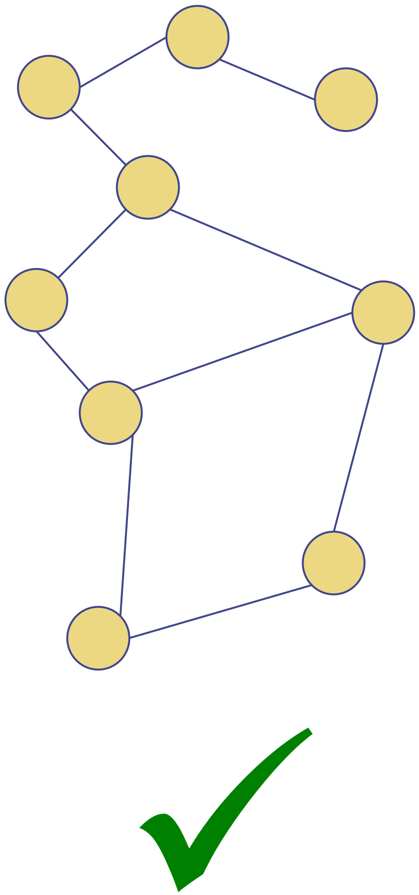
CS 350 Algorithms and Complexity

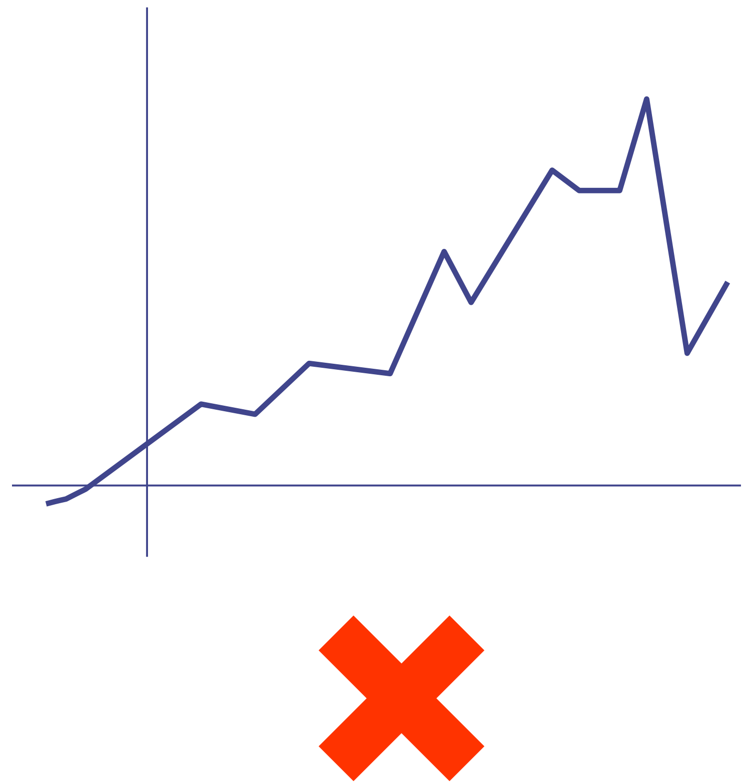
Fall 2018

Lecture 7: Graphs (Introduction and Terminology)

Paul Doliotis - Adjunct Assistant Professor
Department of Computer Science
Portland State University
*(*presentation includes some material originally created by Prof. Mark P. Jones)*

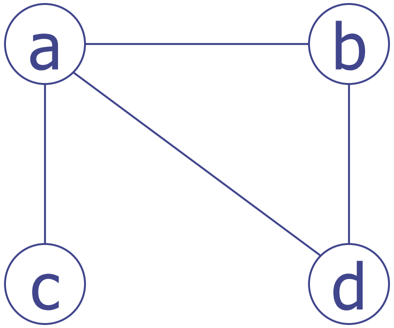
Graphs, not Graphs!





Graphs:

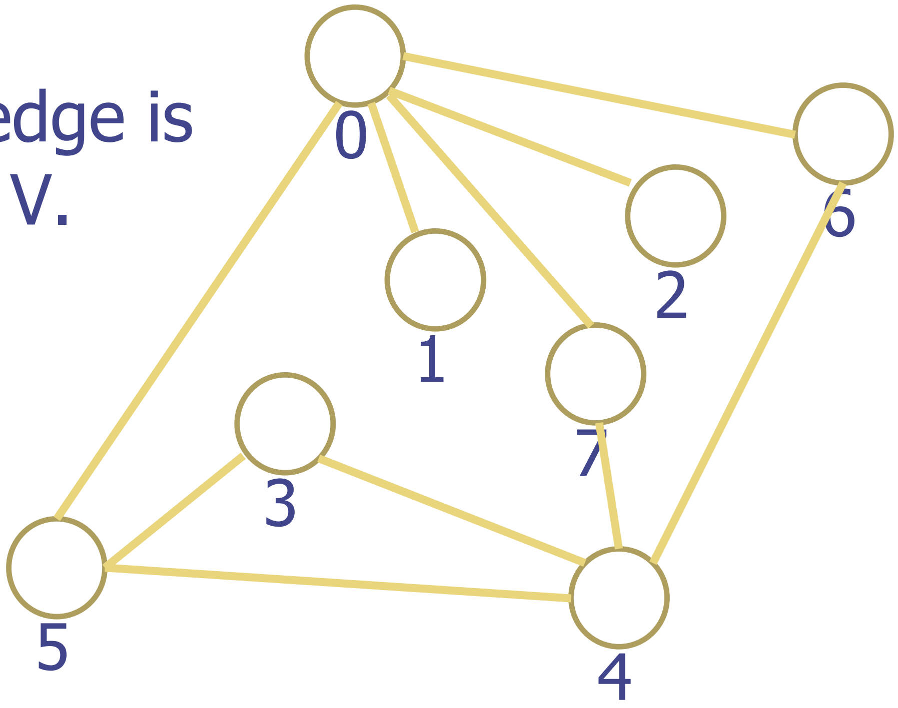
- A *graph* is a pair (V,E) where
  - V is a set of vertices;
  - E is a set of edges {u,v}, where u,v are distinct vertices from V.
- For example:

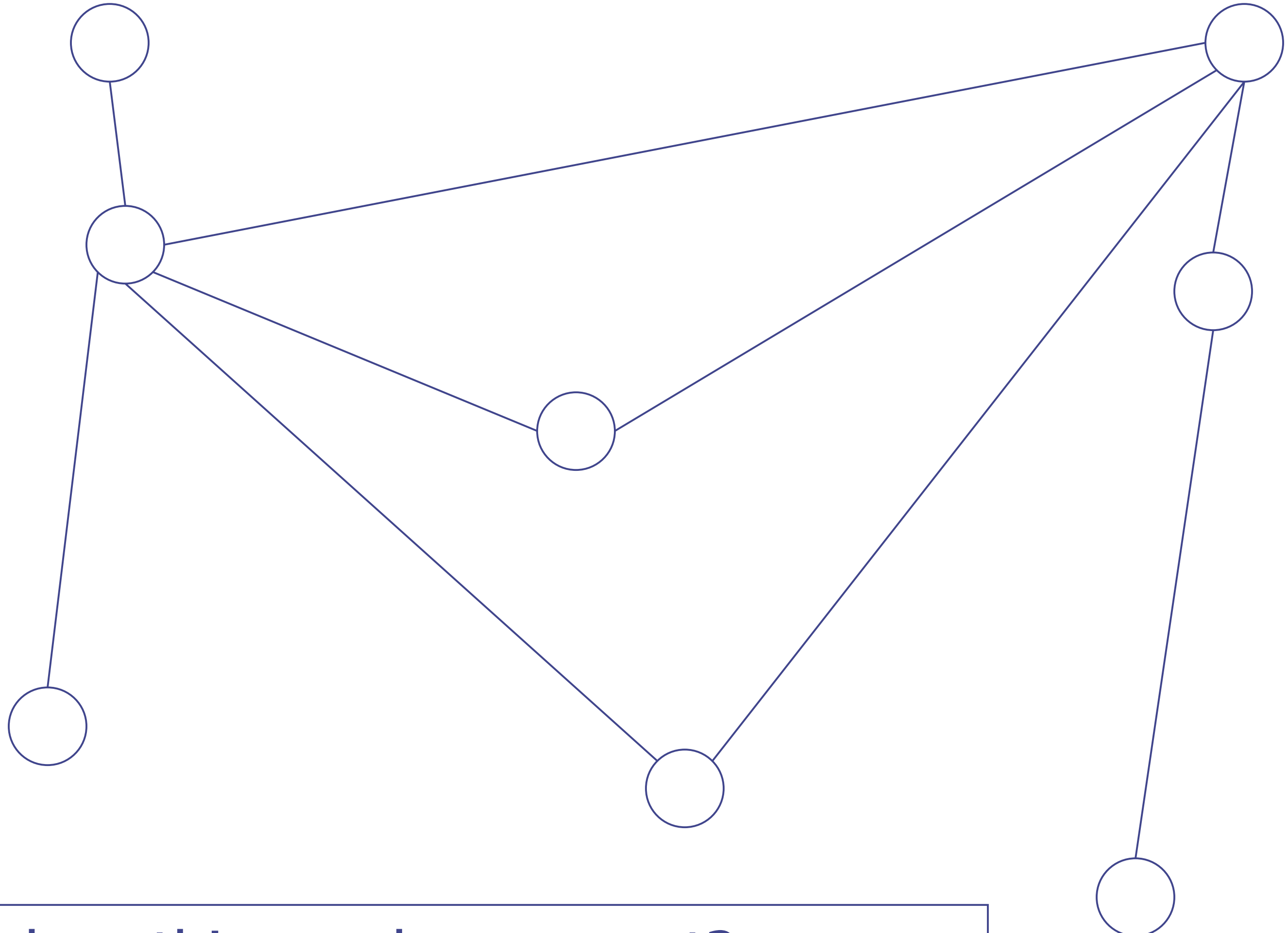


- G = ({a,b,c,d}, {{a,b}, {a,c}, {a,d}, {b,d}})
- Examples: computer networks, street layout, etc…

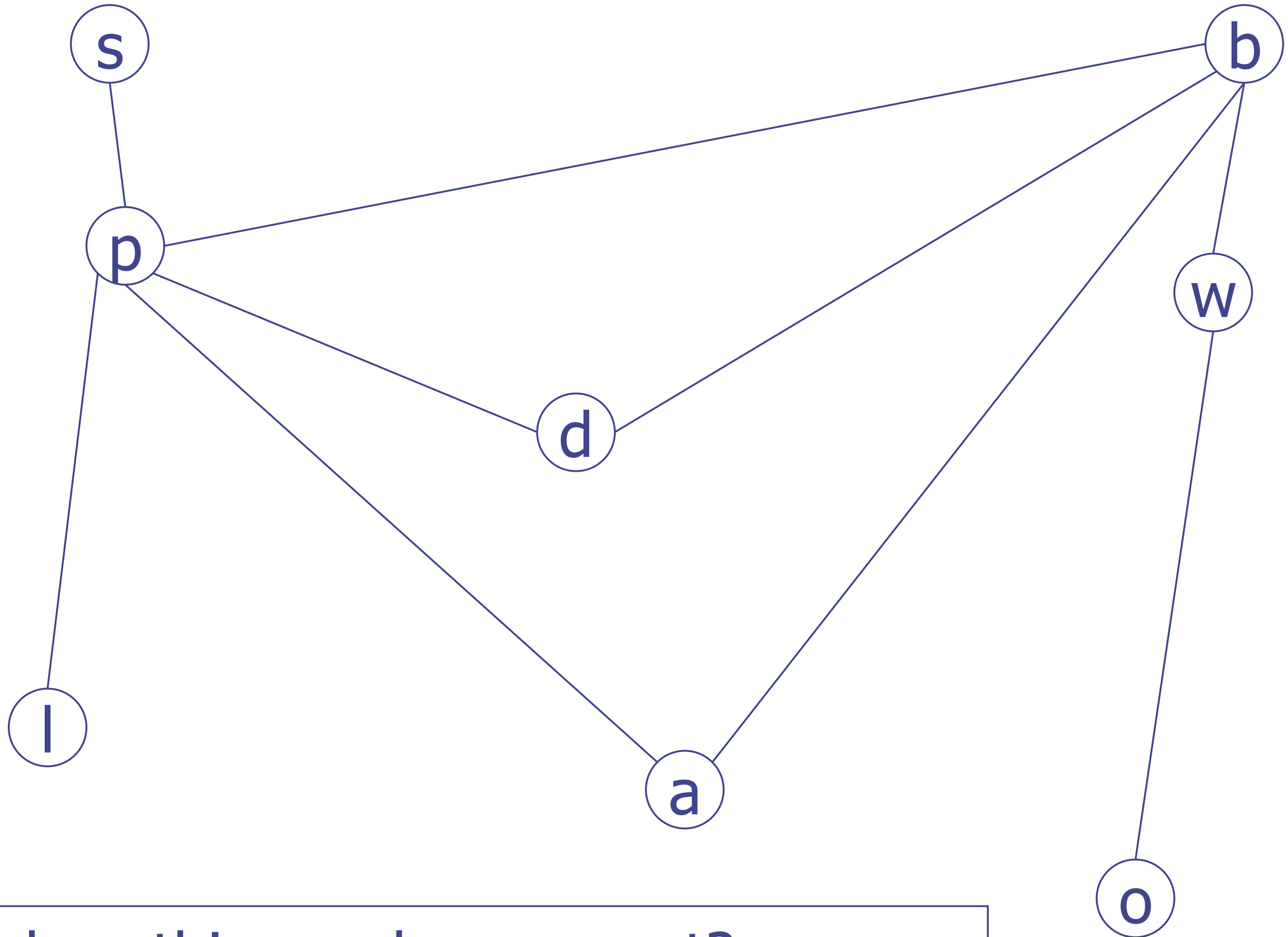
Example: of an Undirected Graph

- A graph is formally defined as:
  - A set V of vertices.
  - A set E of edges. Each edge is a pair of two vertices in V.
- What is the set of vertices on the graph shown here?
  - {0, 1, 2, 3, 4, 5, 6, 7}
- What is the set of edges?
  - {(0,1), (0,2), (0,5), (0,6), (0, 7), (3, 4), (3, 5), (4, 5), (4, 6), (4,7)}.

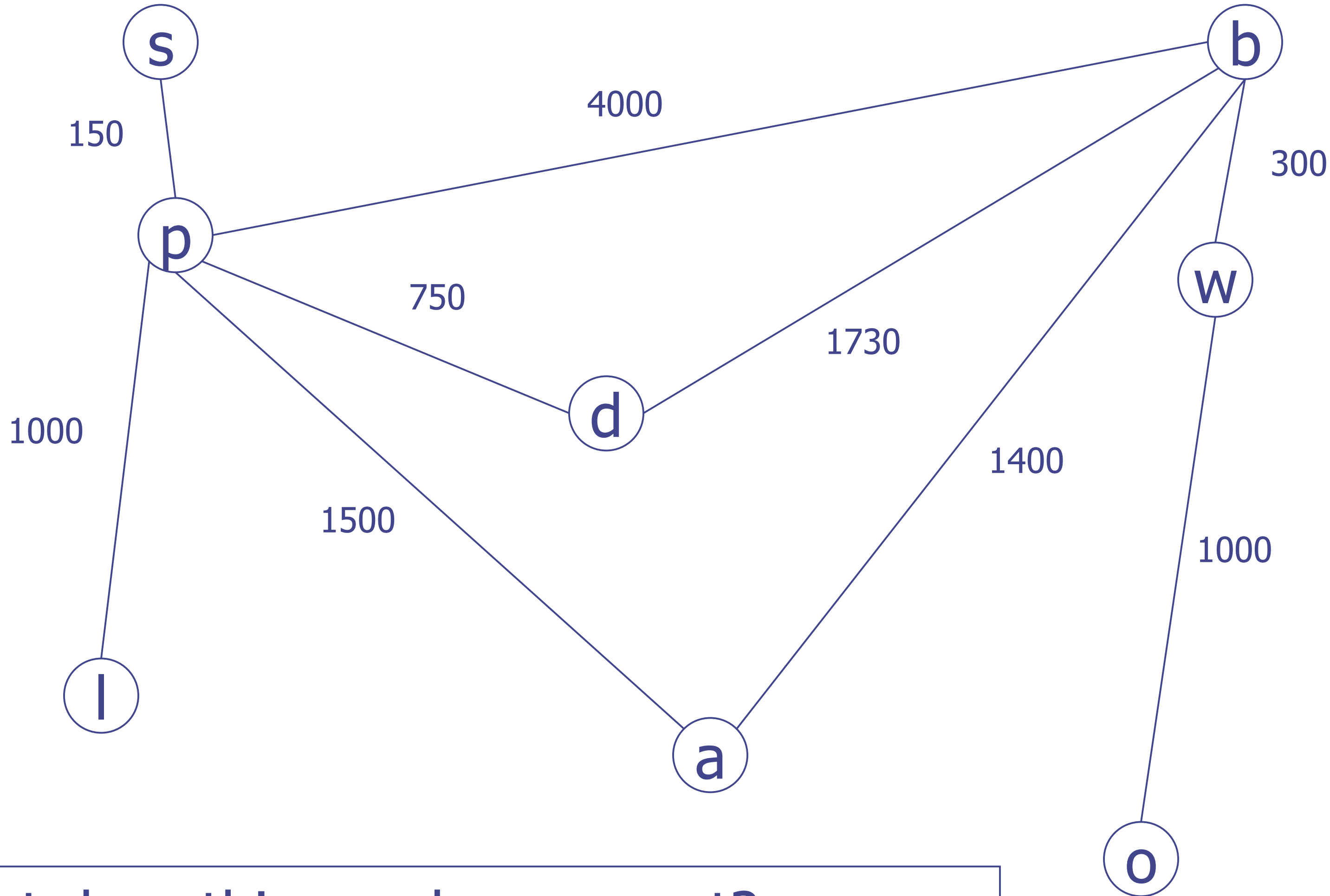




What does this graph represent?



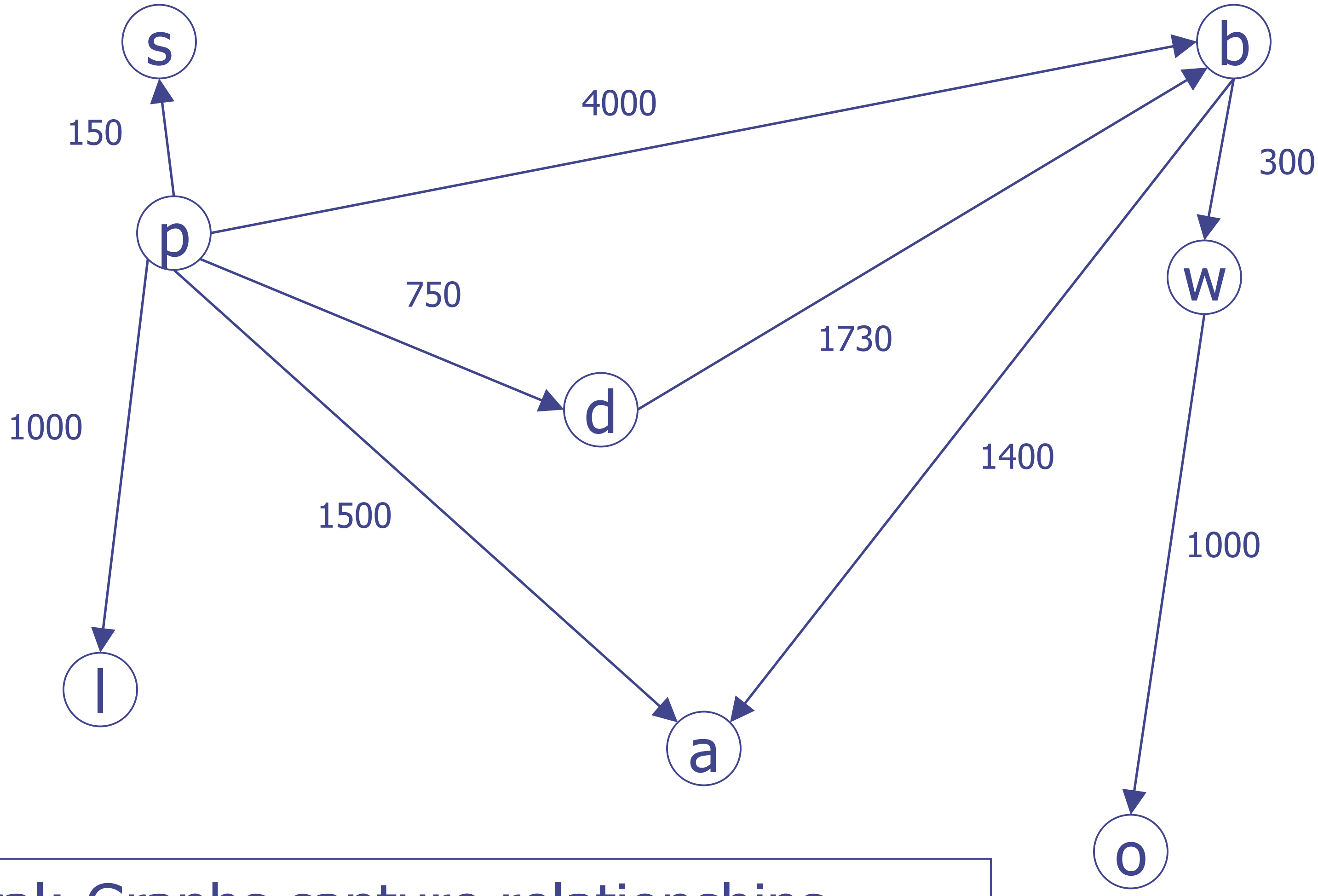
What does this graph represent?



What does this graph represent?

Directed graphs:

- A directed graph is a pair (V,E) where
  - V is a set of vertices;
  - E is a set of edges, each of which is an ordered pair (u,v).
- Directed edges are the “one-way streets” of graph theory.
- Examples: Flowcharts, call graphs, dependency charts, state machines, …

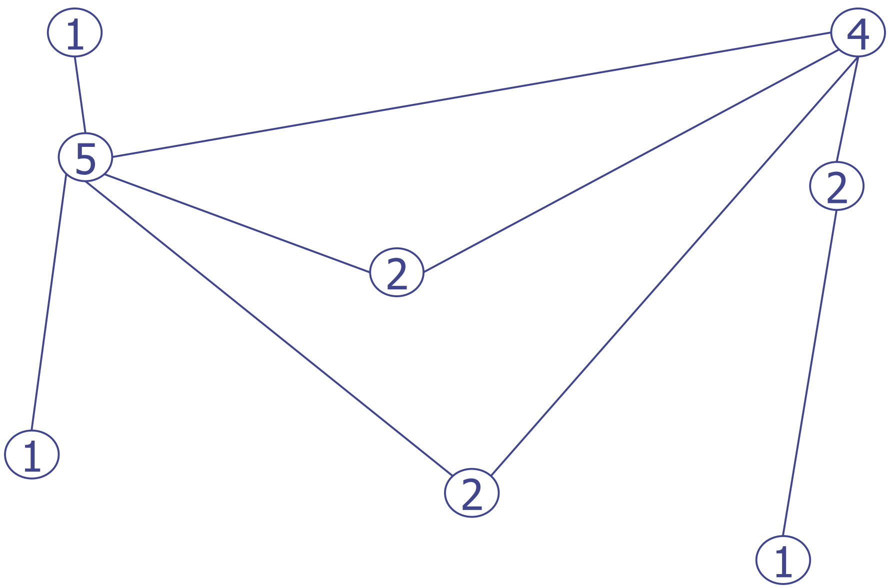


Moral: Graphs capture relationships between objects, while abstracting away from unnecessary details.

- There are a range of algorithms for manipulating graphs from a fairly general abstract point of view.
  - Is there a way to get from one vertex to another?
  - If so, what is the quickest/cheapest route?
  - Are there any alternatives?
  - Etc…
- Many of these turn out to be very useful in practical applications.
- But first we need to establish a vocabulary for talking about graphs, without restricting our attention to any *specific* application.

Degree:

- An edge e is said to be incident on a vertex v if v is one of the vertices in e.

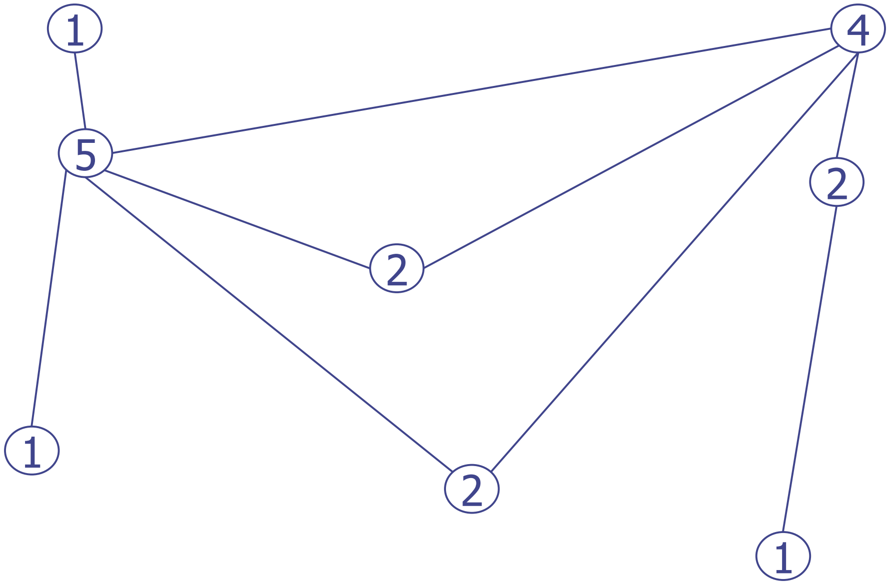


Degree:

- An edge e is said to be <u>incident</u> on a vertex v if v is one of the vertices in e.

- The <u>*degree*</u> of a vertex is the number of edges that are incident on it.

$$\sum_{v \in V} \deg(v) = 2|E|$$



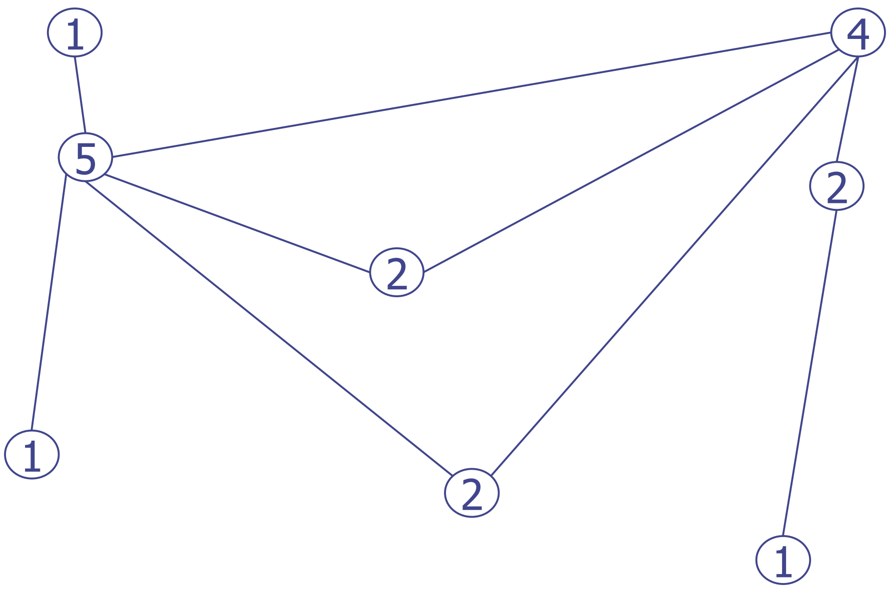
Degree:

- An edge e is said to be <u>incident</u> on a vertex v if v is one of the vertices in e.

- The <u>*degree*</u> of a vertex is the number of edges that are incident on it.

$$\sum_{v \in V} \deg(v) = 2|E|$$

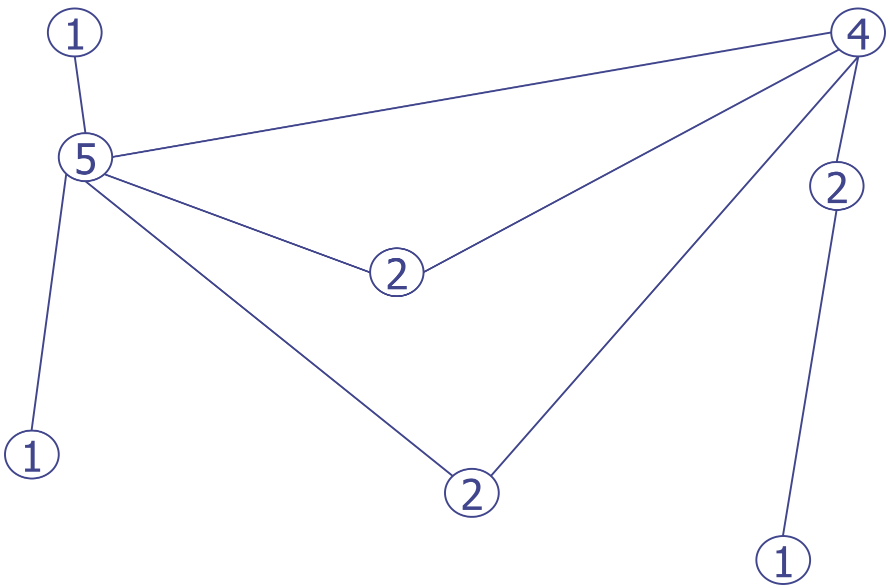
Why?



Degree:

- An edge e is said to be <u>incident</u> on a vertex v if v is one of the vertices in e.

- The <u>*degree*</u> of a vertex is the number of edges that are incident on it.



$$\sum_{v \in V} \deg(v) = 2|E|$$

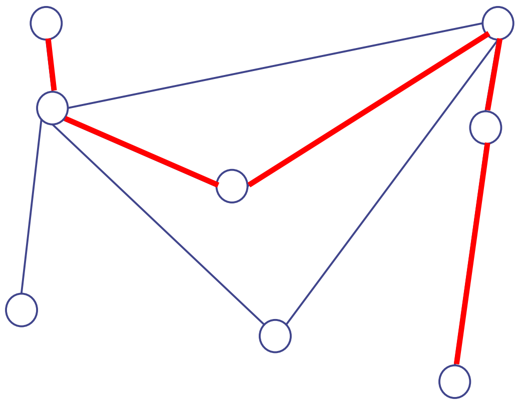
Why?

One edge

Two endpoints

Paths and connectedness:

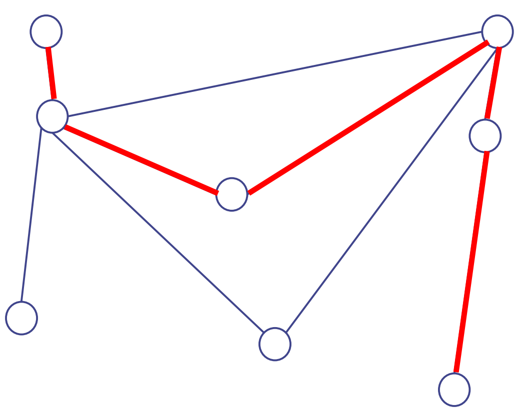
- A <u>path</u> (of length k) in a graph $G=(V,E)$ is a sequence of vertices $(v_0,v_1,\ldots,v_k)$ such that each $\{v_i,v_{i+1}\}\in E$. (We say that v_k is *reachable* from v_0.)



Paths and connectedness:

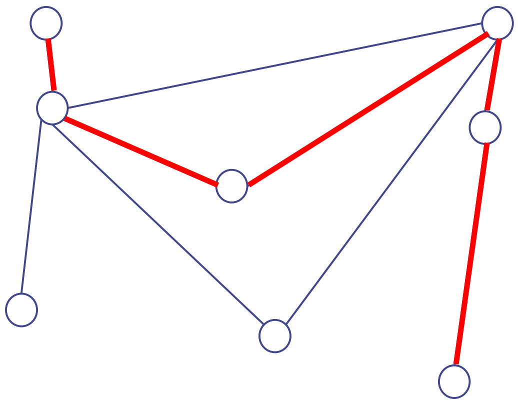
- A <u>path</u> (of length k) in a graph G=(V,E) is a sequence of vertices $(v_0,v_1,\ldots,v_k)$ such that each $\{v_i,v_{i+1}\}\in E$. (We say that v_k is *reachable* from v_0.)

- A path is <u>simple</u> if all of the v_i are distinct.



Paths and connectedness:

- A path (of length k) in a graph $G=(V,E)$ is a sequence of vertices $(v_0,v_1,\ldots,v_k)$ such that each $\{v_i,v_{i+1}\}\in E$. (We say that v_k is *reachable* from v_0.)
- A path is simple if all of the v_i are distinct.
- The graph G is connected if there is a path between any pair of vertices in V.

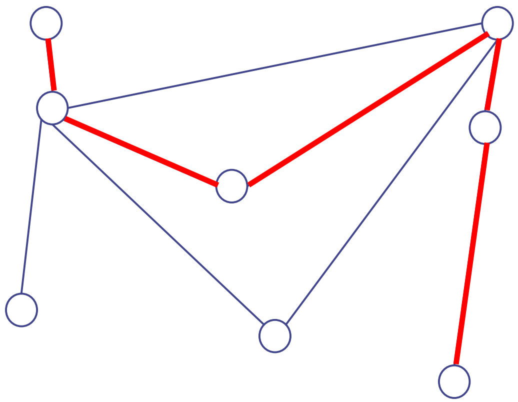


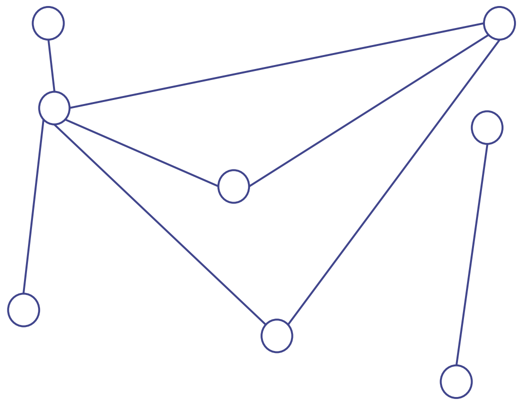
Paths and connectedness:

- A <u>path</u> (of length k) in a graph $G=(V,E)$ is a sequence of vertices $(v_0,v_1,\ldots,v_k)$ such that each $\{v_i,v_{i+1}\}\in E$. (We say that v_k is *reachable* from v_0.)

- A path is <u>simple</u> if all of the v_i are distinct.

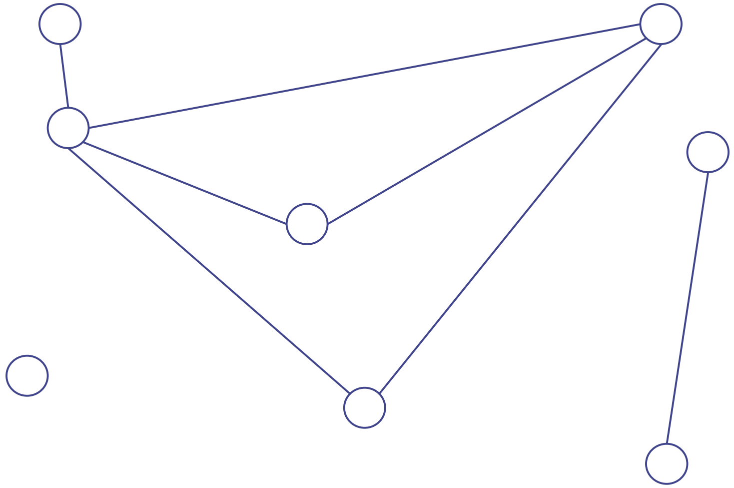
- The graph G is <u>connected</u> if there is a path between any pair of vertices in V.





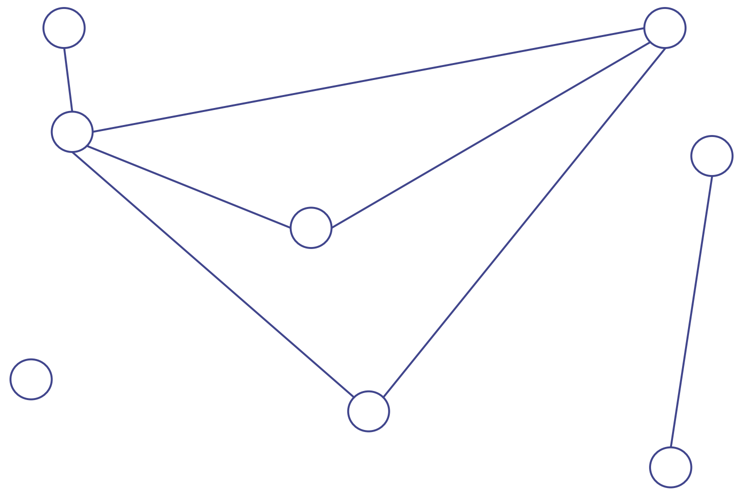
Connected components:

- Every graph can be viewed as a collection of *connected components*:



Connected components:

- Every graph can be viewed as a collection of *connected components*:

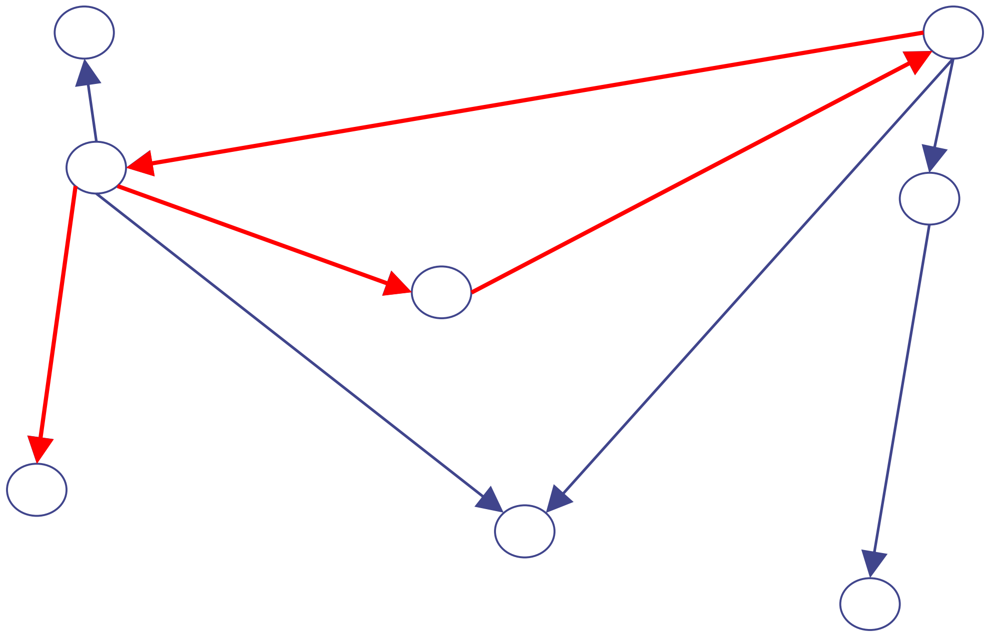


- Vertices u and v are in the same component if, and only if, there is a path from u to v.

Paths in directed graphs:

- A <u>path</u> (of length k) in a directed graph G=(V,E) is a sequence of vertices $(v_0,v_1,\ldots,v_k)$ such that each $(v_i,v_{i+1})\in E$.

- Note: all edges point "forwards"



Strong connectivity:

- We say that v_k is <u>reachable</u> from v_0 if there is a path from v_0 to v_k.

Strong connectivity:

- We say that v_k is <u>reachable</u> from v_0 if there is a path from v_0 to v_k.

- In a directed graph, it does not necessarily follow that the reverse also holds; i.e., that v_0 is reachable from v_k.

Strong connectivity:

- We say that v_k is reachable from v_0 if there is a path from v_0 to v_k.

- In a directed graph, it does not necessarily follow that the reverse also holds; i.e., that v_0 is reachable from v_k.

- We say that two vertices u and v are strongly connected if there is a path from u to v, *and* a path from v to u (clearly strong connectivity is examined for directed graphs).

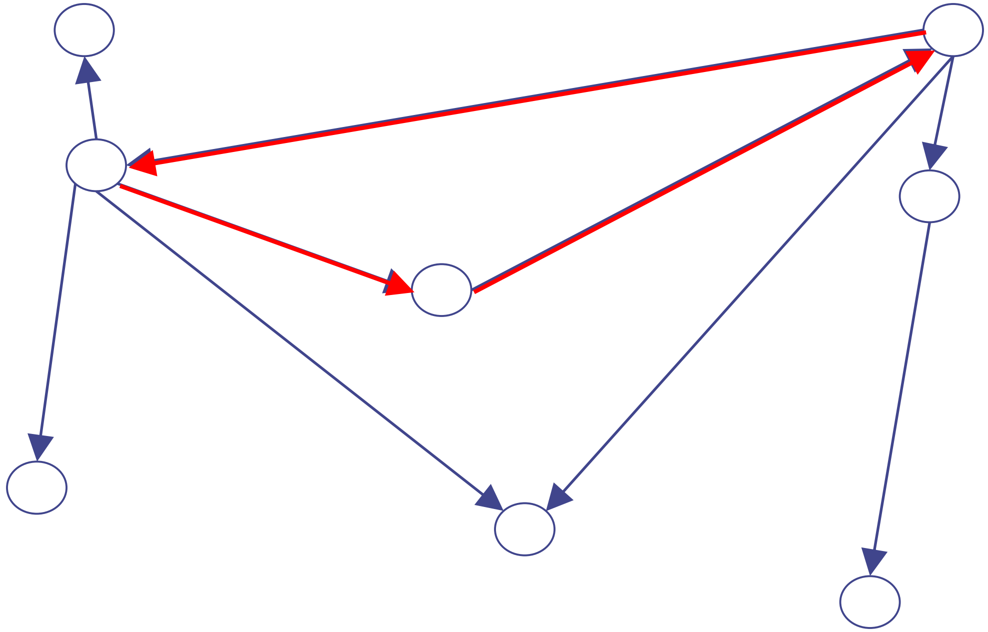
Strong connectivity:

- We say that v_k is reachable from v_0 if there is a path from v_0 to v_k.
- In a directed graph, it does not necessarily follow that the reverse also holds; i.e., that v_0 is reachable from v_k.
- We say that two vertices u and v are strongly connected if there is a path from u to v, *and* a path from v to u (clearly strong connectivity is examined for directed graphs).
- What about strong connectivity and undirected graphs?

Strongly connected components:

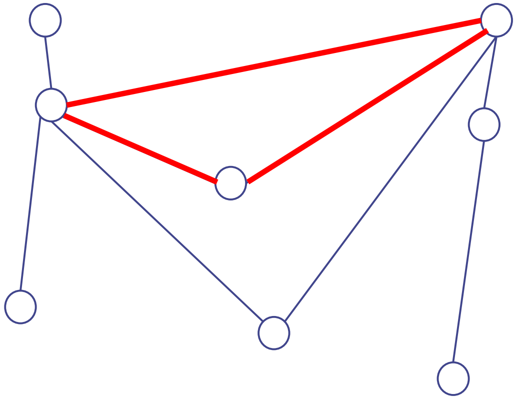
- We can break up a directed graph into its <u>strongly connected components</u>.

- Vertices u and v are in the same component if, and only if, there is a path from u to v.



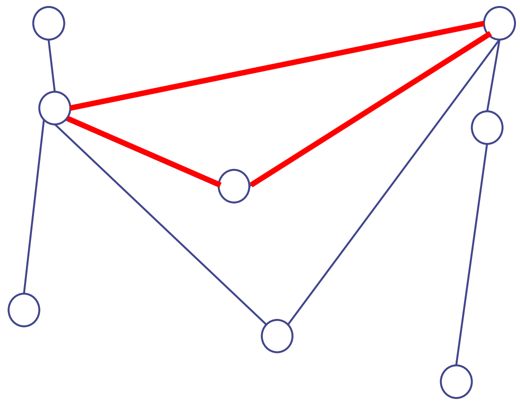
Cycles:

- A cycle is a path of length>0 (or length>2 in an undirected graph) in which the first and the last vertex are the same.



Cycles:

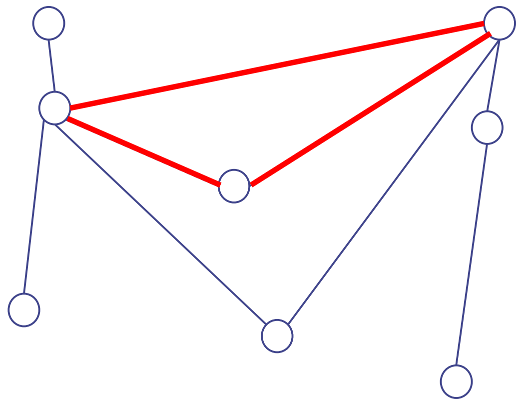
- A cycle is a path of length>0 (or length>2 in an undirected graph) in which the first and the last vertex are the same.



- A cycle is simple if all of the vertices, except the first and last, are distinct.

Cycles:

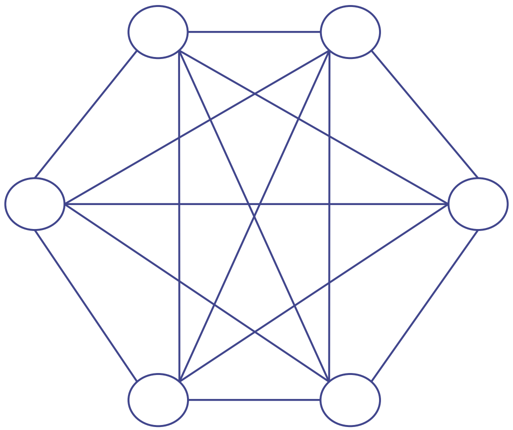
- A cycle is a path of length>0 (or length>2 in an undirected graph) in which the first and the last vertex are the same.



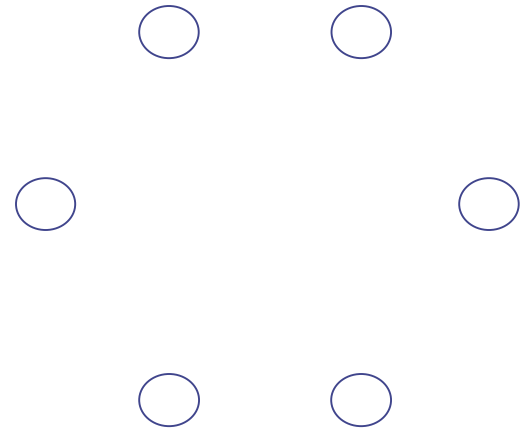
- A cycle is simple if all of the vertices, except the first and last, are distinct.

- A graph is acyclic if it does not contain cycles.

From connected to cyclic



A spectrum of graphs



What lies in between

?

A complete graph: as connected and as cyclic as can be!

A discrete graph: as disconnected and as acyclic as can be!

Trees!

- A <u>tree</u> is a connected, acyclic graph.

Trees!

- A tree is a connected, acyclic graph.
  - Add an edge to a tree and you create a cycle.

Trees!

- A tree is a connected, acyclic graph.
  - Add an edge to a tree and you create a cycle.
  - Remove an edge from a tree and it becomes disconnected.

Trees!

- A tree is a connected, acyclic graph.
  - Add an edge to a tree and you create a cycle.
  - Remove an edge from a tree and it becomes disconnected.
- These follow from: there is a *unique* simple path between any given pair of vertices in a tree:

Trees!

- A <u>tree</u> is a connected, acyclic graph.
  - Add an edge to a tree and you create a cycle.
  - Remove an edge from a tree and it becomes disconnected.
- These follow from: there is a *unique* simple path between any given pair of vertices in a tree:
  - A tree is connected, so there is at least one such path.

Trees!

- A tree is a connected, acyclic graph.
  - Add an edge to a tree and you create a cycle.
  - Remove an edge from a tree and it becomes disconnected.
- These follow from: there is a *unique* simple path between any given pair of vertices in a tree:
  - A tree is connected, so there is at least one such path.
  - If there are two different paths from some u to v, then we can join them together to make a cycle; but this is impossible because trees are acyclic.

Trees and Graphs

- Are trees graphs?
  - Always?
  - Sometimes?
  - Never?
- Are graphs trees?
  - Always?
  - Sometimes?
  - Never?

Trees and Graphs

- All trees are graphs.
- Some graphs are trees, some graphs are not trees.
- What is the distinguishing characteristic of trees?
- What makes a graph a tree?

Trees and Graphs

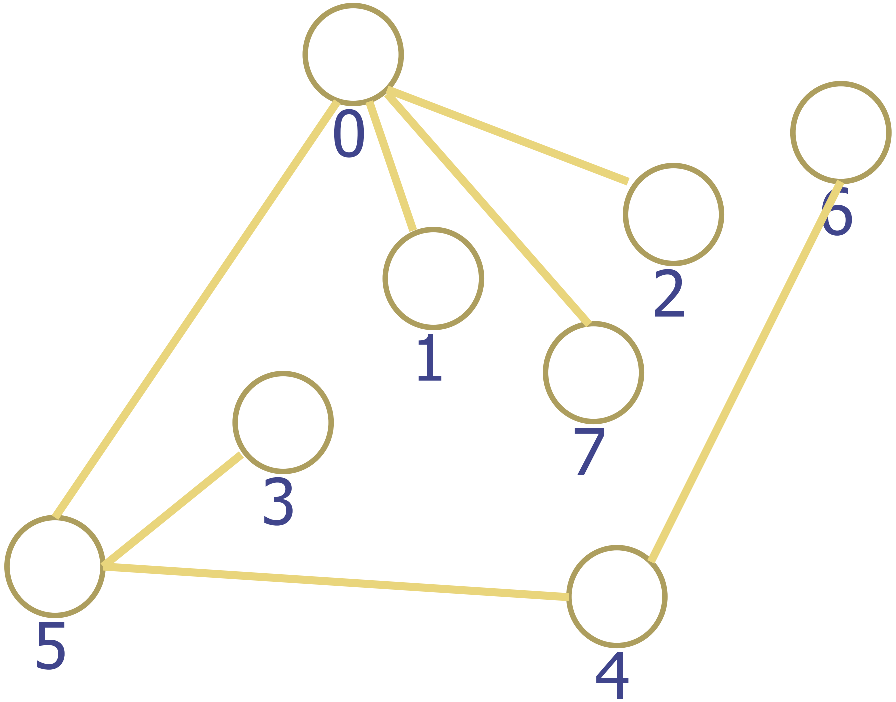
- All trees are graphs.
- Some graphs are trees, some graphs are not trees.
- What is the distinguishing characteristic of trees?
  - What makes a graph a tree?

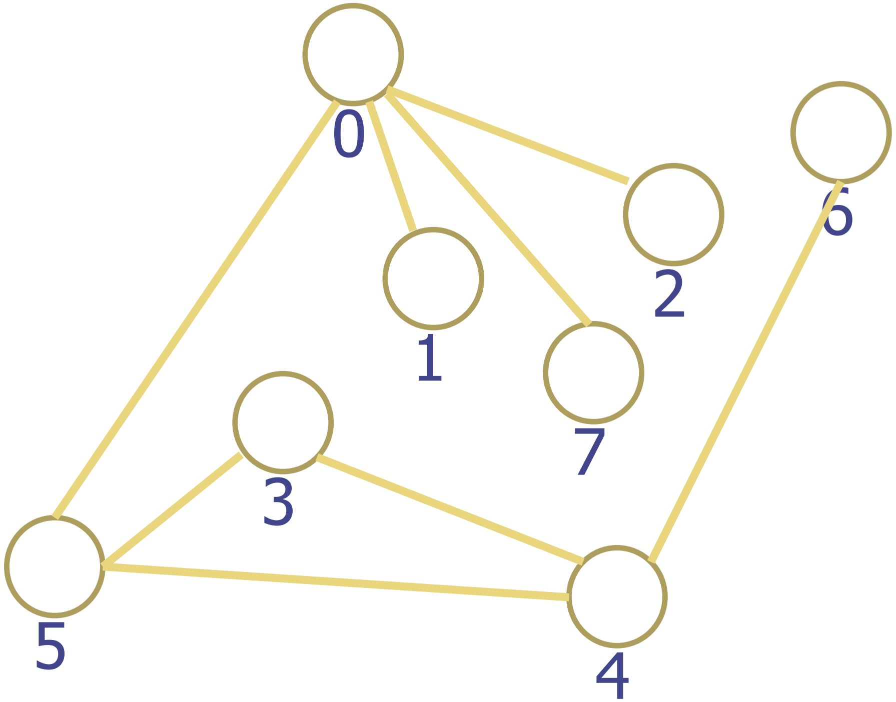
Trees and Graphs

- A tree is a graph such that any two nodes (vertices) are connected by precisely one path.
  - If you can find two nodes that are **not** connected by any path, then the graph is not a tree.
  - If you can find two nodes that are connected to each other by more than one path, then the graph is **not** a tree.

Example

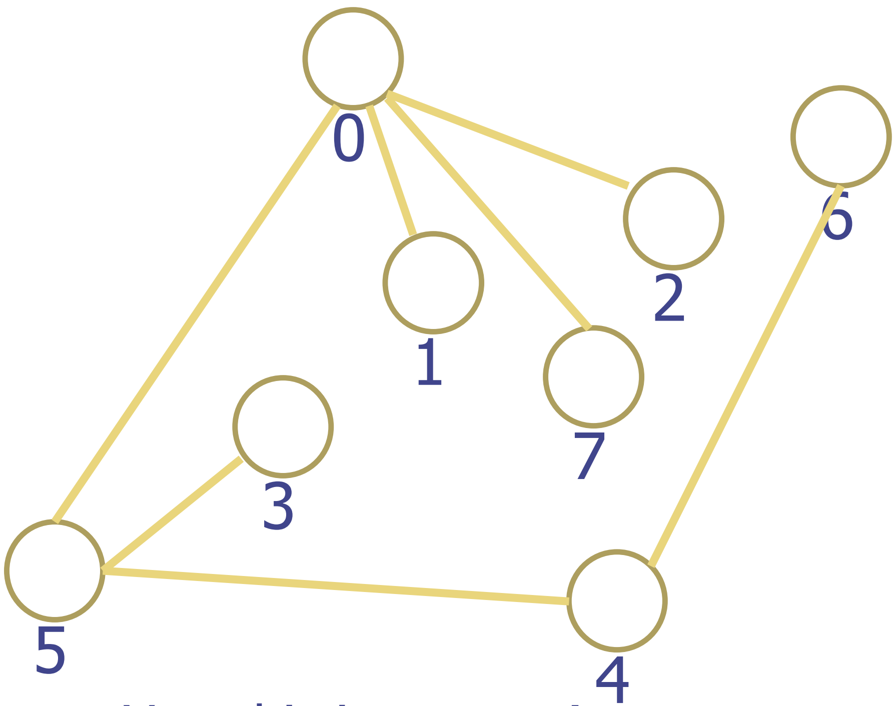
- Are these graphs trees?



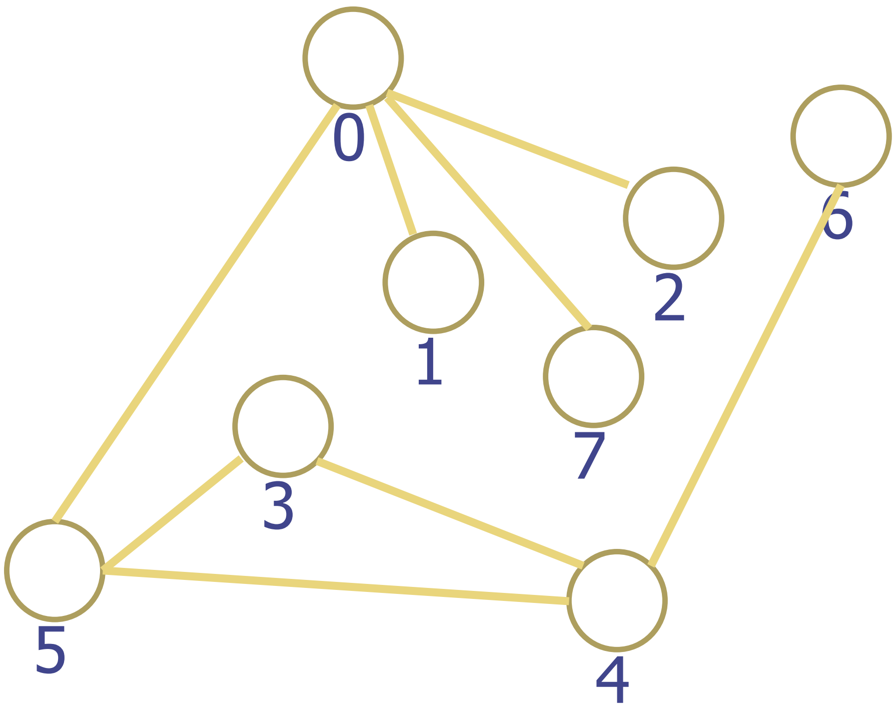


Example

- Are these graphs trees?

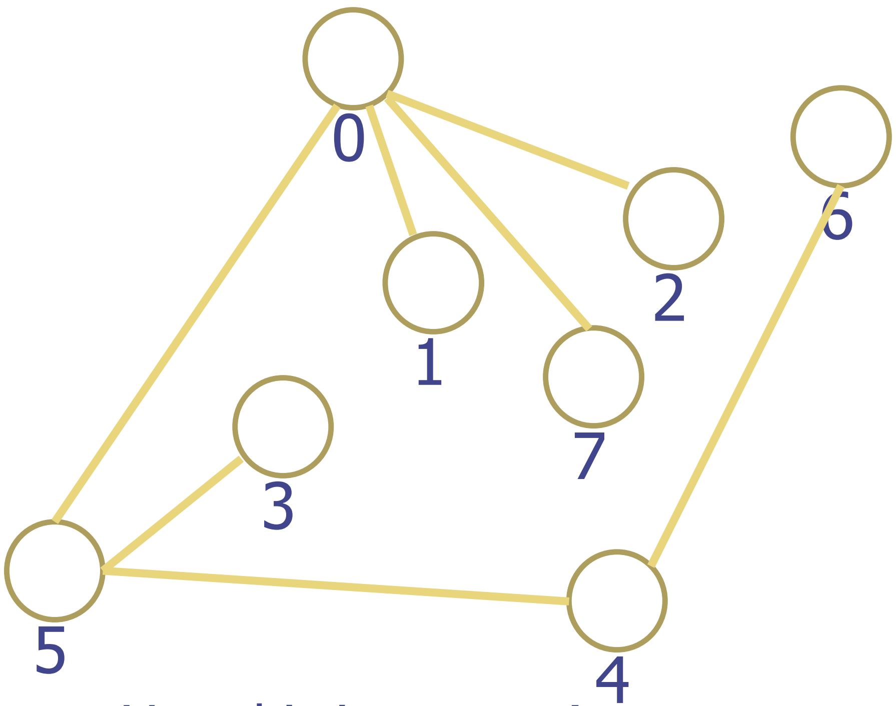


Yes, this is a tree. Any two vertices are connected by exactly one path.

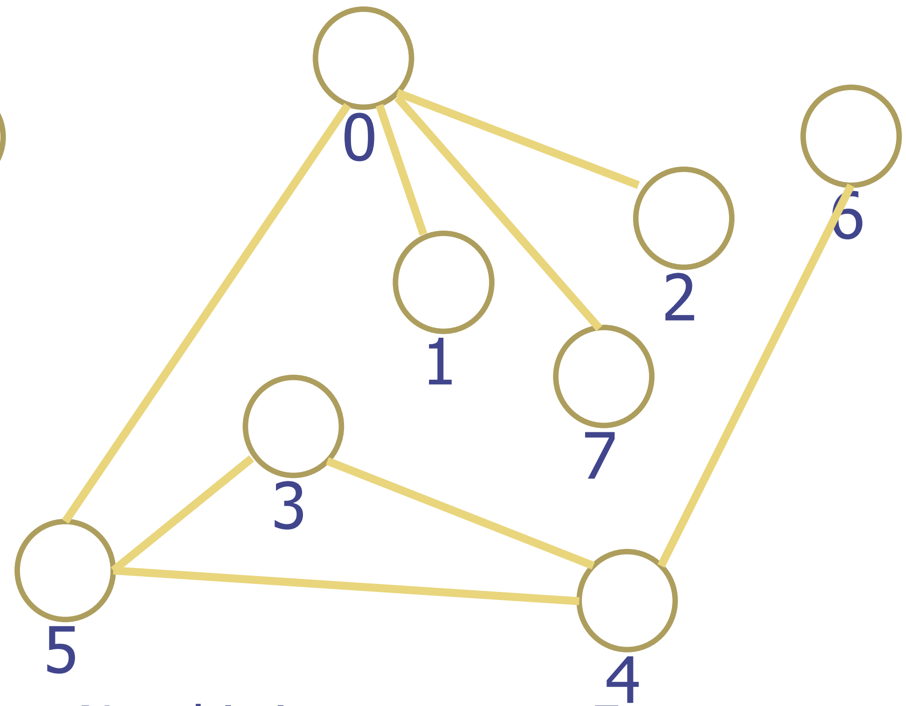


Example

- Are these graphs trees?



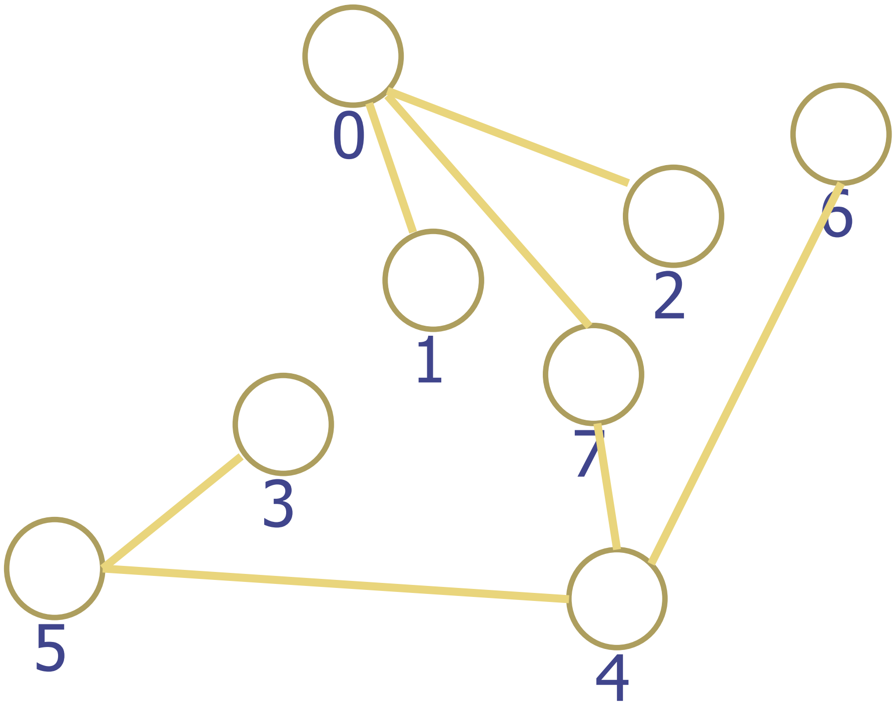
Yes, this is a tree. Any two vertices are connected by exactly one path.

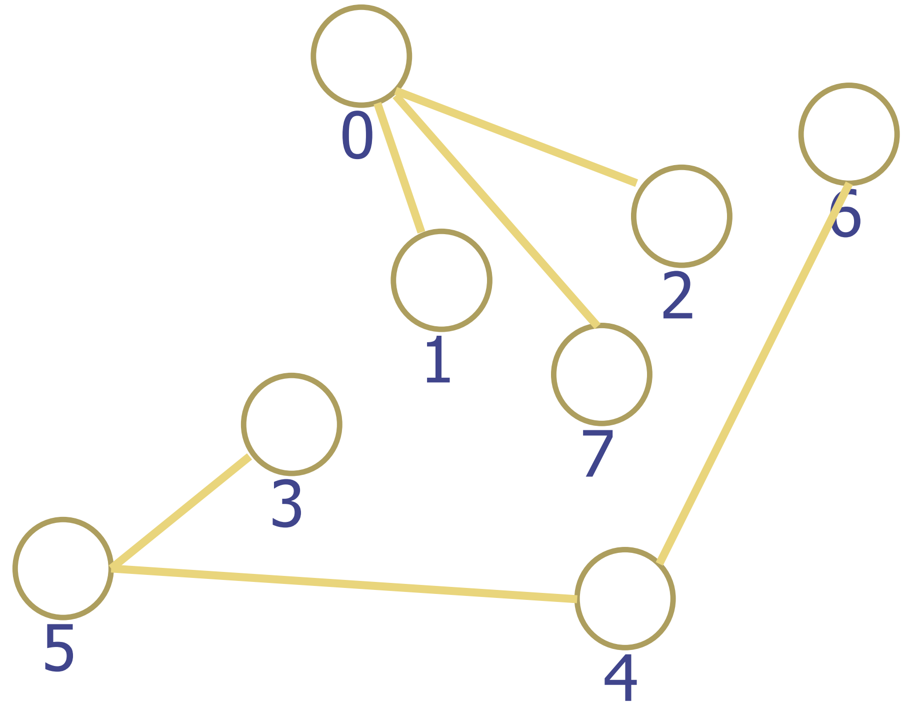


No, this is not a tree. For example, there are two paths connecting node 5 to node 4.

Example

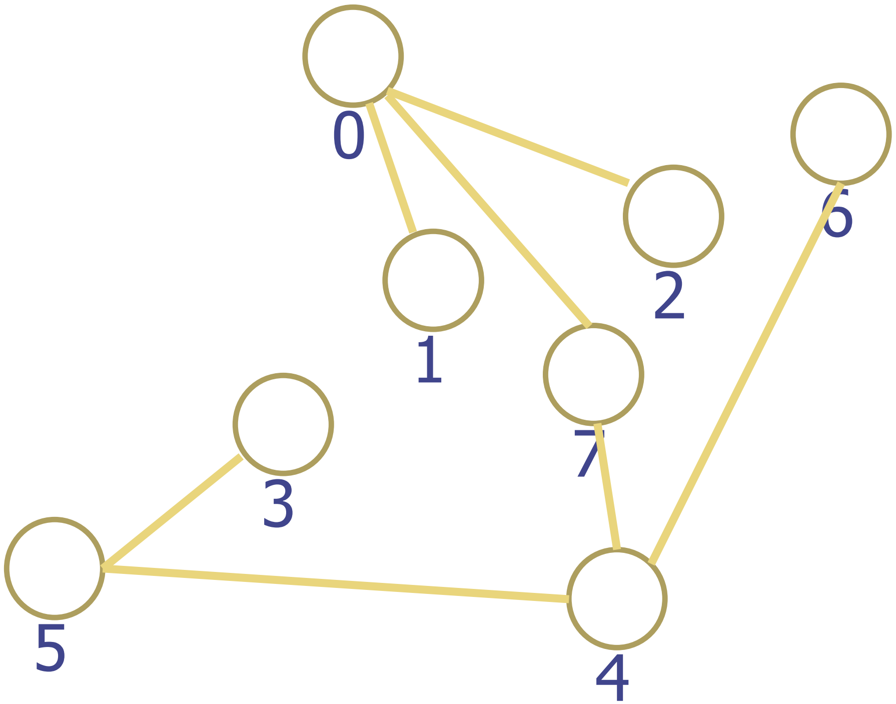
- Are these graphs trees?

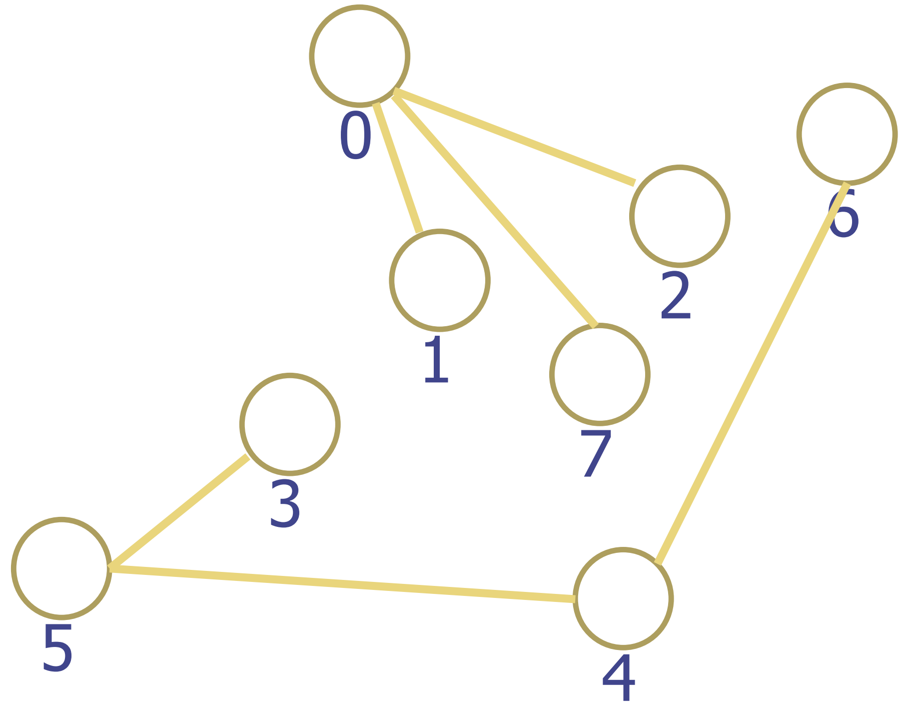




Example

- Are these graphs trees?

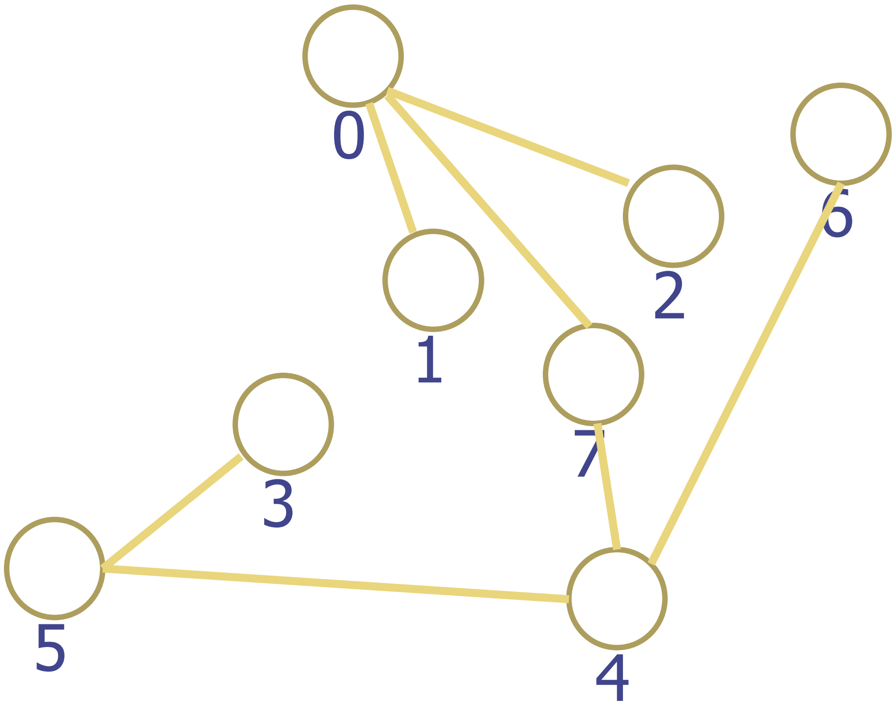




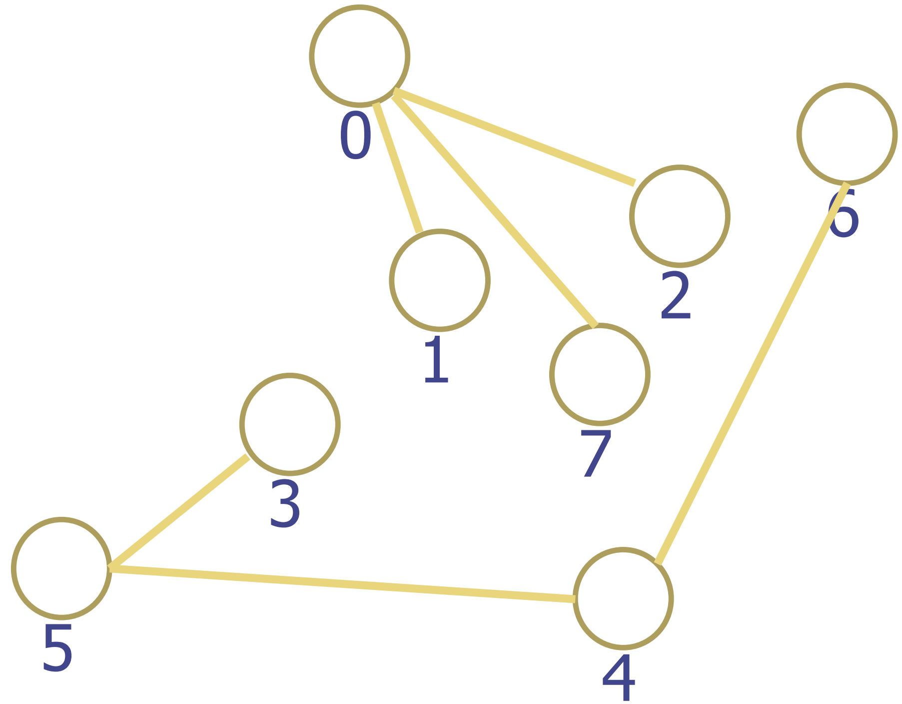
Yes, this is a tree. Any two vertices are connected by exactly one path.

Example

- Are these graphs trees?



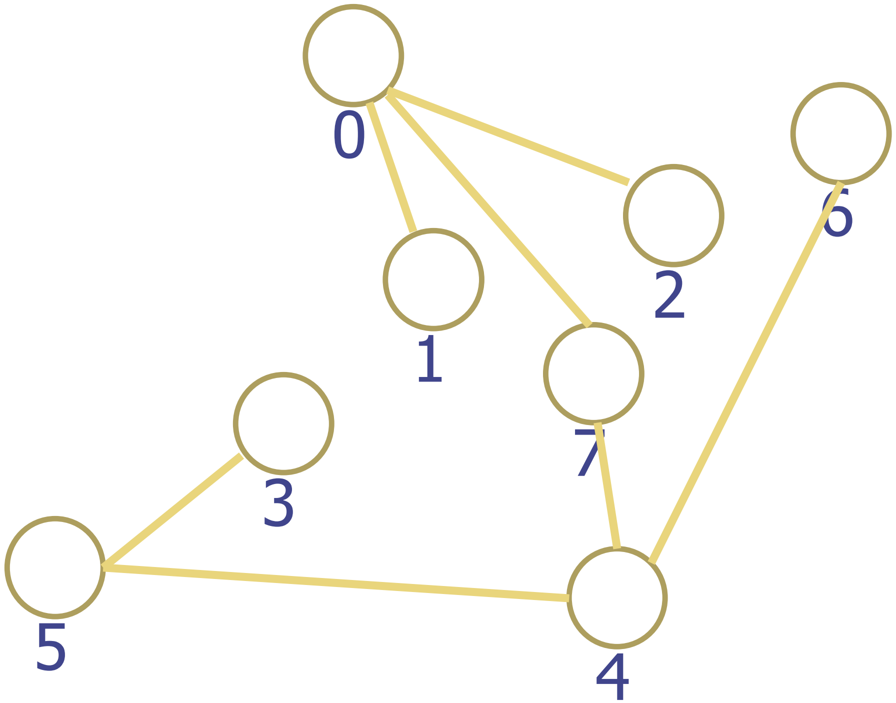
Yes, this is a tree. Any two vertices are connected by exactly one path.



No, this is not a tree. For example, there is no path connecting node 7 to node 4.

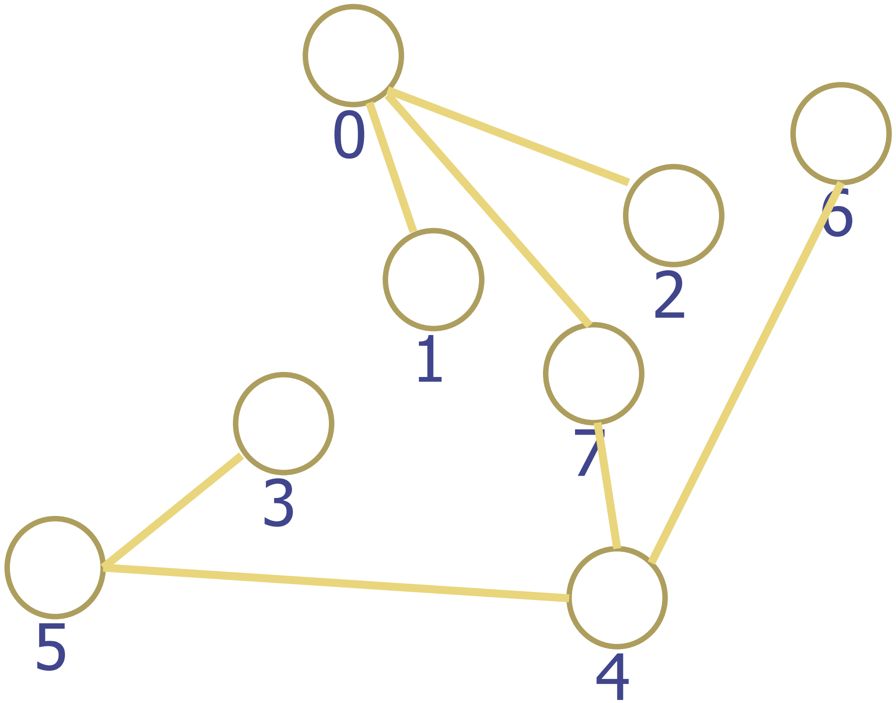
Root of the Tree

- A rooted tree is a tree where one node is designated as the root.



Root of the Tree

- A rooted tree is a tree where one node is designated as the root.
- Given a tree, ANY node can be the root.



Forests:

- Remember: every graph can be broken down into its connected components.

Forests:

- Remember: every graph can be broken down into its connected components.

- If a graph is acyclic, then all of its connected components will be trees.

Forests:

- Remember: every graph can be broken down into its connected components.

- If a graph is acyclic, then all of its connected components will be trees.

- For this reason, acyclic graphs are sometimes described as <u>forests</u>.

DAGs:

- A commonly used data structure:
- DAG = directed, acyclic graph.

DAGs:

- A commonly used data structure:
- DAG = directed, acyclic graph.
- Like a tree, except that:
  - There can be multiple connected components, and multiple "entry points" to each;
  - subtrees can be shared.

Representing graphs:

- Two standard ways to represent graphs:

Representing graphs:

- Two standard ways to represent graphs:

- **Adjacency matrix**: a two dimensional array g[i][j]. An entry of 1 means that there is an edge between vertices i and j.

Representing graphs:

- Two standard ways to represent graphs:

- **Adjacency matrix**: a two dimensional array g[i][j]. An entry of 1 means that there is an edge between vertices i and j.

- **Adjacency list**: for each vertex v, we store a linked list of the vertices u that it connects to by a single edge.

Representing graphs:

- Two standard ways to represent graphs:

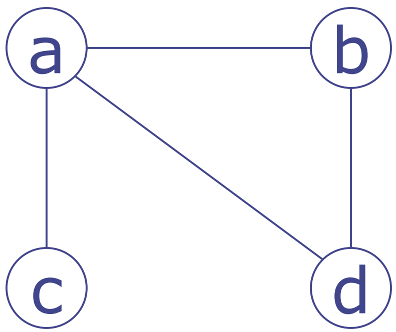
- **Adjacency matrix**: a two dimensional array g[i][j]. An entry of 1 means that there is an edge between vertices i and j.

- **Adjacency list**: for each vertex v, we store a linked list of the vertices u that it connects to by a single edge.

- <u>Choosing between these can have an impact on the complexity of some graph algorithms.</u>

Adjacency matrix representation:

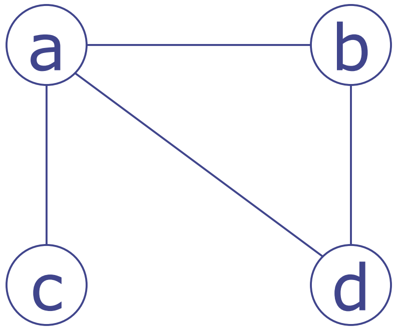
- A simple example:



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

Adjacency matrix representation:

- A simple example:

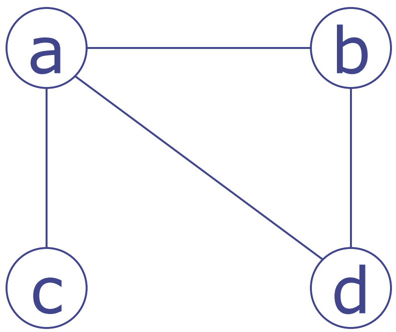


$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

- Uses $\Theta(|V|^2)$ space, much of which will be wasted if the graph is <u>sparse</u> (i.e., relatively few edges).

Adjacency matrix representation:

- A simple example:



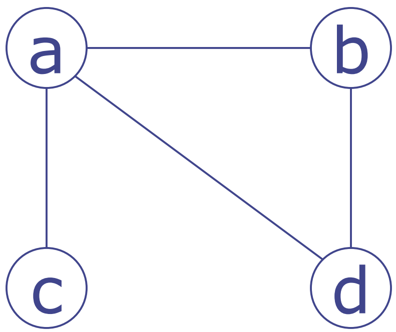
$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

- Uses $\Theta(|V|^2)$ space, much of which will be wasted if the graph is sparse (i.e., relatively few edges).

- Easily adapted to store information about each edge in the entries of the matrix.

Adjacency matrix representation:

- A simple example:



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

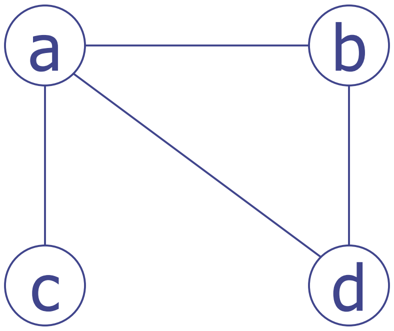
- Uses $\Theta(|V|^2)$ space, much of which will be wasted if the graph is <u>sparse</u> (i.e., relatively few edges).

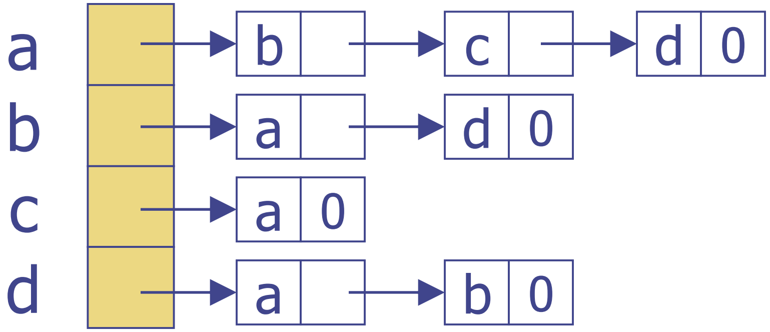
- Easily adapted to store information about each edge in the entries of the matrix.

- Alternatively, if all we need is 0/1, then a single bit will do! (But we still need $|V|^2$ of them …)

Adjacency list representation

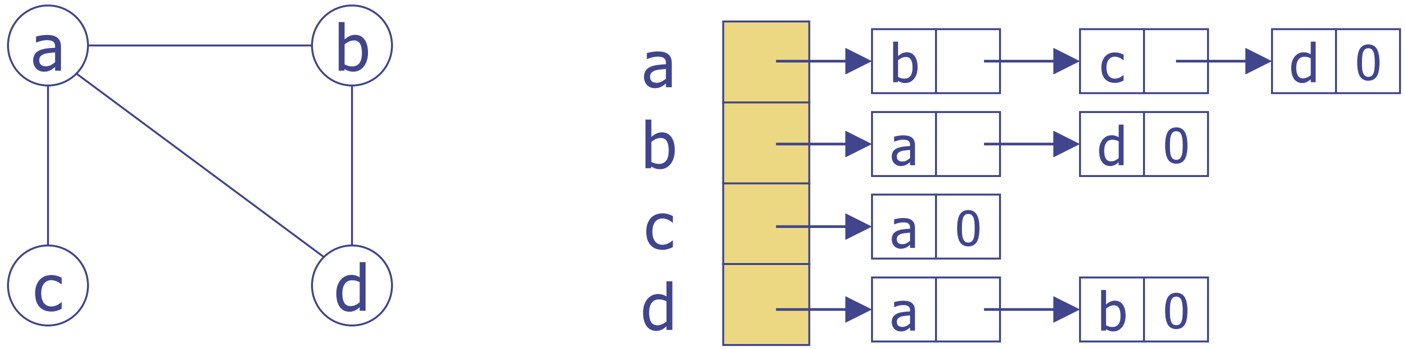
- A simple example:





Adjacency list representation

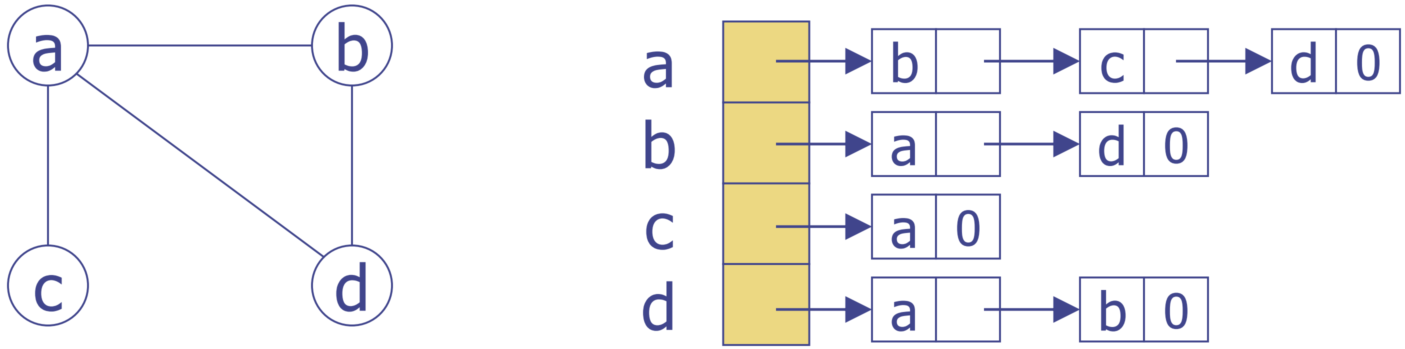
- A simple example:



- Uses $\Theta(|V|+|E|)$ space, good for sparse graphs, more expensive for <u>dense</u> case (i.e., many edges).

Adjacency list representation

- A simple example:

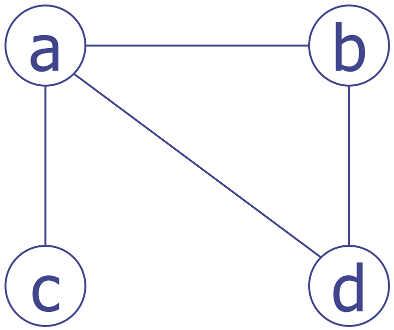


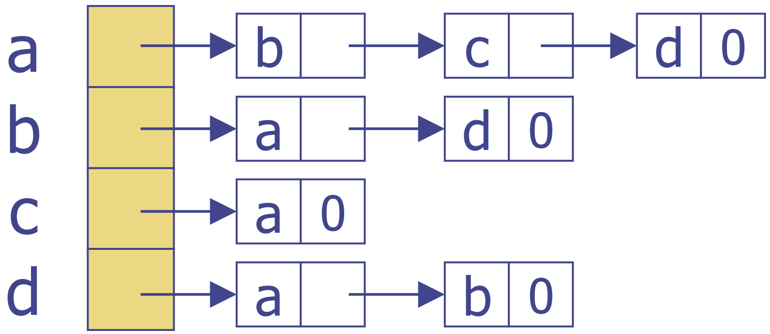
- Uses $\Theta(|V|+|E|)$ space, good for sparse graphs, more expensive for <u>dense</u> case (i.e., many edges).

- Easily adapted to store information about each edge in each part of the linked lists.

Adjacency list representation

- A simple example:





- Uses Θ(|V|+|E|) space, good for sparse graphs, more expensive for <u>dense</u> case (i.e., many edges).

- Easily adapted to store information about each edge in each part of the linked lists.

- Testing to see if there is an edge (u,v) is not Θ(1); we must search the adjacency list of u for v.