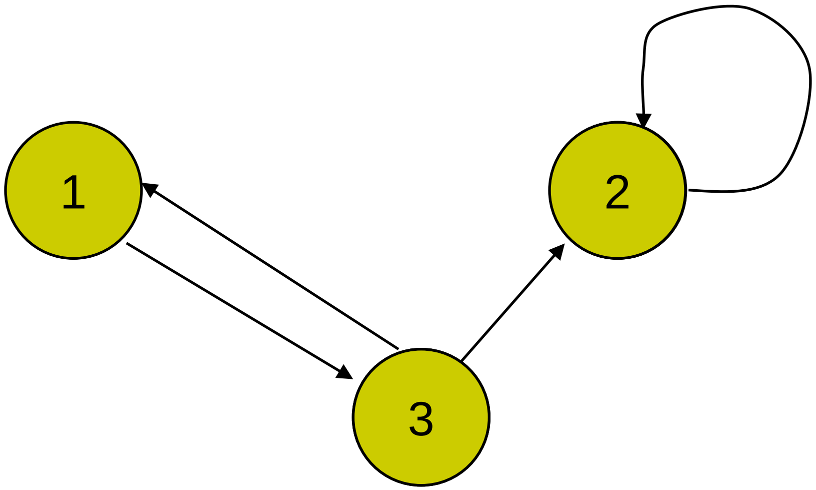
L[3]: 0, 1, 4, 5

L[4]: 0, 1

L[5]: 0, 1



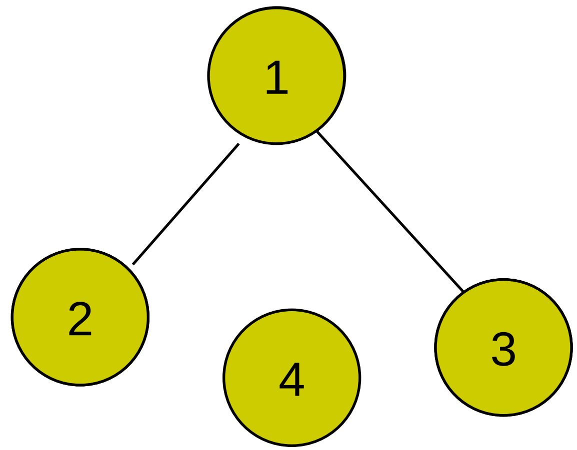
Examples



1 → 3
2 → 2
3 → 1 → 2

Undirected graphs:

- **Each edge is represented twice**

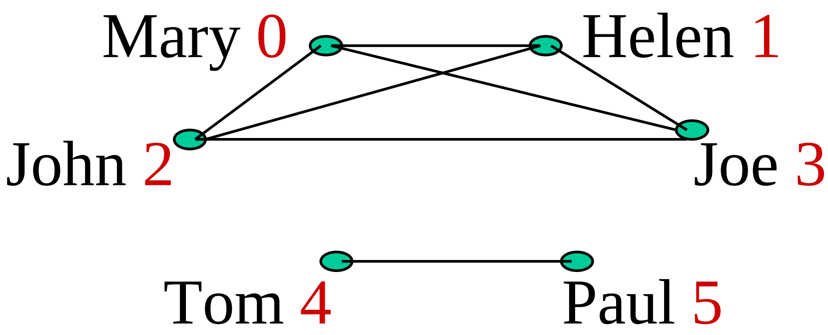


1 → 2 → 3
2 → 1
3 → 1
4 →

Pros and Cons of Adjacency Lists

- Pros:
  - Saves on space (memory): the representation takes as many memory words as there are nodes and edge.
- Cons:
  - It can take up to O(n) time to determine if a pair of nodes (i,j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].

Example of Representations



Linked Lists:

L[0]: 1, 2, 3

L[1]: 0, 2, 3

L[2]: 0, 1, 3

L[3]: 0, 1, 2

L[4]: 5

L[5]: 4

Adjacency Matrix:

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

Thank you ???