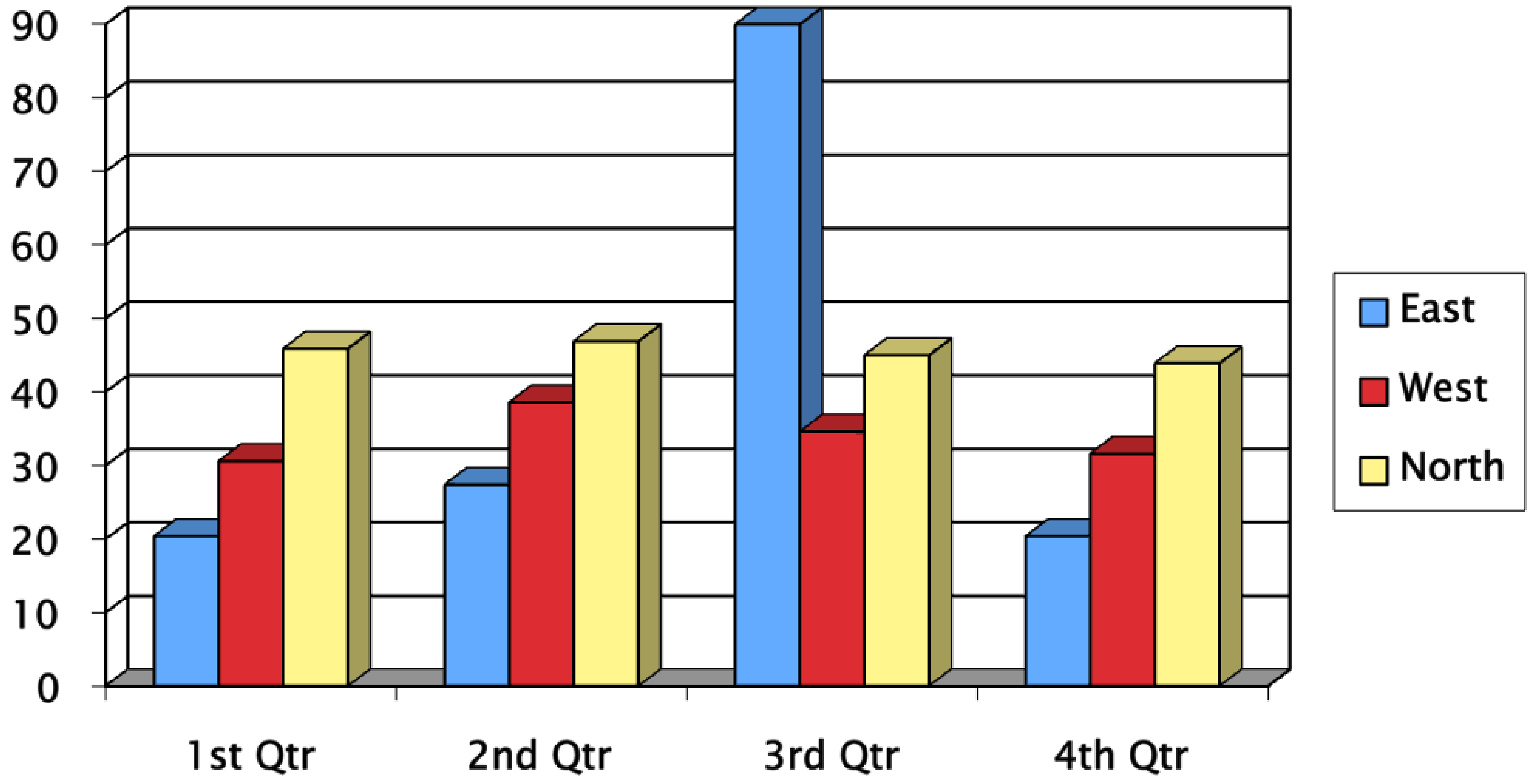
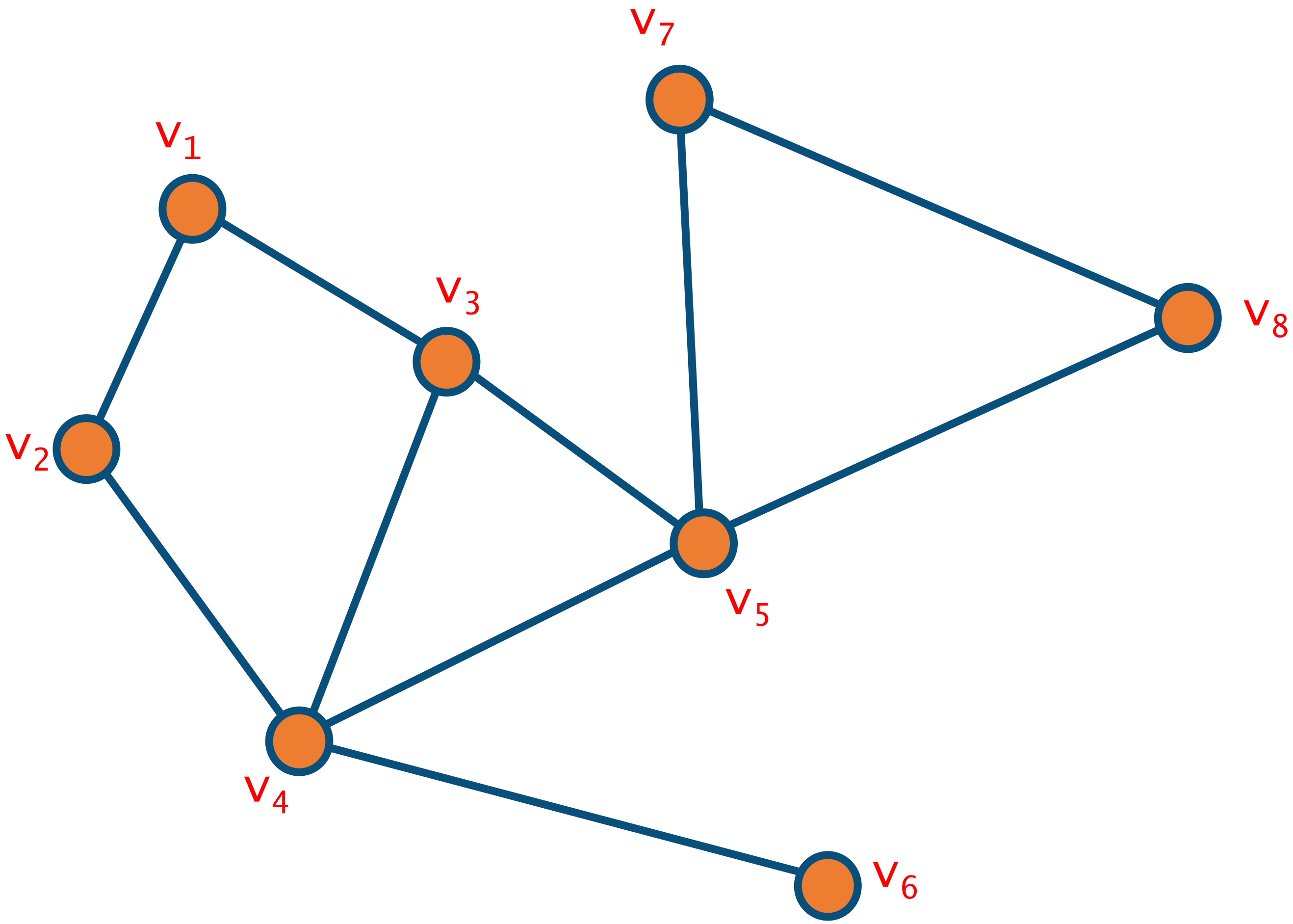
GRAPHS

Graphs – Not one of these

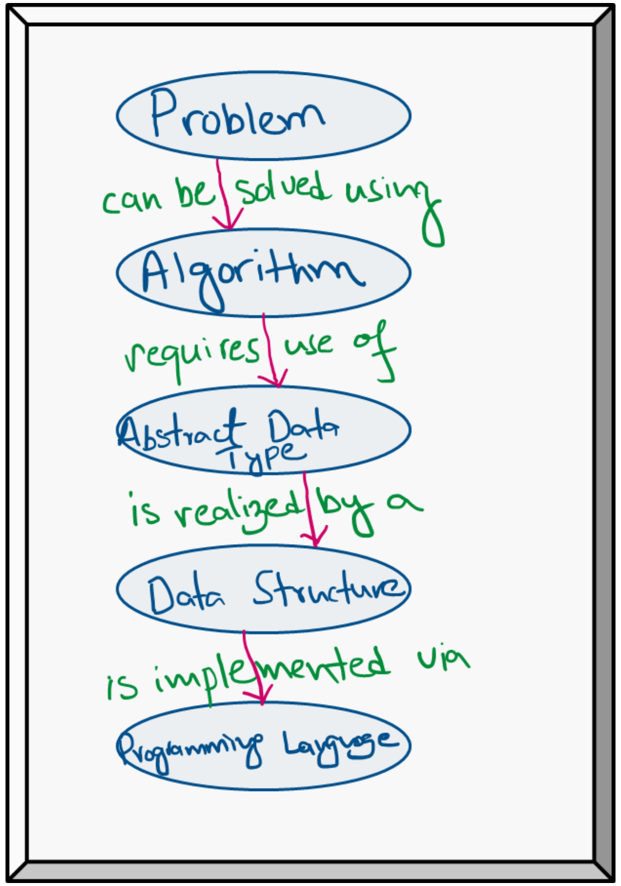


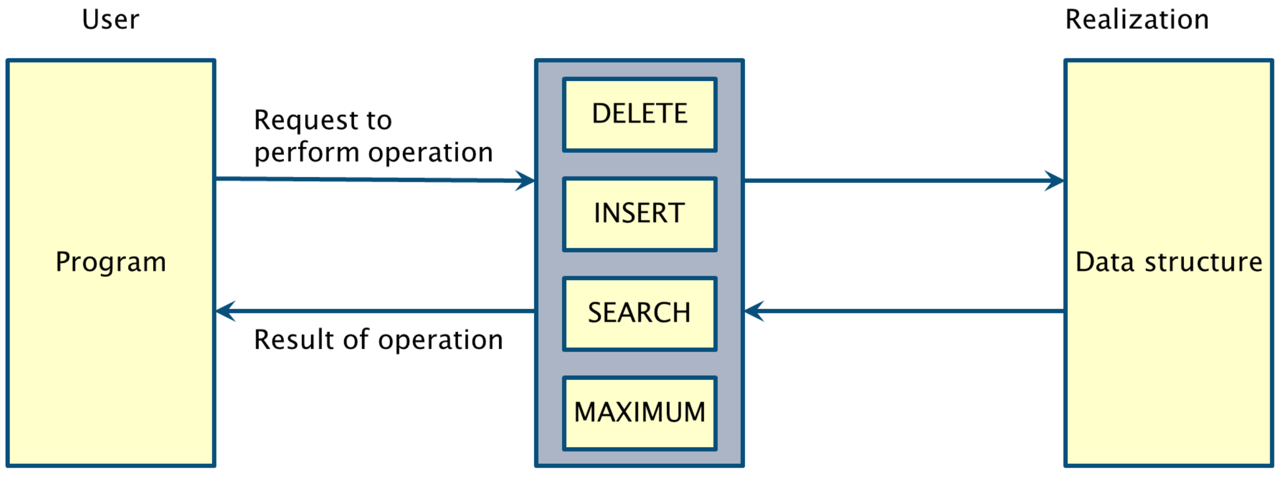
Graphs – One of these



Graph

An *ABSTRACT DATA TYPE*

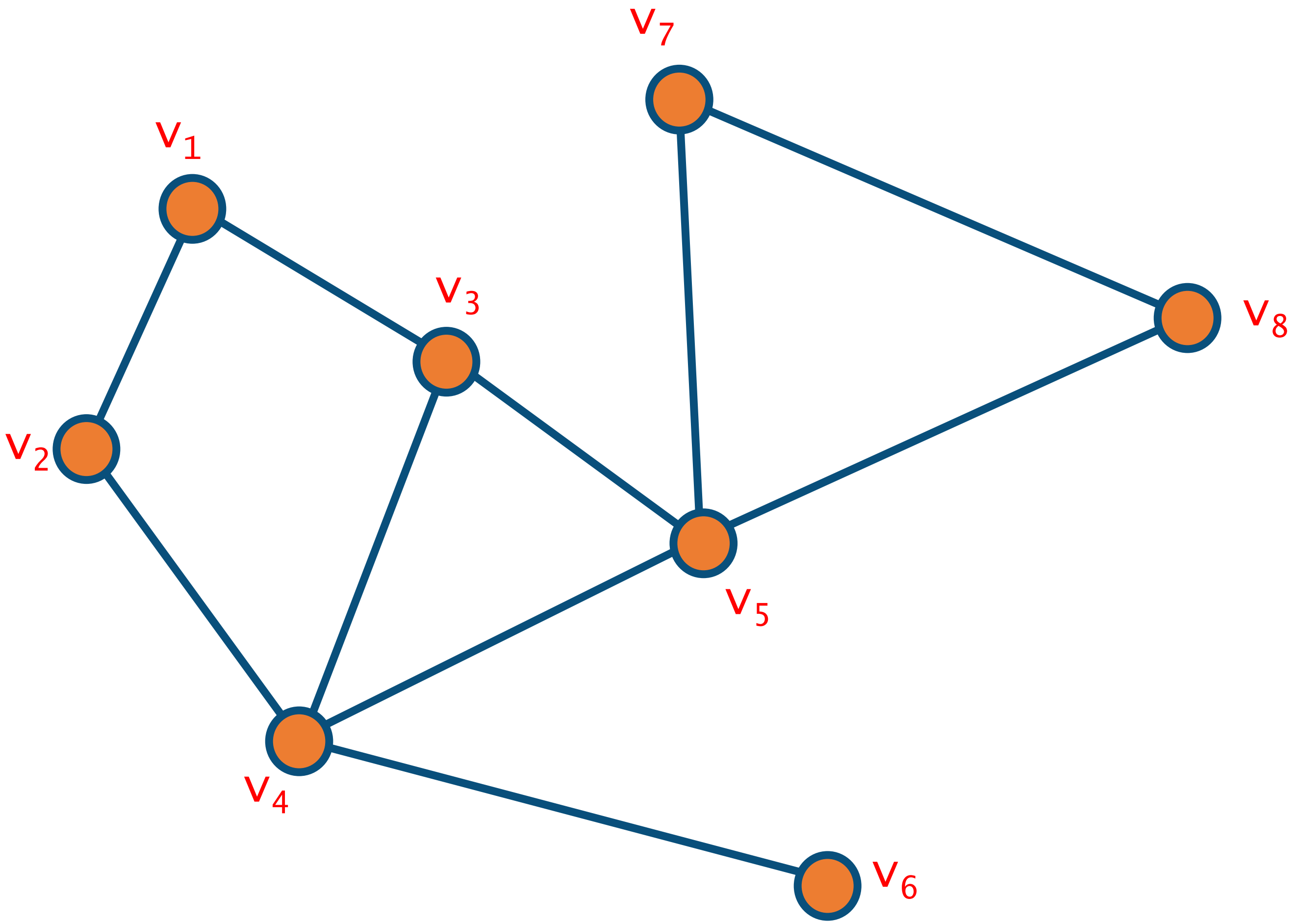




What ADT would you use for:

- Recording the connections between users in a social media application?
- Store and analyse information about traffic between various train stations?
- Measure and record the signal strengths between different WiFI hotspots in a building?
- Keep track of links to and from webpages in the WWW.
- Lists? Queues? Stacks?
  - They *could* work, but don't quite fit the pattern in the data.

Graphs – An Abstract Data Type



Graphs

Numerous applications

- computer networks
- communication networks
- pert networks
- scheduling
- www
- transportation problems
- chemical structures
- chemical reactions
- services (water, cable, …)
- AI and machine learning

Undirected Graphs: Definition

A graph $G = (V, E)$ * consists of:

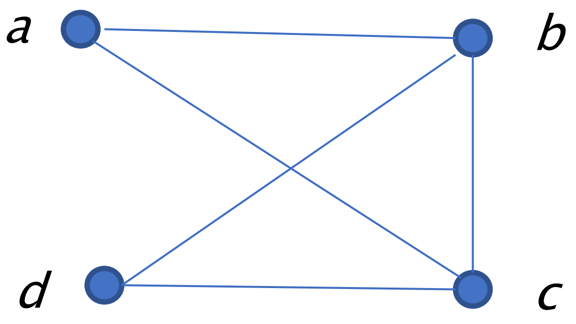
* A pair of sets

- V a finite, non-empty set of vertices (or nodes) (the vertex set)
- $E \subseteq \{\{u, v\} \mid u, v \in V \text{ and } u \neq v\}$ the edge set

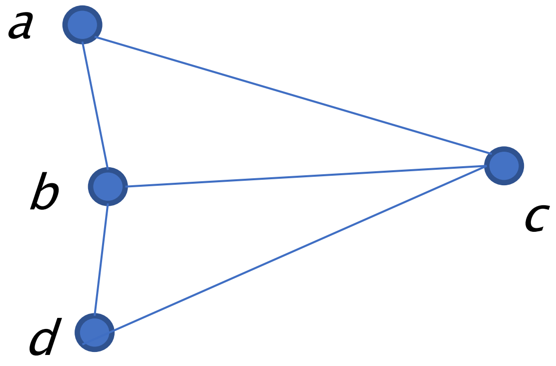
➢ Here we only study simple graphs: No “self-loops” or “parallel”/multiple edges are allowed by our definition

- For an edge $\boldsymbol{e} = \{\boldsymbol{u}, \boldsymbol{v}\} \in \boldsymbol{E}$ we say that nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ are the *endpoints of* $\boldsymbol{e}$.

- **Example: a graph with four vertices and five edges**



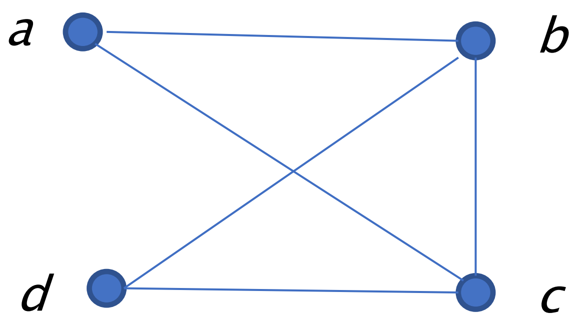
Equivalently:



Undirected Graphs – Representing via Sets

Example:

This is a graph with four vertices and five edges.



Definition: Graph G = (V,E) consists of

V, a finite, non-empty set of vertices (the vertex set)

E, a set of edges.

Each edge is a <u>subset</u> of V of size 2

V = {a, b, c, d}

E = { {a,b},
{a,c},
{b,c},
{b,d},
{c,d}
}

Graphs – Undirected

An (undirected) graph G = (V,E)

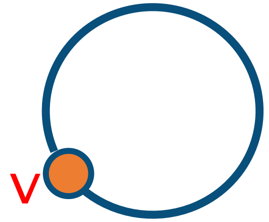
- V is finite set of vertices (the vertex set)
- E is set of edges, each edge is a subset of V of size 2 *(the edge set)*

Note: the definition does not allow loops

- i.e. edges joining vertices to themselves
- an edge is defined as a **set**, so same element cannot be repeated

Example

- the edge {v,v} equals the set containing only v, i.e. {v}
  - that is, an edge {v,v} is meaningless



Graphs – Undirected

An (undirected) graph G = (V,E)

- V is finite set of vertices (the vertex set)
- E is set of edges, each edge is a subset of V of size 2 (the edge set)

Note: the definition does not allow mutiple edges

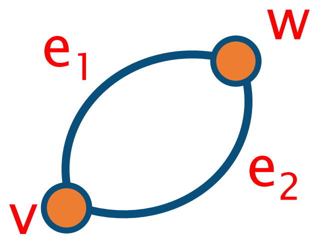
- i.e. two edges between the same vertices

Example

- both e_1 and e_2 equal {v,w} and E is a set
- so edge between u and v can only appear once

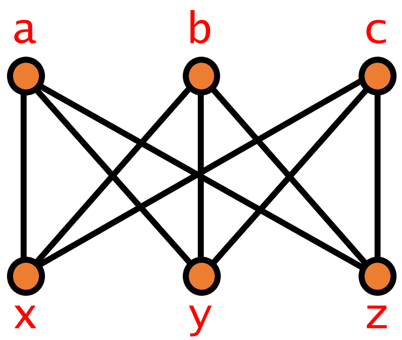
$E = \{\{v,w\},\{v,w\}\}$

$= \{\{v,w\}\}$



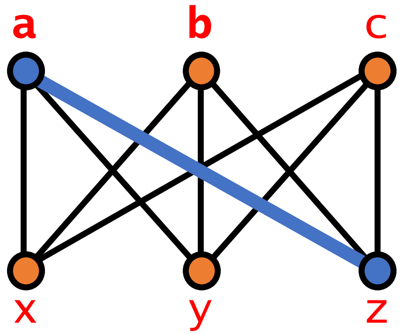
Undirected graphs defined like so, *without* either loops or multiple edges, are often called <u>simple graphs</u>

Graph – Terminology



In this graph:

Graph – Terminology



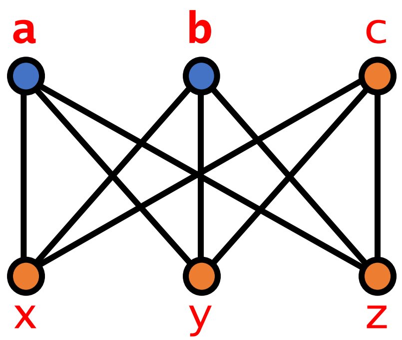
V={a,b,c,x,y,z}

E={ {a,x},{a,y},{a,z},
{b,x},{b,y},{b,z},
{c,x},{c,y},{c,z} }

In this graph:

– vertices a & z are adjacent that is {a,z} is an element of the edge set E

Graph – Terminology



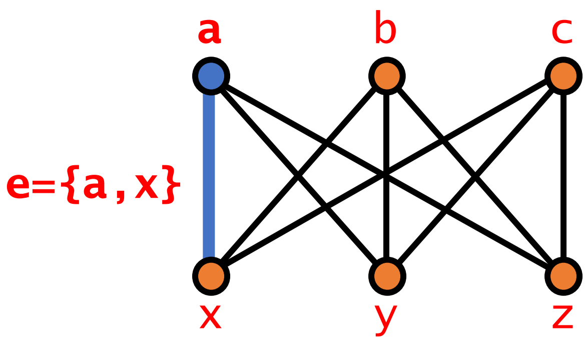
V={a,b,c,x,y,z}

E={ {a,x},{a,y},{a,z},
{b,x},{b,y},{b,z},
{c,x},{c,y},{c,z} }

In this graph:

- vertices a & z are adjacent that is {a,z} is an element of the edge set E
- vertices a & b are non-adjacent that is {a,b} is not an element of E

Graph – Terminology



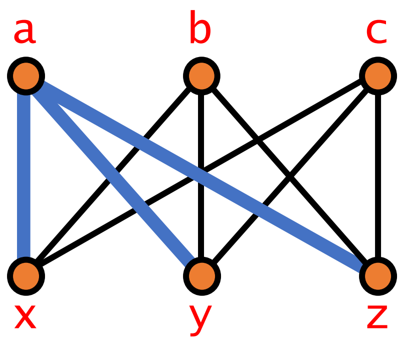
V={a,b,c,x,y,z}

E={ {a,x},{a,y},{a,z},
{b,x},{b,y},{b,z},
{c,x},{c,y},{c,z} }

In this graph:

- vertices a & z are adjacent that is {a,z} is an element of the edge set E
- vertices a & b are non-adjacent that is {a,b} is not an element of E
- vertex a is incident to edge e

Graph – Terminology



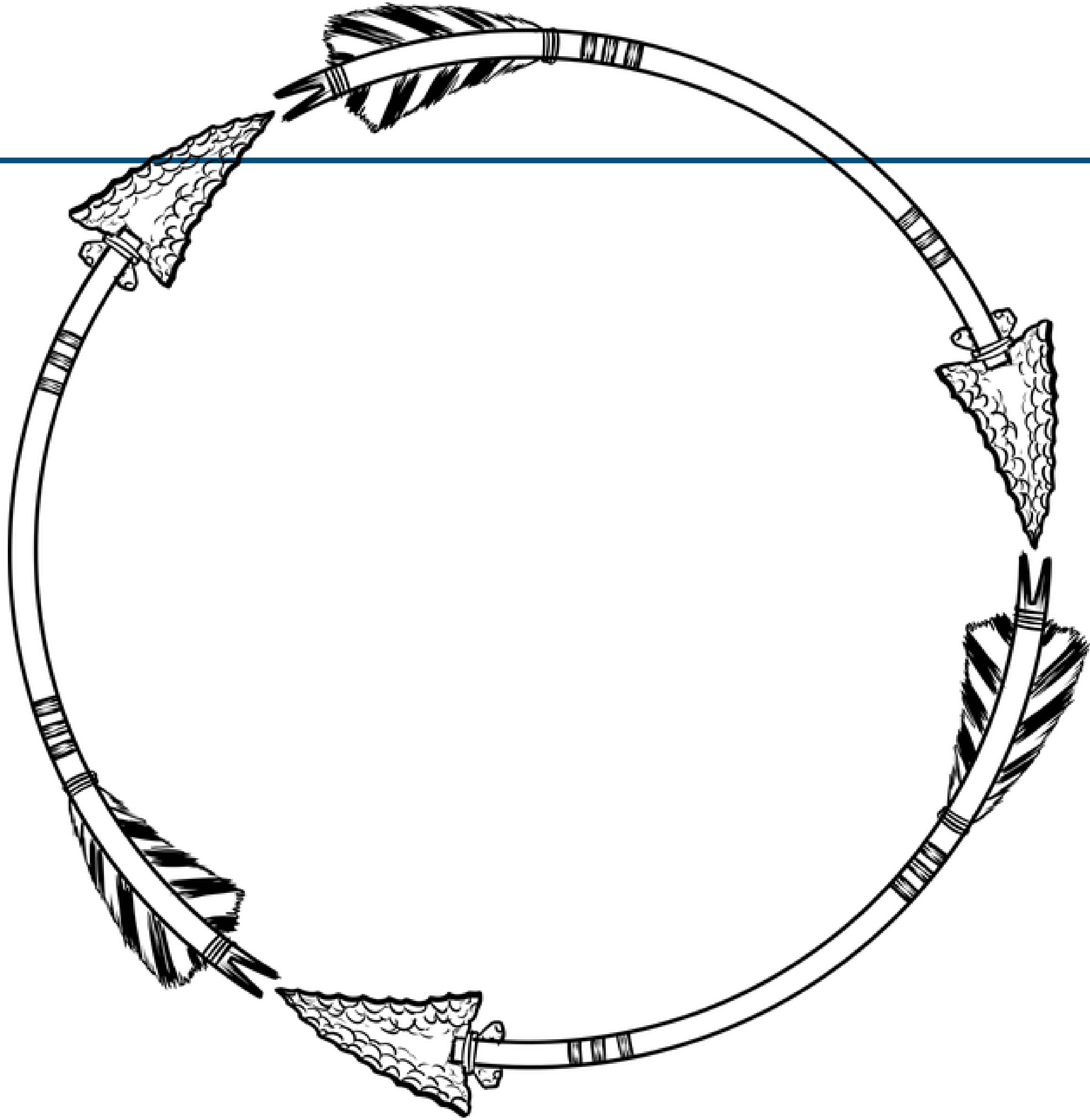
V={a,b,c,x,y,z}

E={ {a,x},{a,y},{a,z},
{b,x},{b,y},{b,z},
{c,x},{c,y},{c,z} }

In this graph:

- vertices a & z are adjacent that is {a,z} is an element of the edge set E
- vertices a & b are non-adjacent that is {a,b} is not an element of E
- vertex a is incident to edge {a,x}
- the degree of a vertex is the number of edges it is incident to
- in this graph all vertices have degree 3, e.g. deg(a)=3
  - i.e. all vertices are incident to three edges

DIRECTED GRAPHS

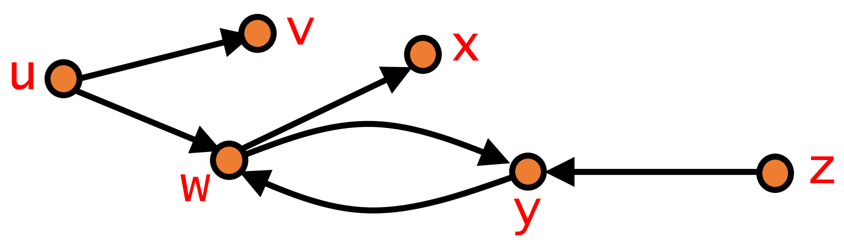


Graphs – Directed

A directed graph (digraph) **D = (V,E)**

- V is the finite set of vertices and E is the finite set of edges
- here each edge is an ***ordered*** pair (x,y) of vertices (element of V×V)

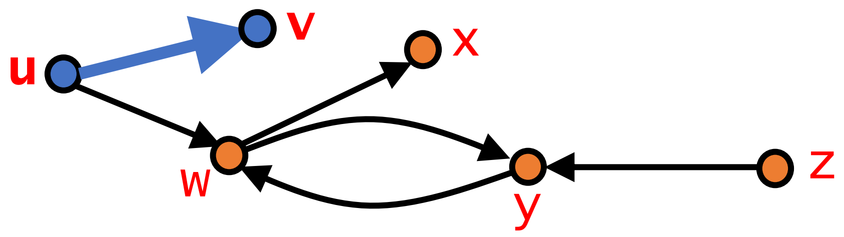
Pictorially: edges are drawn as directed lines/arrows



V={u,v,w,x,y,z}

E={ (u,v),(u,w),(w,x),
(w,y),(y,w),(z,y) }

Directed Graphs – Terminology



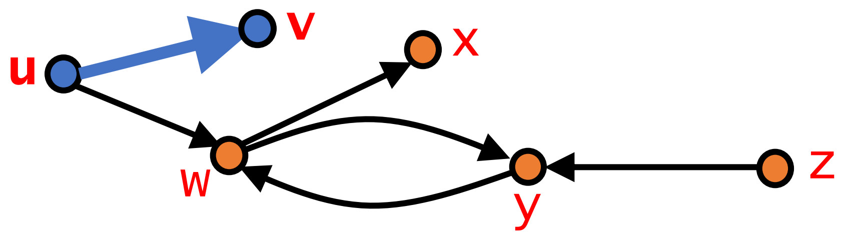
V={u,v,w,x,y,z}

E={ (u,v),(u,w),(w,x),
(w,y),(y,w),(z,y) }

In this graph

- u is the source (initial) vertex of edge e=(u,v)

Directed Graphs – Terminology



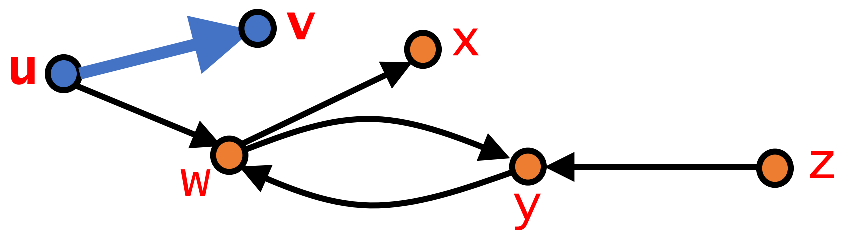
V={u,v,w,x,y,z}

E={ (u,v),(u,w),(w,x),
(w,y),(y,w),(z,y) }

In this graph

- u is the source (initial) vertex of edge e=(u,v)
- v is the target (final) vertex of edge e=(u,v)

Directed Graphs – Terminology



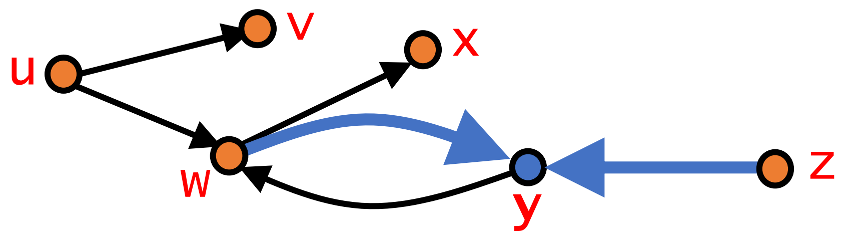
V={u,v,w,x,y,z}

E={ (u,v),(u,w),(w,x),
(w,y),(y,w),(z,y) }

In this graph

- u is the source (initial) vertex of edge e=(u,v)
- v is the target (final) vertex of edge e=(u,v)
- u is adjacent to v and v is adjacent from u

Directed Graphs – Terminology



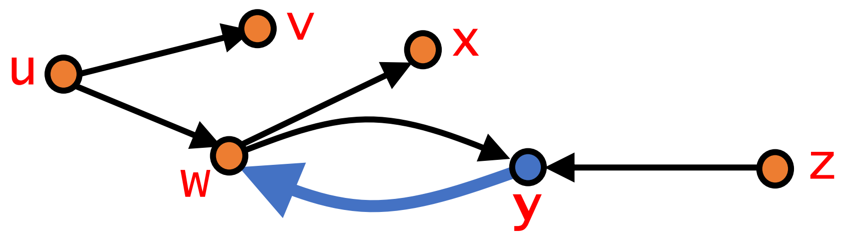
V={u,v,w,x,y,z}

E={ (u,v),(u,w),(w,x),
(w,y),(y,w),(z,y) }

In this graph

- u is the source (initial) vertex of edge e=(u,v)
- v is the target (final) vertex of edge e=(u,v)
- u is adjacent to v and v is adjacent from u
- y has in-degree 2
    - in-degree of vertex y: the number of edges that has y as its target

Directed Graphs – Terminology



V={u,v,w,x,y,z}

E={ (u,v),(u,w),(w,x),
(w,y),(y,w),(z,y) }

In this graph

- u is the source (initial) vertex of edge e=(u,v)
- v is the target (final) vertex of edge e=(u,v)
- u is adjacent to v and v is adjacent from u
- y has in-degree 2
  - in-degree of vertex y: the number of edges that has y as its target
- y has out-degree 1
  - out-degree of vertex y: the number of edges that has y as its source

REPRESENTING GRAPHS

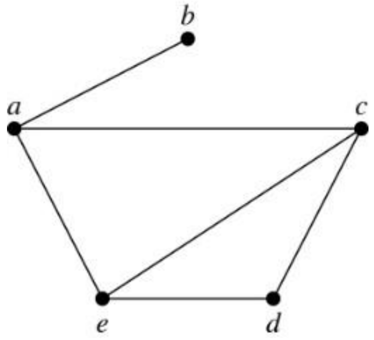
Representing Graphs:
Adjacency Lists

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Representing Graphs: Adjacency Lists

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:



Representing Graphs: Adjacency Lists

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:

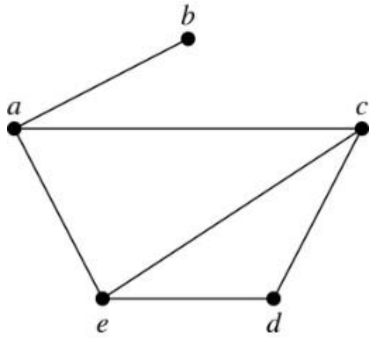


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	*Adjacent Vertices*
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

Representing Graphs: Adjacency Lists

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:

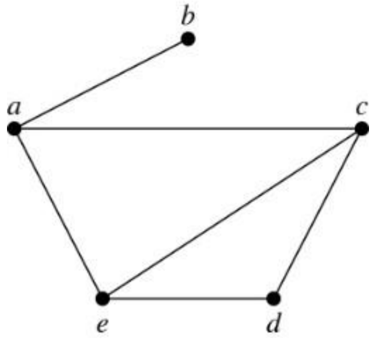
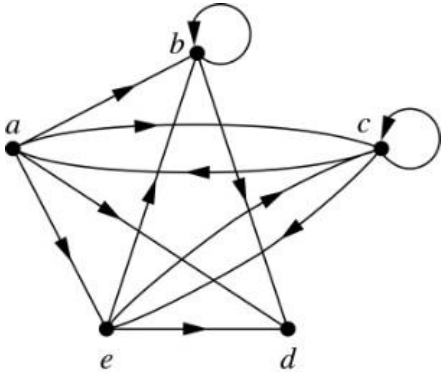


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	*Adjacent Vertices*
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

Example:



Representing Graphs: Adjacency Lists

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:

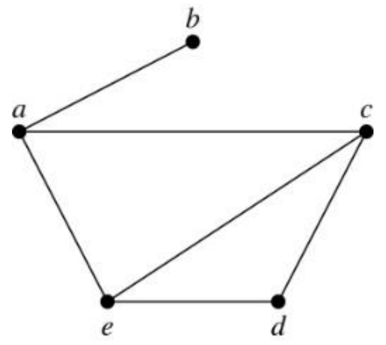


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	*Adjacent Vertices*
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

Example:

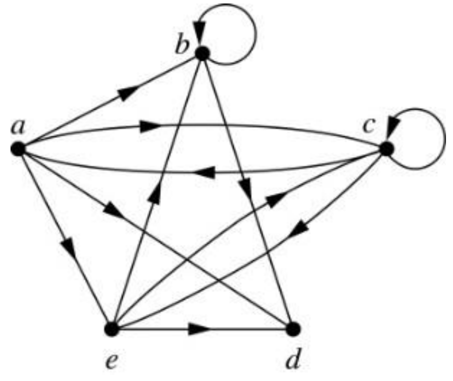
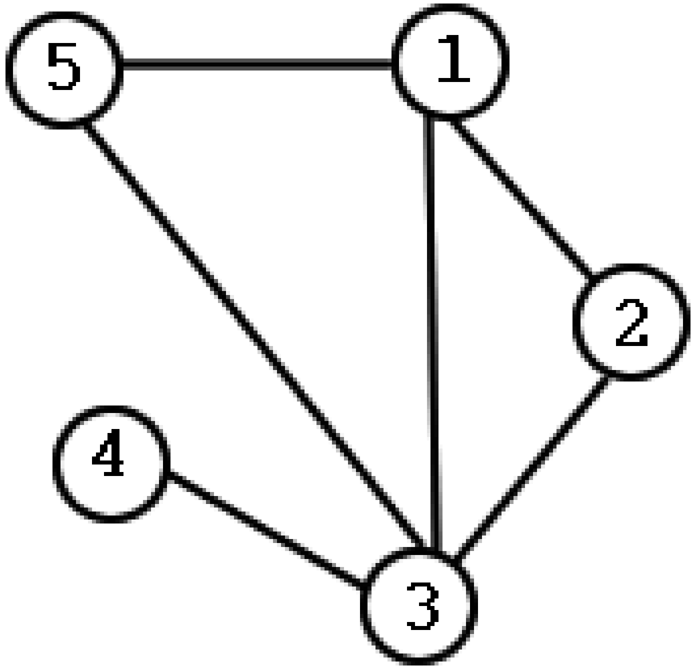


TABLE 2 An Adjacency List for a Directed Graph.

Initial Vertex	*Terminal Vertices*
a	b, c, d, e
b	b, d
c	a, c, e
d	
e	b, c, d

Representing Graphs: Adjacency Matrices

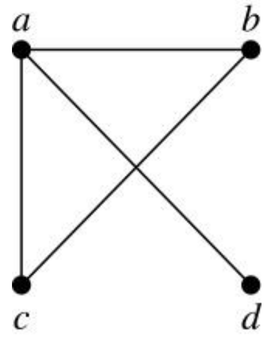


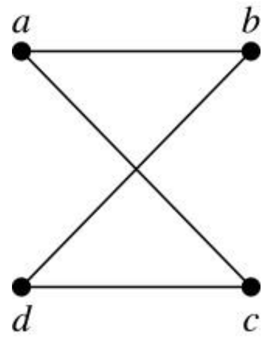
$$
\Rightarrow A = \begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}
\end{array}
$$

Representing Graphs: Adjacency Matrices

For an undirected graph with n vertices, *adjacency matrix* is the $n \times n$ zero-one matrix with 1 as its $(i, j)^{th}$ entry when i^{th} and j^{th} vertex are adjacent, and 0 as its $(i, j)^{th}$ entry when they are not adjacent.

Example:



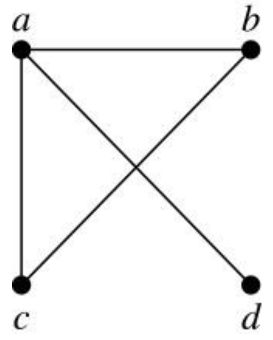


It is possible to represent directed graphs as well.

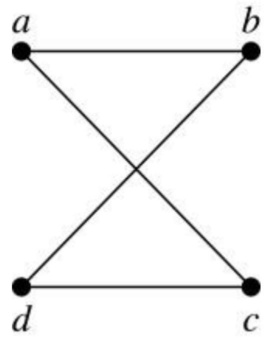
Representing Graphs:
Adjacency Matrices

For an undirected graph with n vertices, *adjacency matrix* is the n × n zero-one matrix
with 1 as its $(i, j)^{th}$ entry when i^{th} and j^{th} vertex are adjacent,
and 0 as its $(i, j)^{th}$ entry when they are not adjacent.

Example:



$$\begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

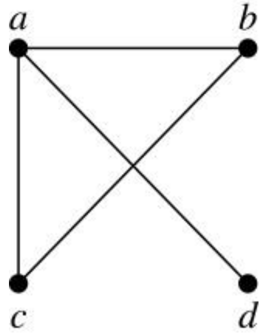


It is possible to represent directed graphs as well.

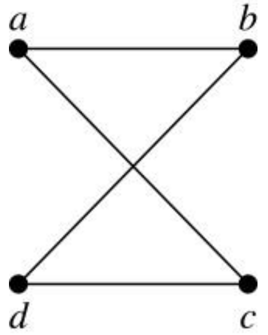
Representing Graphs: Adjacency Matrices

For an undirected graph with n vertices, *adjacency matrix* is the n × n zero-one matrix
with 1 as its (i, j)th entry when ith and jth vertex are adjacent,
and 0 as its (i, j)th entry when they are not adjacent.

Example:

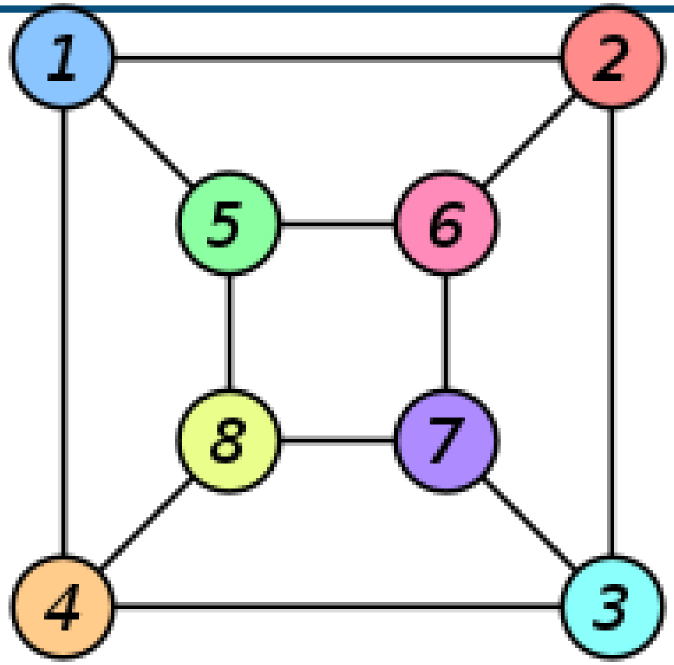


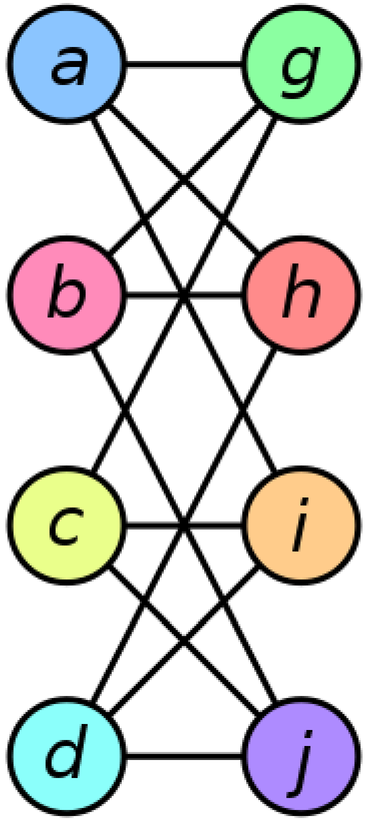
$$\begin{array}{c c} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



$$\begin{array}{c c} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

It is possible to represent directed graphs as well.





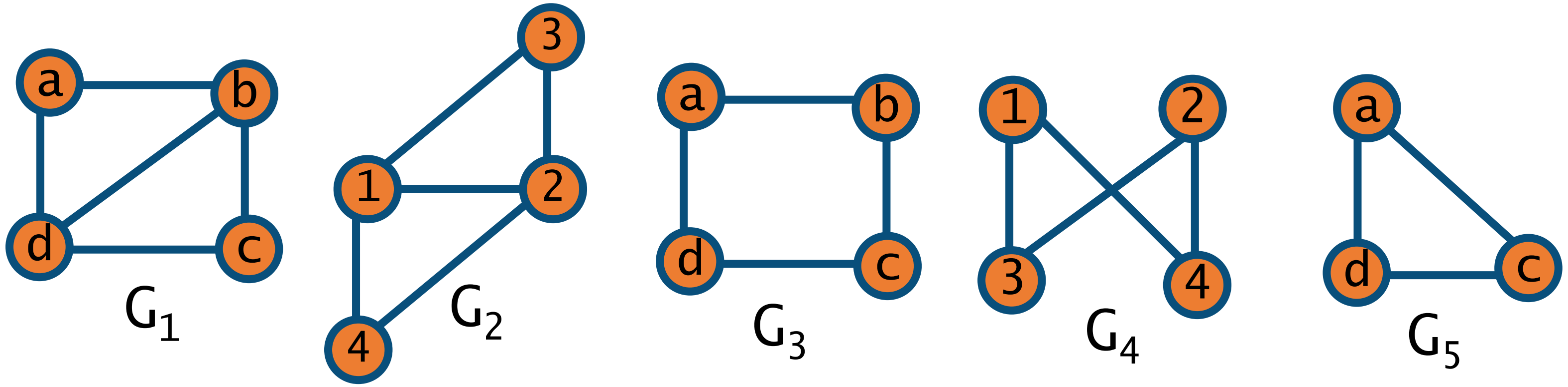
GRAPH ISOMPORPHISM
&
CONNECTIVITY

https://upload.wikimedia.org/wikipedia/commons/thumb/8/84/Graph_isomorphism_b.svg/200px-Graph_isomorphism_b.svg.png

Isomorphic graphs

When are graphs two graphs "similar"? When do they have the *same* structure?

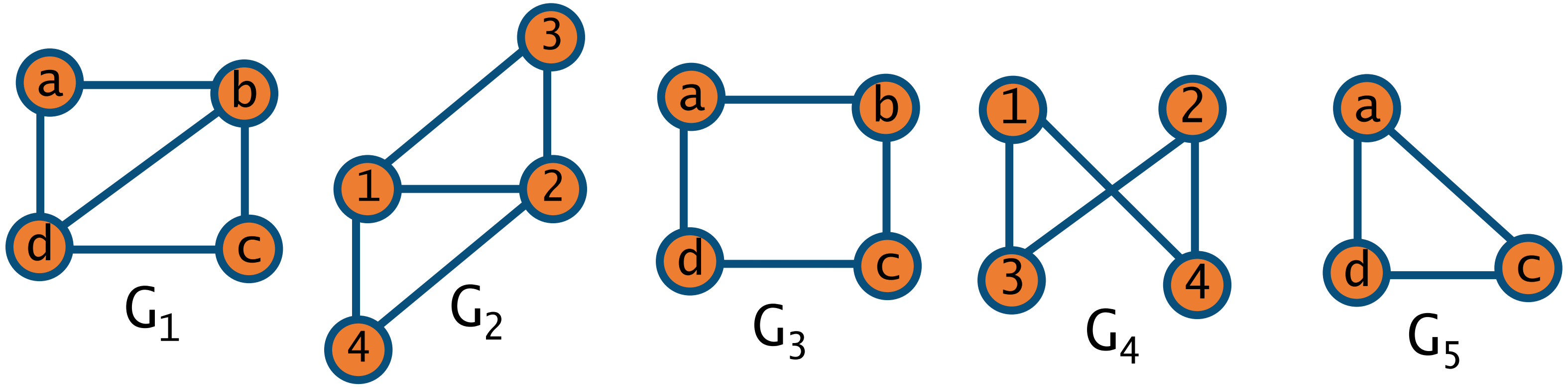
(That is, if we were to rename the vertices of one, it *becomes* the other)



Isomorphic graphs

When are graphs the similar? When do they have the *same* structure?

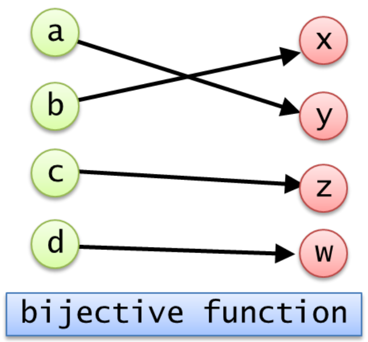
(That is, if we were to rename the vertices of one, it *becomes* the other)



We need the concept of Isomorphism

– e.g. there are isomorphisms between G_1 and G_2 and between G_3 and G_4

Isomorphic graphs



Are two graphs G_1 and G_2 isomorphic

- that is, could the vertices of G_1 be renamed such that G_1 becomes G_2

Formally: Undirected graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are isomorphic if:

- there is a bijection f from the vertex set V_1 to the vertex set V_2
- **and**: $\{v,w\} \in E_1$ if and only if $\{f(v),f(w)\} \in E_2$
  - i.e. v and w are adjacent if and only if f(v) and f(w) are adjacent

So far, best algorithm for checking two graphs are isomorphic is exponential, i.e $O(2^n)$, in the worst case.

Isomorphic graphs

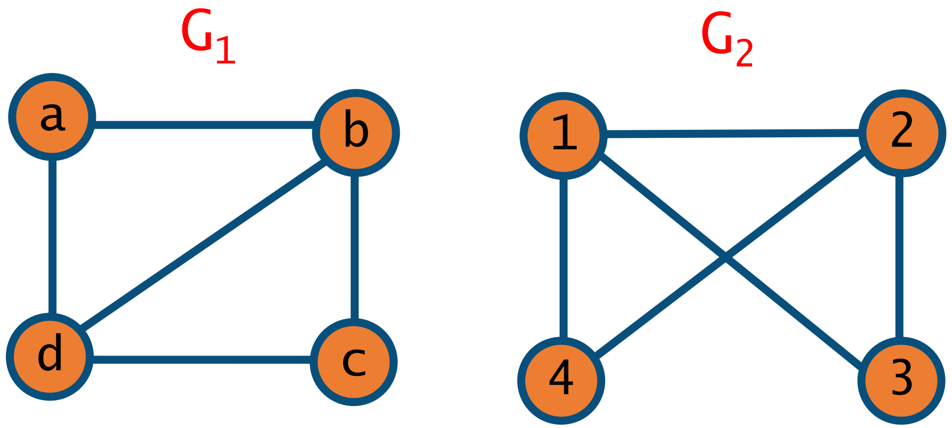
Are two graphs G_1 and G_2 isomorphic

– that is, could the vertices of G_1 be renamed such that G_1 becomes G_2

More formally:

– is there a bijection `f` from the vertex set V_1 to the vertex set V_2 such that $\{v,w\} \in E_1$ if and only if $\{f(v),f(w)\} \in E_2$

Are these graphs isomorphic?



Isomorphic graphs

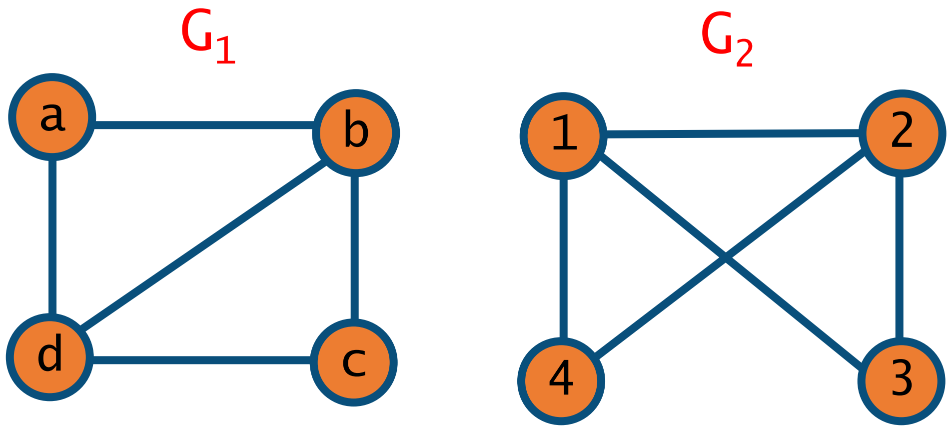
Are two graphs G_1 and G_2 isomorphic

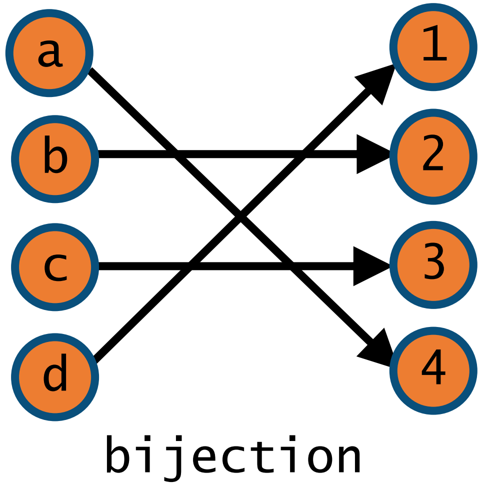
- that is, could the vertices of G_1 be renamed such that G_1 becomes G_2

More formally:

- is there a bijection f from the vertex set V_1 to the vertex set V_2 such that $\{v,w\} \in E_1$ if and only if $\{f(v),f(w)\} \in E_2$

Are these graphs isomorphic? Here is a bijection





Isomorphic graphs

Are two graphs G_1 and G_2 isomorphic

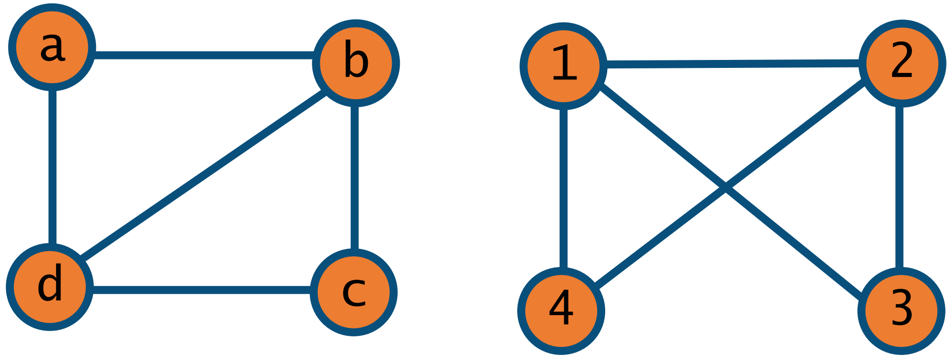
- that is, could the vertices of G_1 be renamed such that G_1 becomes G_2

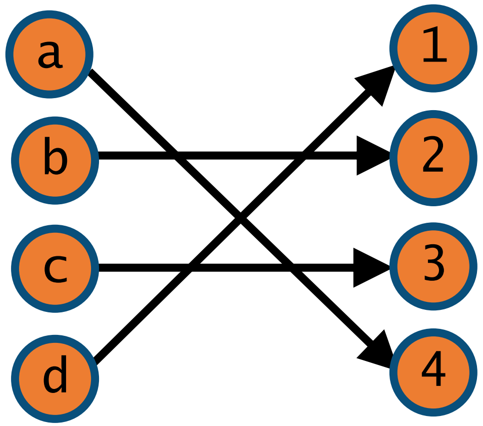
More formally:

- is there a bijection `f` from the vertex set V_1 to the vertex set V_2 such that $\{v,w\} \in E_1$ if and only if $\{f(v),f(w)\} \in E_2$

Are these graphs isomorphic? Here is a bijection

G_1 G_2





bijection

BUT: There can be multiple bijections. To find the "correct" bijection, we need to find one such that the *degree of vertices is preserved*

Isomorphic graphs

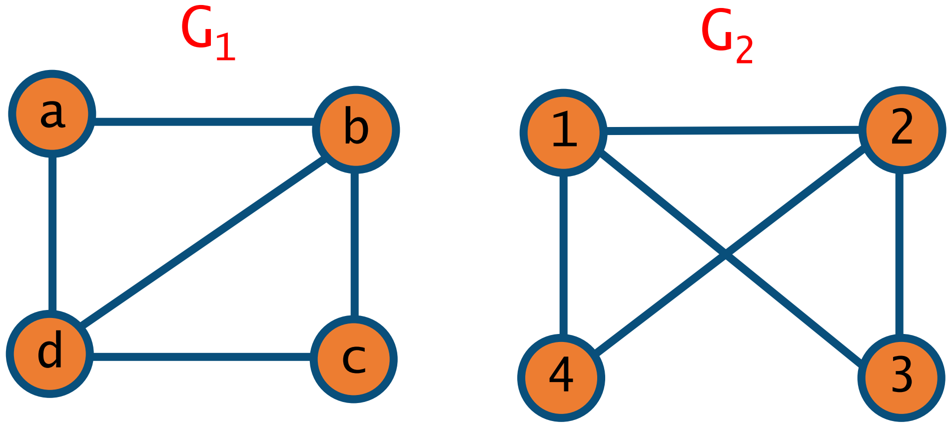
Are two graphs G_1 and G_2 isomorphic

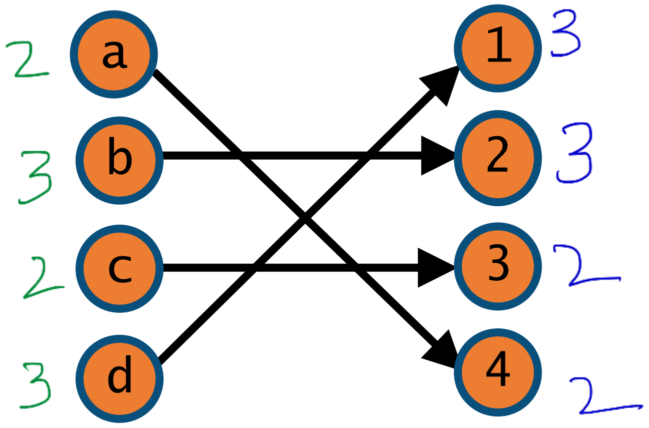
- that is, could the vertices of G_1 be renamed such that G_1 becomes G_2

More formally:

- is there a bijection `f` from the vertex set V_1 to the vertex set V_2 such that $\{v,w\} \in E_1$ if and only if $\{f(v),f(w)\} \in E_2$

Are these graphs isomorphic? Here is a bijection



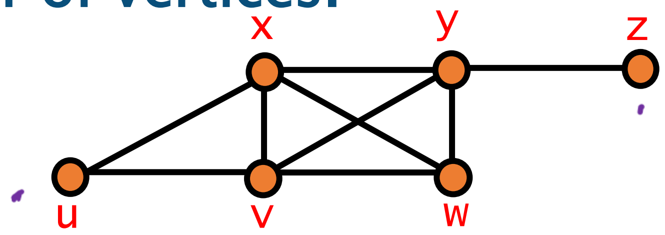


BUT: There can be multiple bijections. We need to find one such that the *degree of vertices is preserved* → These two graphs are indeed ISOMORPHIC

Connectivity

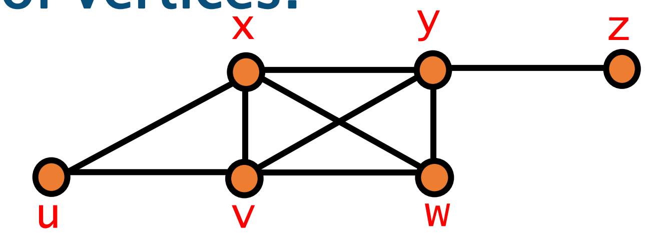
Connectivity

An undirected graph is called *connected* if there is a path between every pair of vertices.

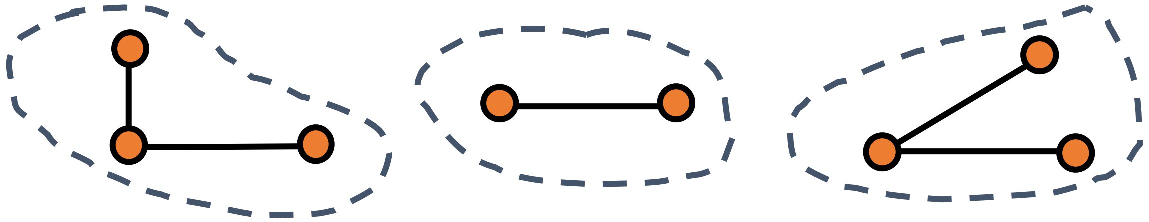


Connectivity

An undirected graph is called *connected* if there is a path between every pair of vertices.

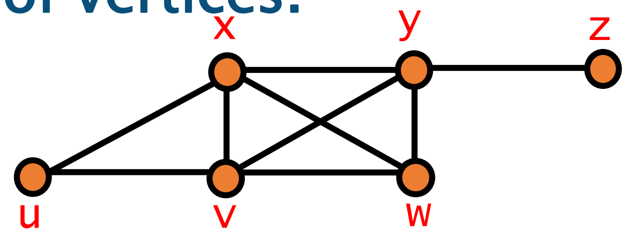


A non-connected graph has two or more connected components

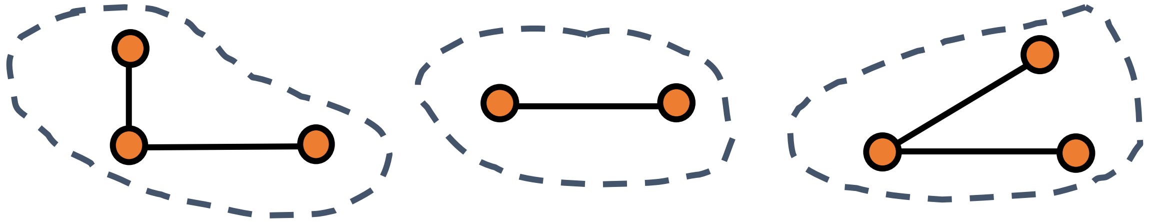


Connectivity

An undirected graph is called *connected* if there is a path between every pair of vertices.



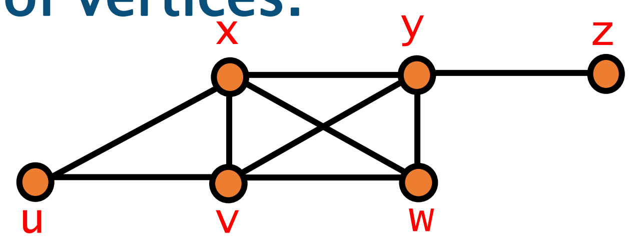
A non-connected graph has two or more connected components



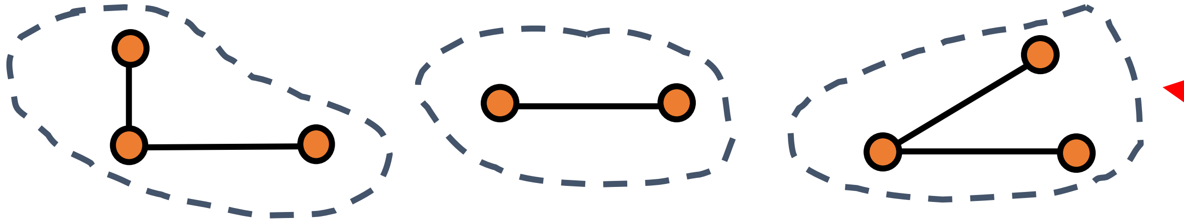
A graph is a **tree** if it is connected and **acyclic** (no cycles/circuits)

Connectivity

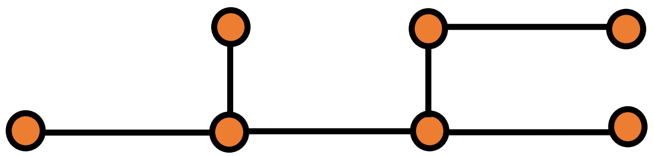
An undirected graph is called *connected* if there is a path between every pair of vertices.



A non-connected graph has two or more connected components



A graph is a **tree** if it is connected and acyclic (no cycles/circuits)



A graph is a forest if it is disconnected and components are trees (acyclic)

EXAMPLES OF USING GRAPHS FOR *MODELLING*

Graph Models:
Computer Networks

- Graphs used extensively for modelling applications a wide range of domains.

- When we build *a graph model*, we use the appropriate type of graph to capture the important features of the application.

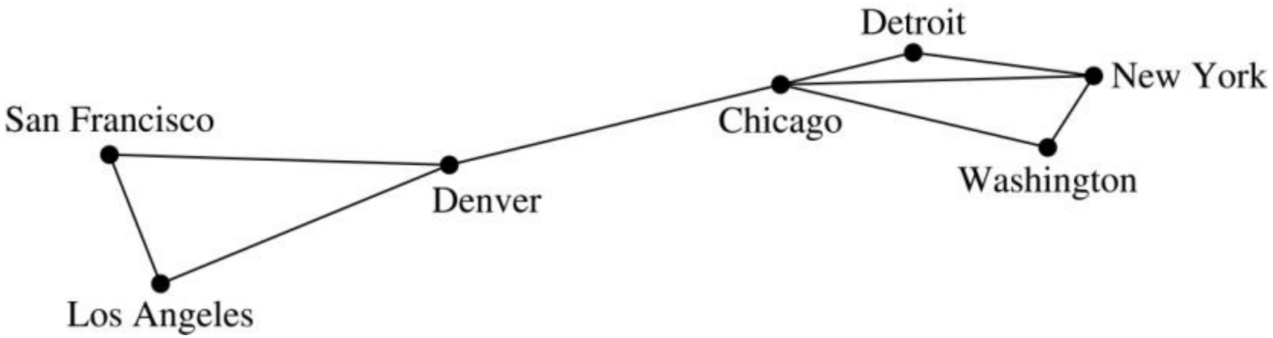
Graph Models: Computer Networks

- Graphs used extensively for modelling applications a wide range of domains.

- When we build *a graph model*, we use the appropriate type of graph to capture the important features of the application.

- **For example, computer networks can be modelled as graphs.**
  - The vertices represent data centers and the edges represent communication links.

- **To model a computer network where we are only concerned whether two data centers are connected by a communications link, we use a simple graph.**
  - This is the appropriate type of graph when we only care whether two data centers are directly linked (and not how many links there may be) and all communications links work in both directions.

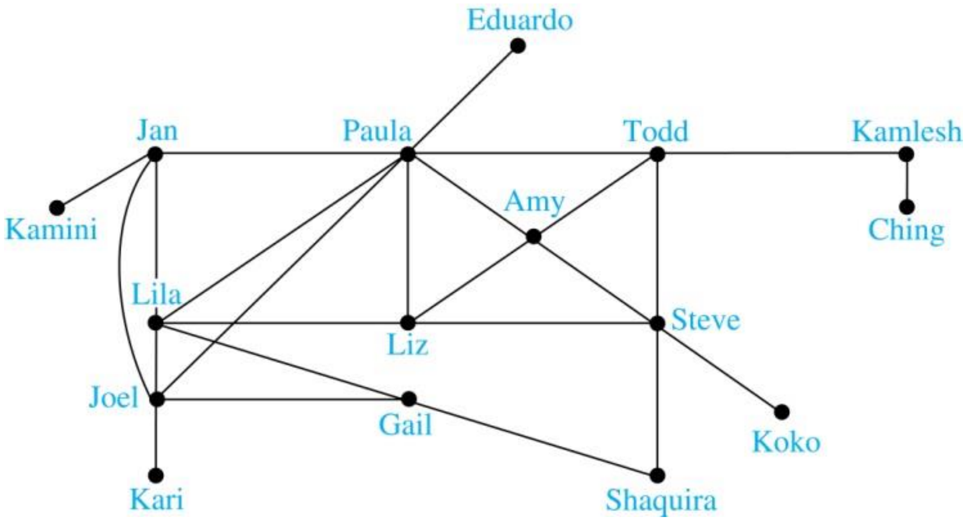


Graph Models: Social Networks

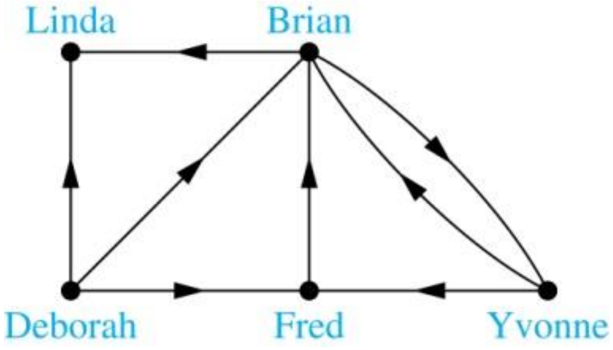
- Graphs can be used to model social structures based on different kinds of relationships between people or groups.
- In a social network, vertices represent individuals or organizations and edges represent relationships between them.
- Useful graph models of social networks include:
  - friendship graphs – undirected graphs where two people are connected if they are friends (in the real world, on Facebook, or in a particular virtual world, and so on.)
  - collaboration graphs – undirected graphs where two people are connected if they collaborate in a specific way
  - influence graphs – directed graphs where there is an edge from one person to another if the first person can influence the second person

Graph Models: Social Networks (*continued*)

Example: A friendship graph where two people are connected if they are Facebook friends.



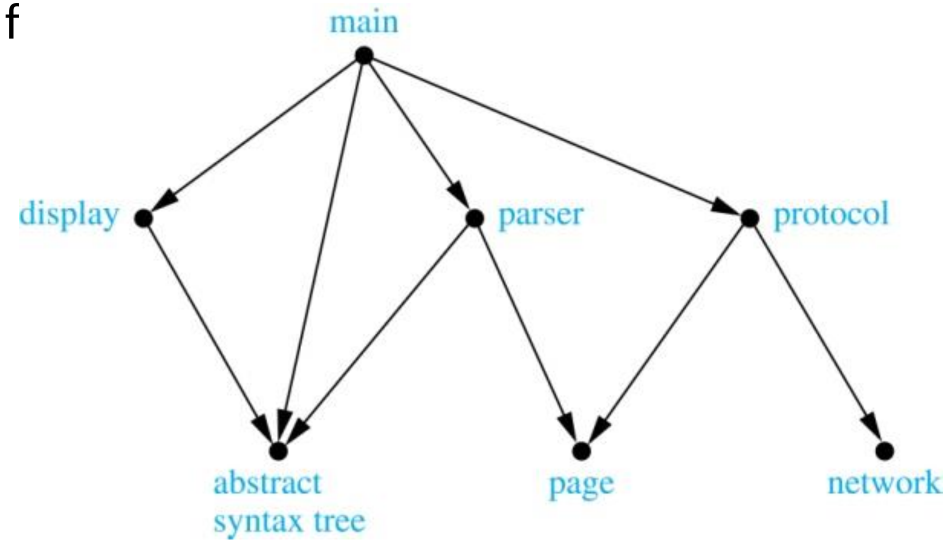
Example: An influence graph



Software Design Applications

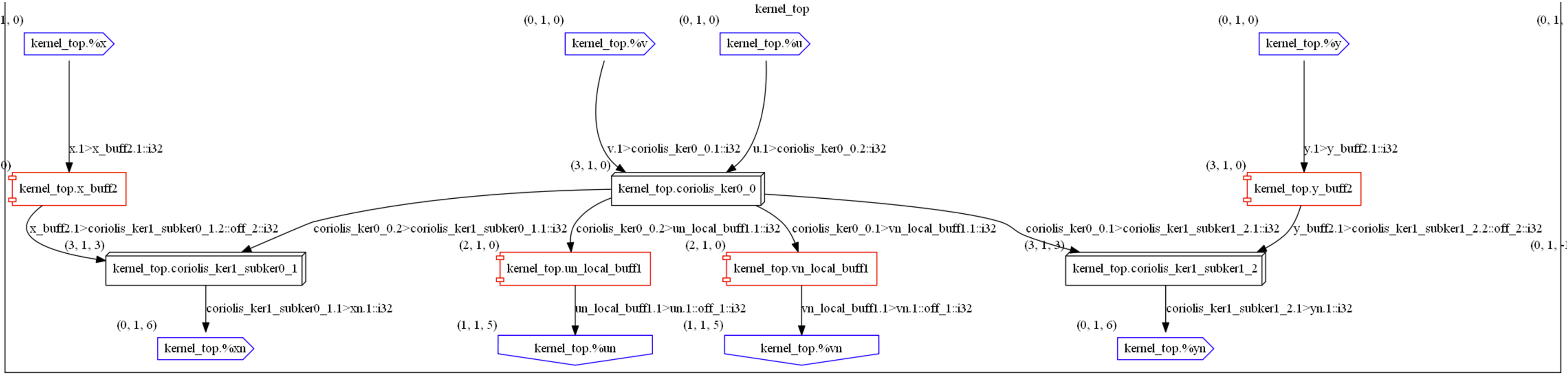
- Graph models are extensively used in software design.
- When a top-down approach is used to design software, the system is divided into modules, each performing a specific task.
- We use a *module dependency graph* to represent the dependency between these modules. These dependencies need to be understood before coding can be done.
  - In a module dependency graph vertices represent software modules and there is an edge from one module to another if the second module depends on the f

Example: The dependencies between the seven modules in the design of a web browser are represented by this module dependency graph.



Applications in Research

- **E.g.: My research!**
  - Using Dataflow graphs (DFGs), for writing compilers for Dataflow Architectures



There are many more graph problems…

shortest path, connectivity, minimum spanning tree, maximum flow,
stable matching, dominating set, feedback vertex set,
minimum maximal matching, partitioning into triangles,
partitioning into cliques, partitioning into perfect matchings,
covering by cliques, bandwidth, subgraph isomorphism,
largest common subgraph, graph grundy numbering,
weighted diameter, graph partitioning, steiner tree in graphs,
maxcut, network reliability, travelling salesman problem,
chinese postman for mixed graphs, rural postman,
minimum broadcast time, min-sum multicentre,
stable matching with ties and incomplete lists, …