Today we will discuss two sections:

Section 10.3 *Representing Graphs and Graph Isomorphism*

Section 10.4 *Connectivity*
(up to paths and isomorphism, not including)

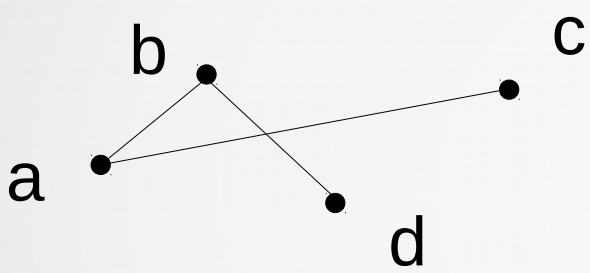
10.3 *Representing Graphs and Graph Isomorphism*

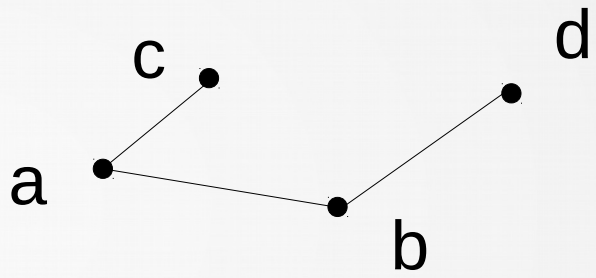
When we are working on an algorithm for graphs, it is inconvenient to get the graphs as pictures.

Therefore, there are several ways to *represent graphs* so that they could be easily “processed”.

10.3 *Representing Graphs and Graph Isomorphism*

Another issue: the same graph can be “drawn” differently.





$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2)$

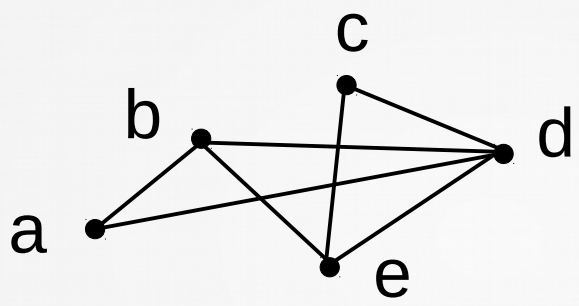
graphs G_1 and G_2 have the same vertices and the same edges: $V_1 = V_2$, $E_1 = E_2$

In such case we say that *the graphs are isomorphic*. *Graph representation(s)* can help us with it.

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



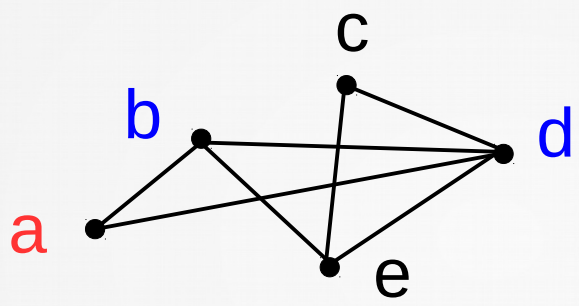
G = (V,E)

An adjacency list for a simple graph G	
Vertex	**Adjacent vertices**

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



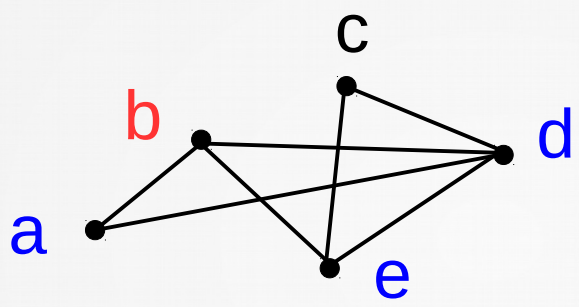
G = (V,E)

An adjacency list for a simple graph G	
Vertex	**Adjacent vertices**
a	b, d

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



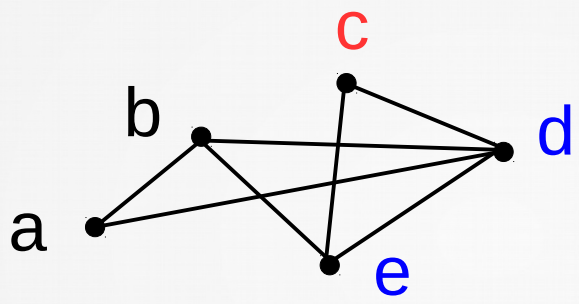
G = (V,E)

An adjacency list for a simple graph G	
Vertex	**Adjacent vertices**
a	b, d
b	a, e, d

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



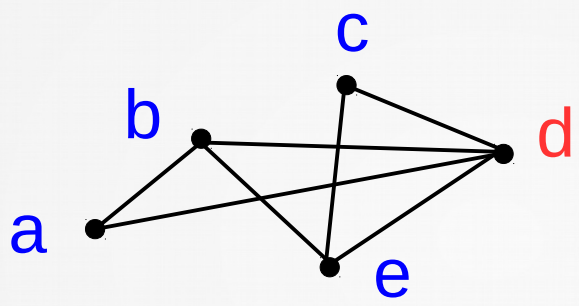
G = (V,E)

An adjacency list for a simple graph G	
Vertex	**Adjacent vertices**
a	b, d
b	a, e, d
c	d, e

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



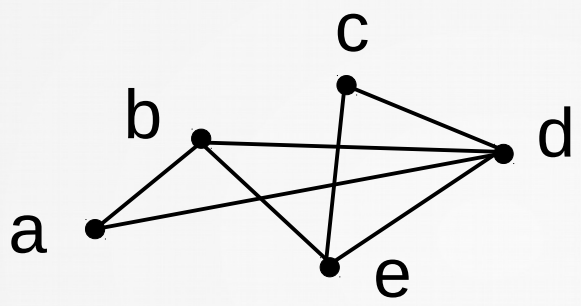
G = (V,E)

An adjacency list for a simple graph G	
Vertex	**Adjacent vertices**
a	b, d
b	a, e, d
c	d, e
d	a, b, c, e

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



G = (V,E)

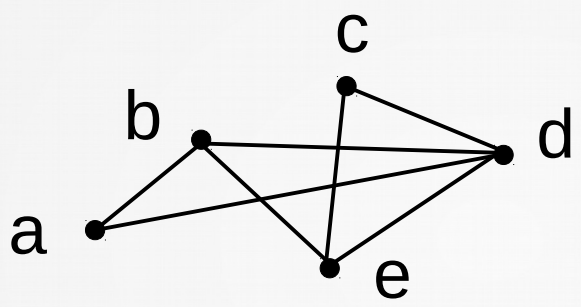
An adjacency list for a simple graph G

Vertex	Adjacent vertices
a	b, d
b	a, e, d
c	d, e
d	a, b, c, e
e	b, c, d

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

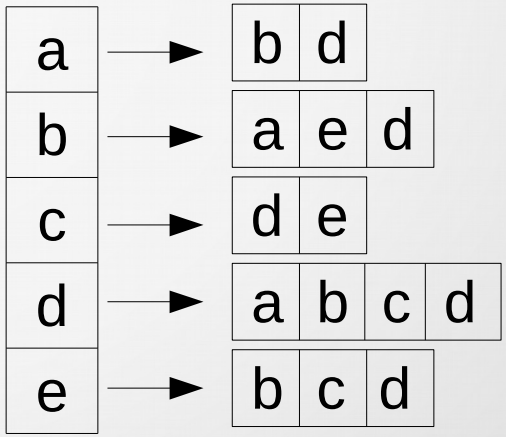
- specify vertices that are adjacent to each vertex in the graph



$G = (V,E)$

An adjacency list for a simple graph G

Vertex	Adjacent vertices
a	b, d
b	a, e, d
c	d, e
d	a, b, c, e
e	b, c, d

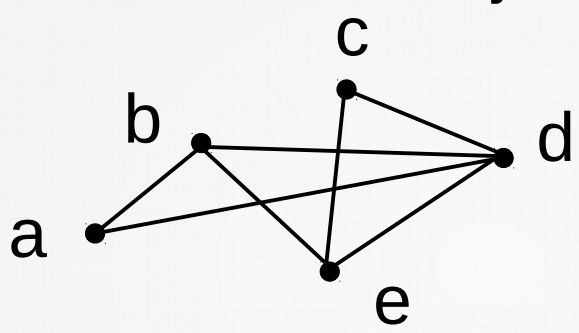


An adjacency list for a simple graph G

10.3 *Representing Graphs and Graph Isomorphism*

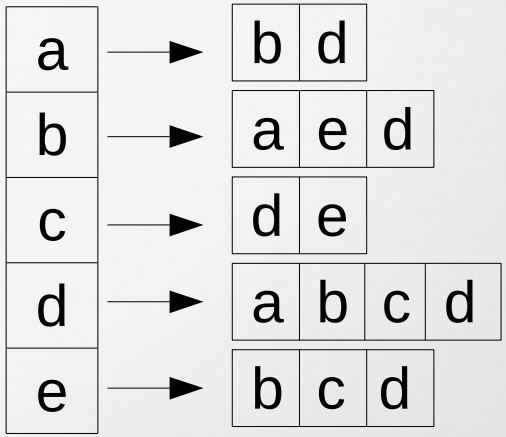
Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



$G = (V,E)$

Let's talk about time it takes to *create the adjacency list* and to *locate an element* in it.

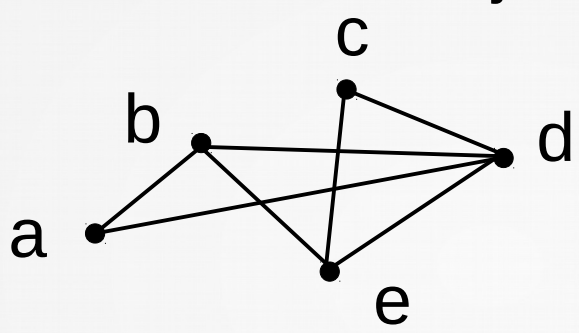


An adjacency list for a simple graph G

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



G = (V,E)

- the time required to list the neighbors of a vertex v is proportional to deg(v), the number of vertices to be listed.

  Therefore, the total time will be proportional to $\sum_{v \in V} deg(v)$

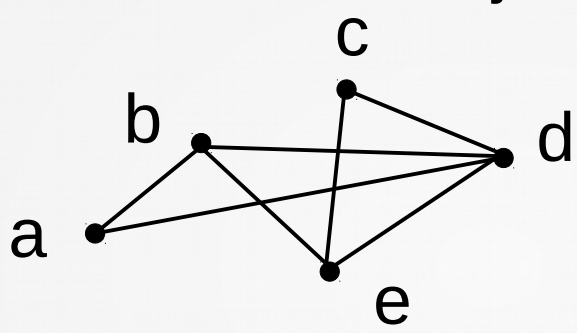
a	b d
b	a e d
c	d e
d	a b c d
e	b c d

An adjacency list for a simple graph G

10.3 *Representing Graphs and Graph Isomorphism*

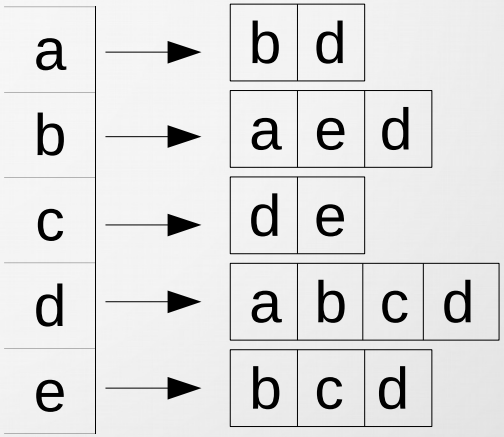
Adjacency list representation:

- specify vertices that are adjacent to each vertex in the graph



G = (V,E)

- the time required to list the neighbors of a vertex v is proportional to deg(v), the number of vertices to be listed. $\sum_{v \in V} deg(v)$
- to determine if {a, b} is an edge, it is enough to scan the list of a's neighbors or the list of b's neighbors. In the worst case, the time required is proportional to the min(deg(a),deg(b)).

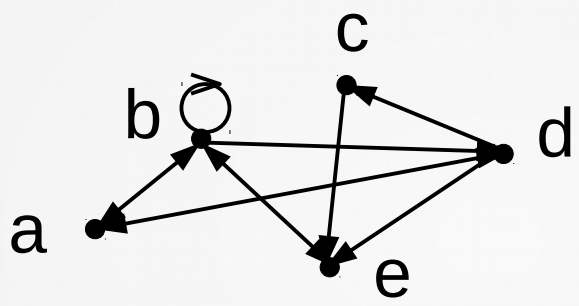


An adjacency list for a simple graph G

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

For directed graphs it is similar



G = (V,E)

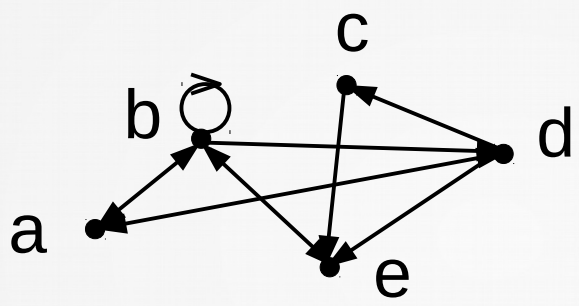
An adjacency list for a directed graph G

Vertex	**Adjacent vertices**

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

For directed graphs it is similar



G = (V,E)

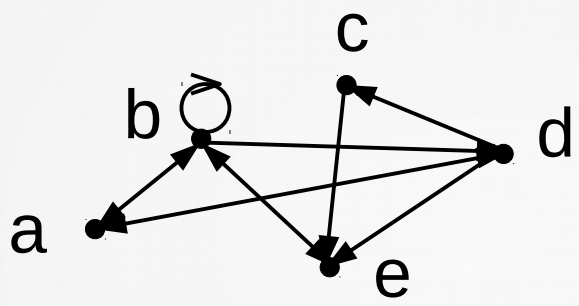
An adjacency list for a directed graph G

Vertex	Adjacent vertices
a	b, d
b	a, b, d, e
c	e
d	a, c, e
e	b

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency list representation:

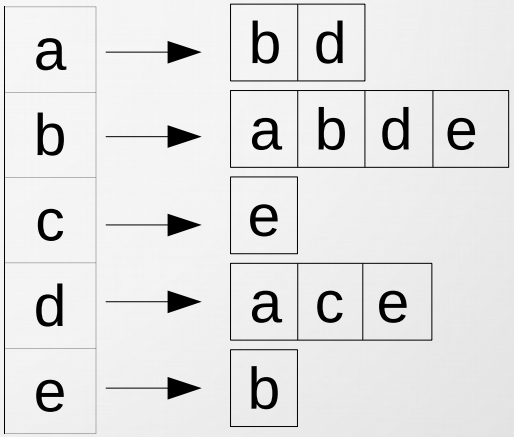
For directed graphs it is similar



G = (V,E)

An adjacency list for a directed graph G

Vertex	Adjacent vertices
a	b, d
b	a, b, d, e
c	e
d	a, c, e
e	b



An adjacency list for a simple graph G

10.3 *Representing Graphs and Graph Isomorphism*

Adjacency matrix representation:

- using zero-one matrices

- $a_{ij} = \begin{cases} 1, \text{ if } \{v_i, v_j\} \in E \\ 0, \text{ otherwise} \end{cases}$

- $a_{ij} = \begin{cases} 1, \text{ if } (v_i, v_j) \in E \\ 0, \text{ otherwise} \end{cases}$

- vertices should be ordered

10.3 *Representing Graphs and Graph Isomorphism*

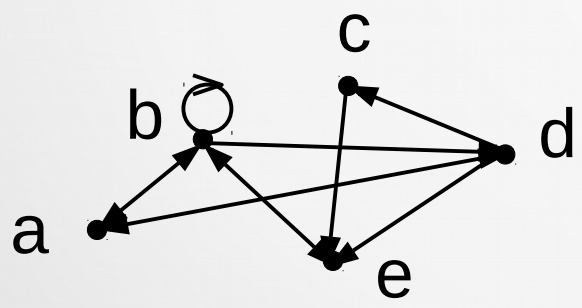
Adjacency matrix representation:

- using zero-one matrices

- $a_{ij} = \begin{cases} 1, \text{ if } \{v_i, v_j\} \in E \\ 0, \text{ otherwise} \end{cases}$

- $a_{ij} = \begin{cases} 1, \text{ if } (v_i, v_j) \in E \\ 0, \text{ otherwise} \end{cases}$

- vertices should be ordered



G = (V,E)

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

adjacency matrix representation of the graph G

10.3 *Representing Graphs and Graph Isomorphism*

Practice

page 676 / 25

Is every zero-one square matrix that is symmetric and has zeros on the diagonal he adjacency matrix of a simple graph?

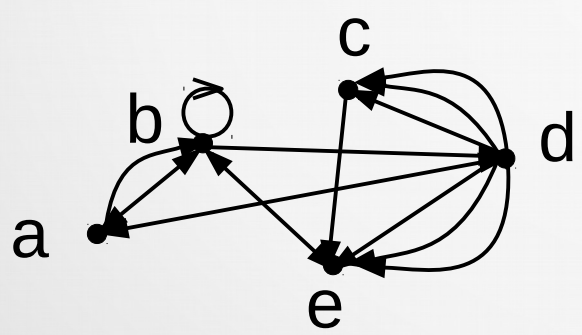
10.3 *Representing Graphs and Graph Isomorphism*

Graphs with multiple edges:

- using zero-one matrices
- $a_{ij} = \begin{cases} \#\text{of edges } \{v_i,v_j\}, \text{ if } \{v_i,v_j\} \in E \\ 0, \text{ otherwise} \end{cases}$

similarly for directed graphs

- vertices should be ordered



G = (V,E)

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 3 & 0 & 3 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

adjacency matrix representation of the graph G

10.3 *Representing Graphs and Graph Isomorphism*

When to use adjacency lists/matrices?

10.3 *Representing Graphs and Graph Isomorphism*

When to use adjacency lists/matrices?

For *sparse graphs*, that contain relatively few edges, *adjacency lists* are preferred.

For *dense graphs*, that contain more than half of all possible edges, *adjacency matrices* are preferred.

10.3 *Representing Graphs and Graph Isomorphism*

When to use adjacency lists/matrices?

For *sparse graphs*, that contain relatively few edges, *adjacency lists* are preferred.

For *dense graphs*, that contain more than half of all possible edges, *adjacency matrices* are preferred.

Some things to consider:

- matrices contain $|V|^2$ entries (*for any type of graph*)
- adjacency lists use less space (*for sparse graphs*)

10.3 *Representing Graphs and Graph Isomorphism*

When to use adjacency lists/matrices?

For *sparse graphs*, that contain relatively few edges, *adjacency lists* are preferred.

For *dense graphs*, that contain more than half of all possible edges, *adjacency matrices* are preferred.

Some things to consider:

- matrices contain $|V|^2$ entries (*for any type of graph*)
- adjacency lists use less space (*for sparse graphs*)
- To determine whether a vertex v_i is adjacent to v_j:
  - Matrices: just examine a_{ij} entry
  - Lists: can require up to $C|V|$, $C \in Z^+$ comparisons

10.3 *Representing Graphs and Graph Isomorphism*

Incidence matrices

is another common way to represent undirected graphs.

Incidence matrices can be used to represent undirected graphs with multiple edges.

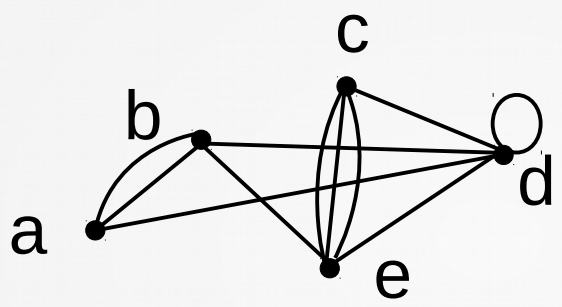
For vertices $v_1,\ldots v_n$ and edges $e_1,\ldots,e_m$, the incidence matrix with respect of ordering V and E is $n \times m$ matrix $M = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise} \end{cases}$$

10.3 *Representing Graphs and Graph Isomorphism*

Incidence matrices

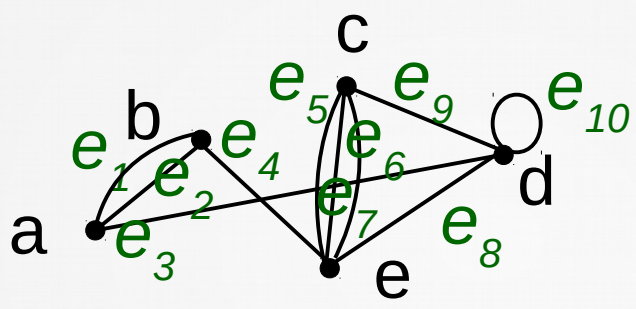
Example:represent the given graph with incident matrix



10.3 *Representing Graphs and Graph Isomorphism*

Incidence matrices

Example:represent the given graph with incident matrix



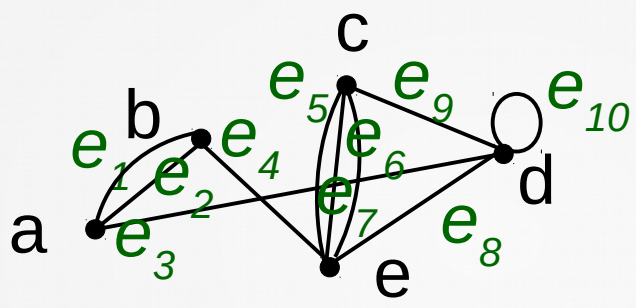
vertices order:
a,b,c,d,e

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
a										
b										
c										
d										
e										

10.3 *Representing Graphs and Graph Isomorphism*

Incidence matrices

Example:represent the given graph with incident matrix



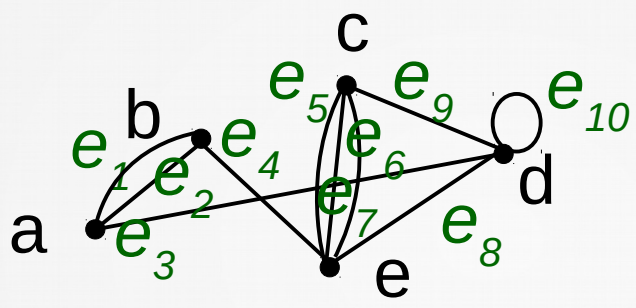
vertices order:
a,b,c,d,e

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
a	1	1	1	0	0	0	0	0	0	0
b	1	1	0	1	0	0	0	0	0	0
c	0	0	0	0	1	1	1	0	1	0
d	0	0	1	0	0	0	0	1	1	1
e	0	0	0	1	1	1	1	1	0	0

10.3 *Representing Graphs and Graph Isomorphism*

Incidence matrices

Example:represent the given graph with incident matrix



vertices order:
a,b,c,d,e

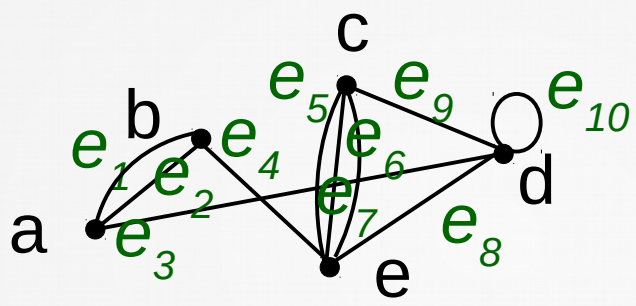
	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
a	1	1	1	0	0	0	0	0	0	0
b	1	1	0	1	0	0	0	0	0	0
c	0	0	0	0	1	1	1	0	1	0
d	0	0	1	0	0	0	0	1	1	1
e	0	0	0	1	1	1	1	1	0	0

loops are columns with exactly one entry equal to 1

10.3 *Representing Graphs and Graph Isomorphism*

Incidence matrices

Example:represent the given graph with incident matrix



vertices order:
a,b,c,d,e

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
a	1	1	1	0	0	0	0	0	0	0
b	1	1	0	1	0	0	0	0	0	0
c	0	0	0	0	1	1	1	0	1	0
d	0	0	1	0	0	0	0	1	1	1
e	0	0	0	1	1	1	1	1	0	0

multiple edges are columns with identical entries

10.3 *Representing Graphs and Graph Isomorphism*

Practice

Page 676 / 30

What is the sum of the entries in a row of the incidence matrix for an undirected graph?

10.3 *Representing Graphs and Graph Isomorphism*

Isomorphism of Graphs

Let $G = (V, E)$ and $G'=(V',E')$ be simple graphs.
G and G' are *isomorphic* if there is a *bijection* $f: V \rightarrow V'$ such that for every pair of vertices $x, y \in V$, $\{x, y\} \in E$ if and only if $\{f(x), f(y)\} \in E'$.

The function f is called an *isomorphism* from G to G'.

Two simple graphs that are not isomorphic are called *nonisomorphic*.

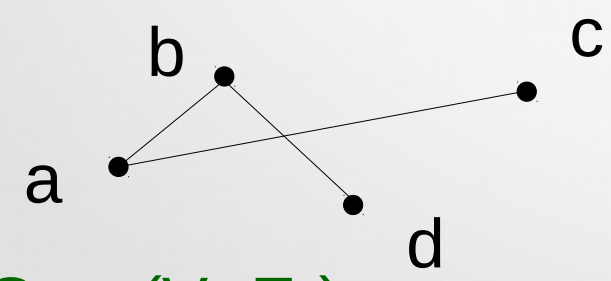
10.3 *Representing Graphs and Graph Isomorphism*

Isomorphism of Graphs

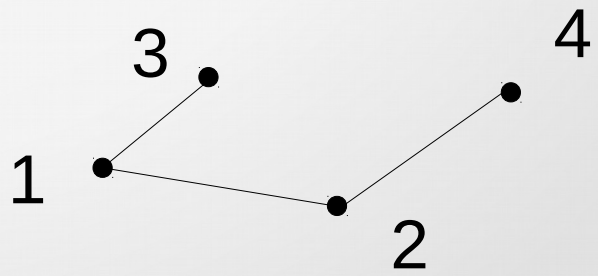
Let $G = (V, E)$ and $G'=(V',E')$ be simple graphs.
G and G' are *isomorphic* if there is a *bijection* $f: V \rightarrow V'$ such that for every pair of vertices $x, y \in V$, $\{x, y\} \in E$ if and only if $\{f(x), f(y)\} \in E'$.

The function f is called an *isomorphism* from G to G'.

Two simple graphs that are not isomorphic are called *nonisomorphic*.



$G_1 = (V_1,E_1)$



$G_2 = (V_2,E_2)$

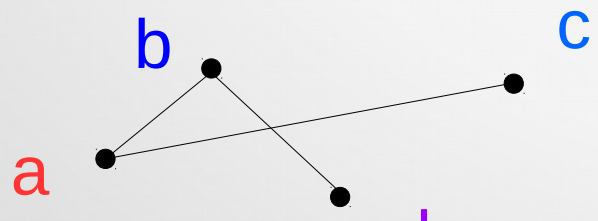
10.3 *Representing Graphs and Graph Isomorphism*

Isomorphism of Graphs

Let G = (V, E) and G'=(V',E') be simple graphs.
G and G' are *isomorphic* if there is a *bijection f*: V → V' such that for every pair of vertices x, y ∈ V, {x, y} ∈ E if and only if {f(x), f(y)} ∈ E'.

The function *f* is called an *isomorphism* from G to G'.

Two simple graphs that are not isomorphic are called *nonisomorphic*.



$G_1 = (V_1, E_1)$

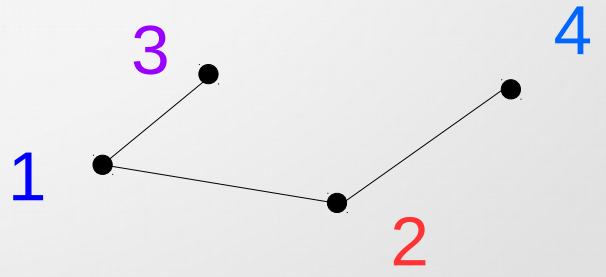
$f: V_1 \rightarrow V_2$

$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$



$G_2 = (V_2, E_2)$

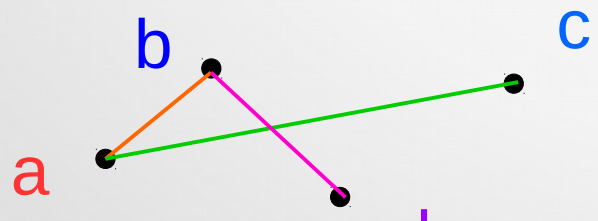
10.3 *Representing Graphs and Graph Isomorphism*

Isomorphism of Graphs

Let G = (V, E) and G'=(V',E') be simple graphs.
G and G' are *isomorphic* if there is a *bijection f*: V → V' such that for every pair of vertices x, y ∈ V, {x, y} ∈ E if and only if {f(x), f(y)} ∈ E'.

The function *f* is called an *isomorphism* from G to G'.

Two simple graphs that are not isomorphic are called *nonisomorphic*.



$G_1 = (V_1, E_1)$

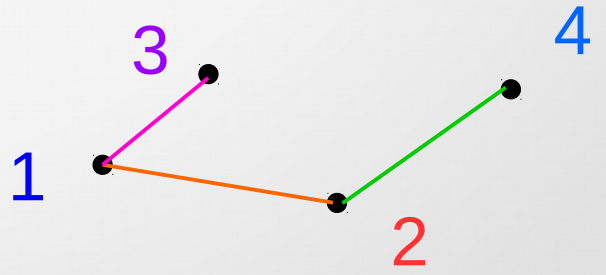
$f: V_1 \rightarrow V_2$

$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$



$G_2 = (V_2, E_2)$

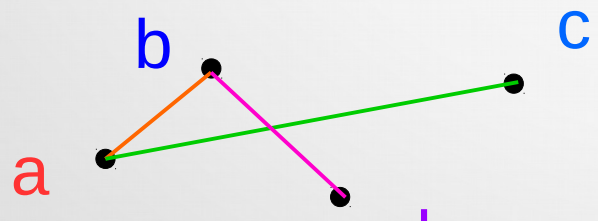
10.3 *Representing Graphs and Graph Isomorphism*

Isomorphism of Graphs

Let G = (V, E) and G'=(V',E') be simple graphs.
G and G' are *isomorphic* if there is a *bijection f*: V → V' such that for every pair of vertices x, y ∈ V, {x, y} ∈ E if and only if {f(x), f(y)} ∈ E'.

The function *f* is called an *isomorphism* from G to G'.

Two simple graphs that are not isomorphic are called *nonisomorphic*.



$G_1 = (V_1, E_1)$

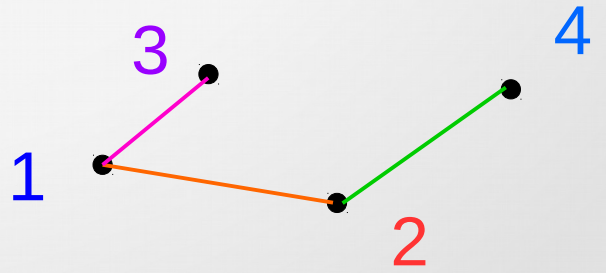
$f: V_1 \rightarrow V_2$

$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$



$G_2 = (V_2, E_2)$

G_1 and G_2 are isomorphic

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

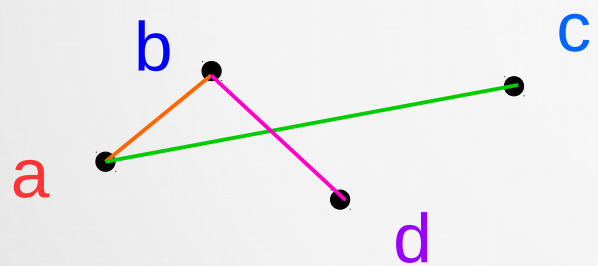
To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.
One way to do it: adjacency matrices

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.

One way to do it: adjacency matrices



$G_1 = (V_1, E_1)$

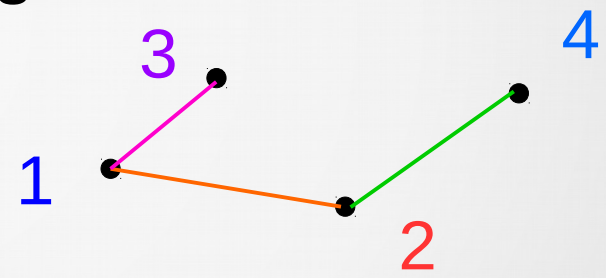
$f: V_1 \rightarrow V_2$

$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$



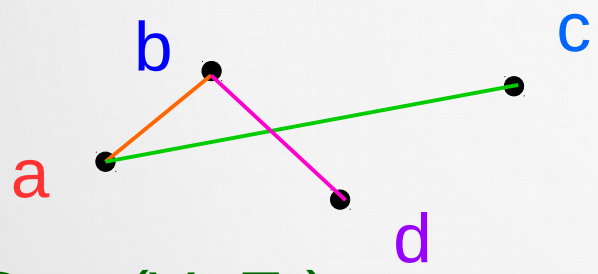
$G_2 = (V_2, E_2)$

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.

One way to do it: adjacency matrices



$G_1 = (V_1, E_1)$

a	0	1	1	0
b	1	0	0	1
c	1	0	0	0
d	0	1	0	0

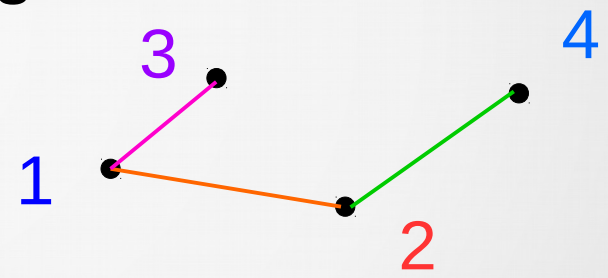
$f: V_1 \rightarrow V_2$

$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$



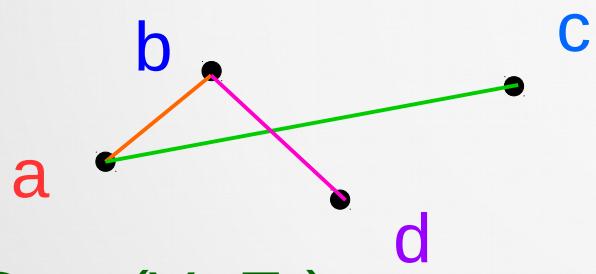
$G_2 = (V_2, E_2)$

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.

One way to do it: adjacency matrices



$G_1 = (V_1, E_1)$

a	0	1	1	0
b	1	0	0	1
c	1	0	0	0
d	0	1	0	0

$f: V_1 \rightarrow V_2$

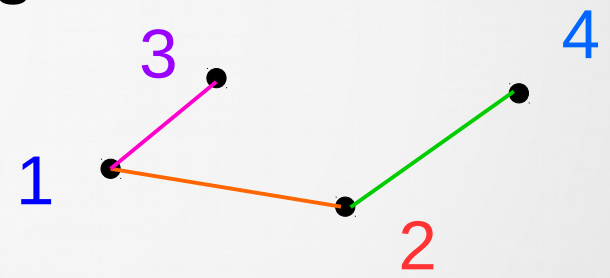
$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$

Matrices are equal



$G_2 = (V_2, E_2)$

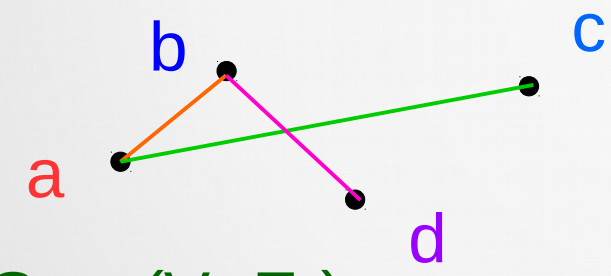
2	0	1	1	0
1	1	0	0	1
4	1	0	0	0
3	0	1	0	0

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.

One way to do it: adjacency matrices



$G_1 = (V_1, E_1)$

a	0	1	1	0
b	1	0	0	1
c	1	0	0	0
d	0	1	0	0

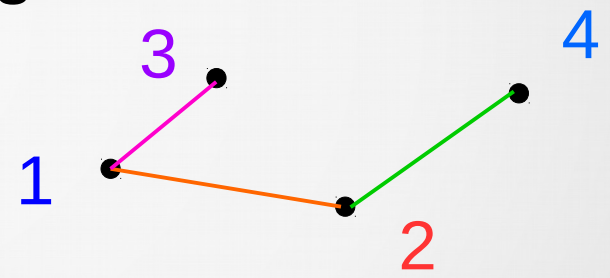
$f: V_1 \rightarrow V_2$

$a \rightarrow 2$

$b \rightarrow 1$

$c \rightarrow 4$

$d \rightarrow 3$



$G_2 = (V_2, E_2)$

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

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.

One way to do it: adjacency matrices

Drawback: if the given function is not an isomorphism, it doesn't guarantee that two graphs are not isomorphic. There might be another correspondence of the vertices in graphs that is an isomorphism.

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

To show that a function $f : V_1 \rightarrow V_2$ is an isomorphism we need to show that *f* preserves the *presence* and the *absence* of edges.

One way to do it: adjacency matrices

Drawback: if the given function is not an isomorphism, it doesn't guarantee that two graphs are not isomorphic. There might be another correspondence of the vertices in graphs that is an isomorphism.

The best algorithms known for determining whether two graphs are isomorphic have exponential worst-case time complexity (in $|V|$). We want a polynomial one.

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

If we are not given a correspondence (function *f*) to check, then there are |V|! possible *bijections* (*one-to-one correspondences*) to check, which is impractical for large |V|.

Let |V| = n

$v_1 \rightarrow$ ___
n options

$v_2 \rightarrow$ ___
(n-1) options

…

$v_n \rightarrow$ ___
1 option

Therefore, we get
n·(n-1)·(n-2)·…·1 = n!

10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

Sometimes, it is easy to see that two graphs are not isomorphic: check the number of vertices and the number of edges, they should be equal accordingly.

$|V_1| = |V_2|$
$|E_1| = |E_2|$

Also, the degrees of vertices in isomorphic graphs must be the same.

graph invariant – is a property preserved by isomorphism. Properties above are graph invariants.

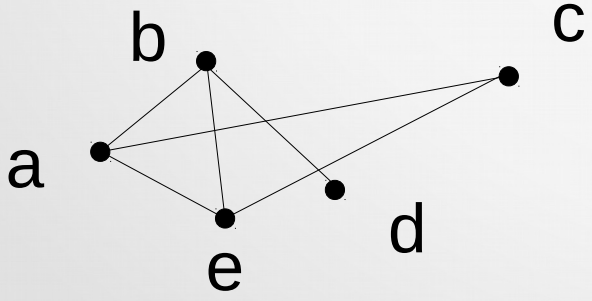
10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

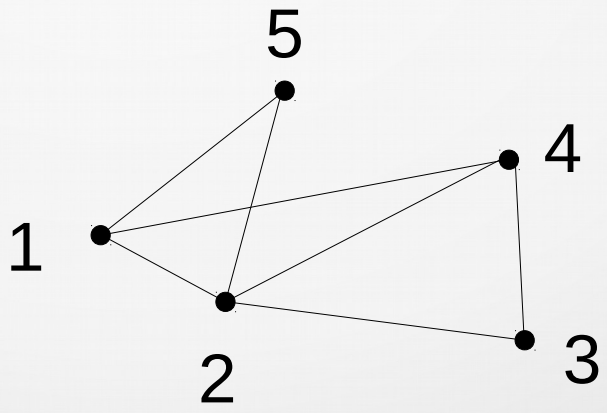
Sometimes, it is easy to see that two graphs are not isomorphic: check the number of vertices and the number of edges, they should be equal accordingly.

$|V_1| = |V_2|$
$|E_1| = |E_2|$

the degrees of vertices in isomorphic graphs must be the same.



$G_1 = (V_1, E_1)$



$G_2 = (V_2, E_2)$

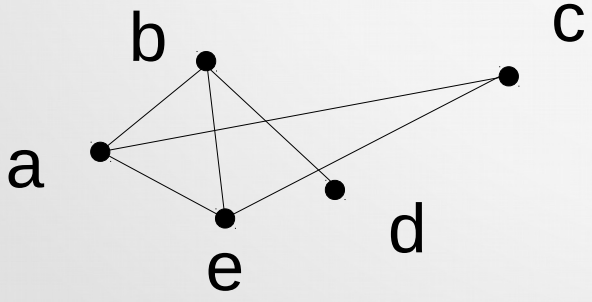
10.3 *Representing Graphs and Graph Isomorphism*

How to check that two graphs are isomorphic?

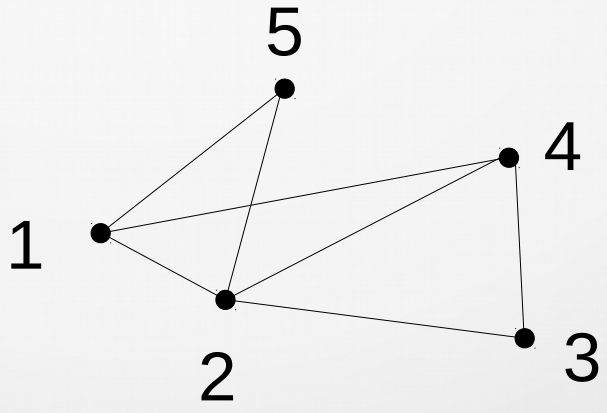
Sometimes, it is easy to see that two graphs are not isomorphic: check the number of vertices and the number of edges, they should be equal accordingly.

$|V_1| = |V_2|$
$|E_1| = |E_2|$

the degrees of vertices in isomorphic graphs must be the same.



$G_1 = (V_1, E_1)$



$G_2 = (V_2, E_2)$

graphs are not isomorphic, because $|E_1| = 6$, $|E_2| = 7$

10.3 *Representing Graphs and Graph Isomorphism*

Practice

page 677 / 57(c)

Are the simple graphs with the following adjacency matrices isomorphic?

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

10.3 *Representing Graphs and Graph Isomorphism*

Practice

page 677 / 45

Show that isomorphism of simple graphs is an equivalence relationship.

(recall that equivalence relation if it is *reflexive*, *symmetric* and *transitive*.)

10.3 *Representing Graphs and Graph Isomorphism*

Applications of graph isomorphisms

Chemistry:
To model chemical compounds chemists use multigraphs, known as *molecular graphs*.
vertices: atoms
edges: chemical bounds between the atoms

Two structural isomers, molecules with identical molecular formulas but with atoms bonded differently, have non-isomorphic molecular graphs.

When a potentially new chemical compound is synthesized, a database of molecular graphs is checked to see whether the molecular graph of the compound is the same as one already known.

10.3 *Representing Graphs and Graph Isomorphism*

Applications of graph isomorphisms

Electrical engineering:
Electronic circuits are modeled using graphs which
vertices: components
edges: connections between components

Modern integrated circuits (*chips*) are miniaturized electronic circuits, often with millions of transistors and connections between them.
Automation tools are used to design chips because of the complexity.

Graph isomorphism is the basis for the verification that a particular layout of a circuit produced by an automated tools corresponds to the original schematics of the design.

10.3 *Representing Graphs and Graph Isomorphism*

Applications of graph isomorphisms

Electrical engineering:
Electronic circuits are modeled using graphs which
vertices: components
edges: connections between components

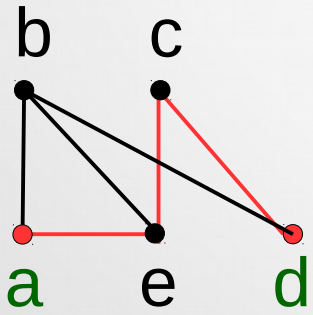
Modern integrated circuits (*chips*) are miniaturized electronic circuits, often with millions of transistors and connections between them.
Automation tools are used to design chips because of the complexity.

Graph isomorphism can also be used to determine whether a chip from one company includes intellectual property from a different vendor by looking for large isomorphic subgraphs in the graphs modeling these chips.

10.4 *Connectivity*

A *path* from *u* to *v* in an undirected graph G is a sequence of edges of G that starts at vertex *u* and ends at vertex *v*:
⟨{*u*,v_1}, {v_1,v_2}, ..., {v_{n-1},*v*}⟩

A *path* can also be denoted by the sequence of vertices u, v1, ..., v when the graph is simple.

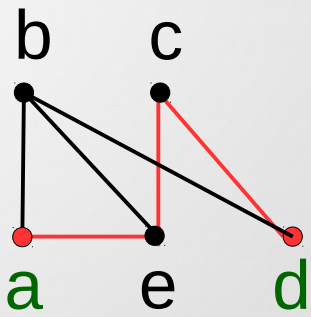


10.4 *Connectivity*

A *path* from *u* to *v* in an undirected graph G is a sequence of edges of G that starts at vertex *u* and ends at vertex *v*:
⟨{*u*,v_1}, {v_1,v_2}, ..., {v_{n-1},*v*}⟩

A *path* can also be denoted by the sequence of vertices u, v1, ..., v when the graph is simple.

The *length of a path* is the number of edges in the walk.



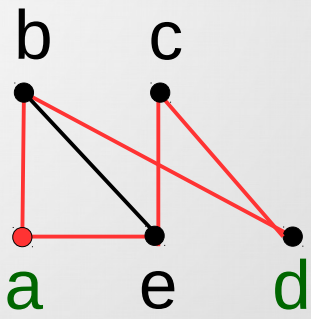
10.4 *Connectivity*

A *path* from *u* to *v* in an undirected graph G is a sequence of edges of G that starts at vertex *u* and ends at vertex *v*:
⟨{*u*,v_1}, {v_1,v_2}, ..., {v_{n-1},*v*}⟩

A *path* can also be denoted by the sequence of vertices u, v1, ..., v when the graph is simple.

The *length of a path* is the number of edges in the walk.

The path is a *circuit* if it begins and ends at the same vertex, i.e. *u* = *v*, and has a length greater than zero .



10.4 *Connectivity*

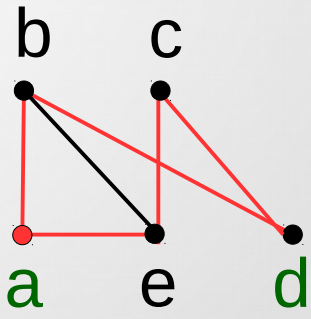
A *path* from *u* to *v* in an undirected graph G is a sequence of edges of G that starts at vertex *u* and ends at vertex *v*:
$\langle\{u,v_1\}, \{v_1,v_2\}, \ldots, \{v_{n-1},v\}\rangle$

A *path* can also be denoted by the sequence of vertices u, v1, ..., v when the graph is simple.

The *length of a path* is the number of edges in the walk.

The path is a *circuit* if it begins and ends at the same vertex, i.e. $u = v$, and has a length greater than zero .

A *path* / *circuit* is said to *pass through the vertices* $u, v_1, v_2, \ldots, v_{n-1}, v$ or *traverse the edges* $e_1, \ldots, e_n$.



10.4 *Connectivity*

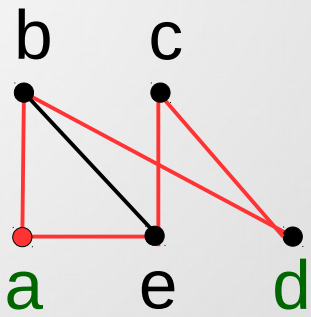
A *path* from *u* to *v* in an undirected graph G is a sequence of edges of G that starts at vertex *u* and ends at vertex *v*:
⟨{*u*,v_1}, {v_1,v_2}, ..., {v_{n-1},*v*}⟩

A *path* can also be denoted by the sequence of vertices u, v1, ..., v when the graph is simple.

The *length of a path* is the number of edges in the walk.

The path is a *circuit* if it begins and ends at the same vertex, i.e. *u* = *v*, and has a length greater than zero .

A *path* / *circuit* is said to *pass through the vertices* *u*, v_1, v_2, ..., v_{n-1}, *v* or *traverse the edges* e_1, … , e_n.



A path or circuit is *simple* if it doesn't contain the same edge more than once.

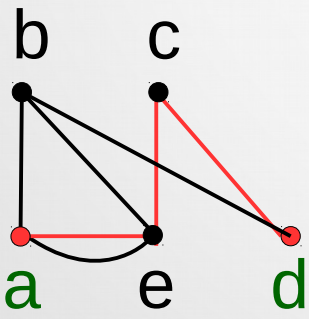
10.4 *Connectivity*

A path of length zero consists of a single vertex.

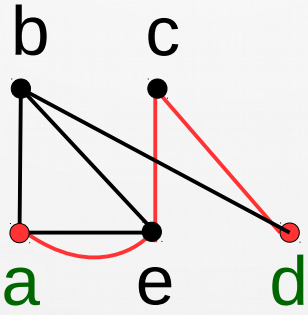
When we do not distinguish between multiple edges:

- we will denote a path $e_1, e_2, \ldots, e_n$, where e_i is associated with $\{ x_{i-1}, x_i \}$ for $i = 1, 2, \ldots, n$ by its vertex sequence $x_1, x_2, \ldots, x_n$

  note that it is not a unique path



path: *a,e,c,d*



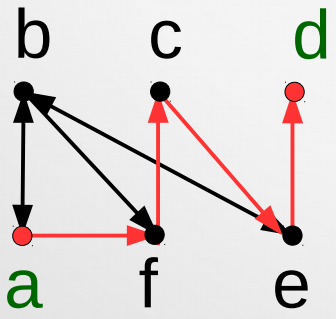
path: *a,e,c,d*

10.4 *Connectivity*

A *path* from *u* to *v* in a directed graph G is a sequence of edges of G that starts at vertex *u* and ends at vertex *v*: (u,v_1), (v_1,v_2), ..., (v_{n-1},v)

A *path* can also be denoted by the sequence of vertices u, v1, ..., v when there is no multiple edges.

The *length of a path* is the number of edges in the walk.



path: (a,f), (f,c), (c,e), (e,d)

or

path: a, f, c, e, d

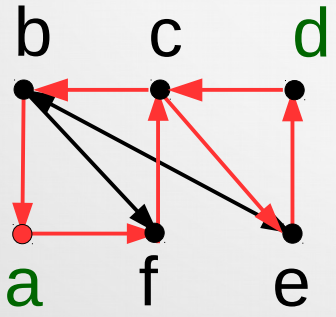
length of the path: 4

10.4 *Connectivity*

The path is a *circuit* / *cycle* if it begins and ends at the same vertex, i.e. $u = v$, and has a length greater than zero .

A *path* / *circuit* is said to *pass through the vertices* $u, v_1, v_2, ..., v_{n-1}, v$ or *traverse the edges* $e_1, \ldots, e_n$.

A path or circuit is *simple* if it doesn't contain the same edge more than once.



path: (a,f), (f,c), (c,e), (e,d), (d,c), (c,b), (b,a)

or

path: a, f, c, e, d, c, b, a

length of the path: 7

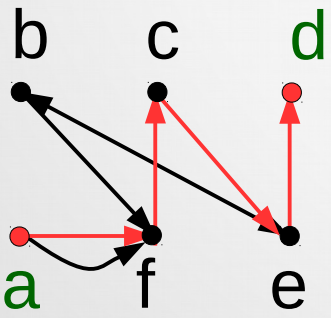
10.4 *Connectivity*

A path of length zero consists of a single vertex.

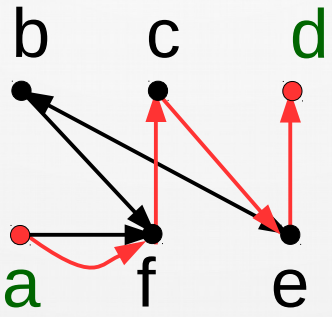
When we do not distinguish between multiple edges:

- we will denote a path $e_1, e_2, \ldots, e_n$, where e_i is associated with $\{ x_{i-1}, x_i \}$ for $i = 1, 2, \ldots, n$ by its vertex sequence $x_1, x_2, \ldots, x_n$

  note that it is not a unique path



path: *a, f, c, e, d*



path: *a, f, c, e, d*

10.4 *Connectivity*

Many problems can be modeled with paths formed by traveling along the edges of the graphs.

Examples:

- message exchange between two computers

- planning efficiency routes for deliveries, garbage pickup, and so forth.

10.4 *Connectivity*

More examples:

Paths in *acquaintanceship graphs*:
There is a path between two people if there is a chain of people linking them.

Many social scientists have conjectured that almost every pair of people in the world are linked by a small chain of people, perhaps containing just 5 or fewer people.

The play ***Six Degrees of Separation***, written by American playwright John Guare that premiered in 1990, explores the existential premise that everyone in the world is connected to everyone else in the world by a chain of no more than six acquaintances, thus, "six degrees of separation".

10.4 *Connectivity*

Connectedness in undirected graphs:

[Def] an *undirected graph is connected* if there is a path between every pair of distinct vertices of the graph.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Def] an *undirected graph is connected* if there is a path between every pair of distinct vertices of the graph.

An *undirected graph* which is not connected *is* called *disconnected*.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Def] an *undirected graph is connected* if there is a path between every pair of distinct vertices of the graph.

An *undirected graph* which is not connected *is* called *disconnected*.

To *disconnect a graph*: remove vertices or edges, or both, to produce a disconnected graph.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Def] an *undirected graph is connected* if there is a path between every pair of distinct vertices of the graph.

An *undirected graph* which is not connected *is* called *disconnected*.

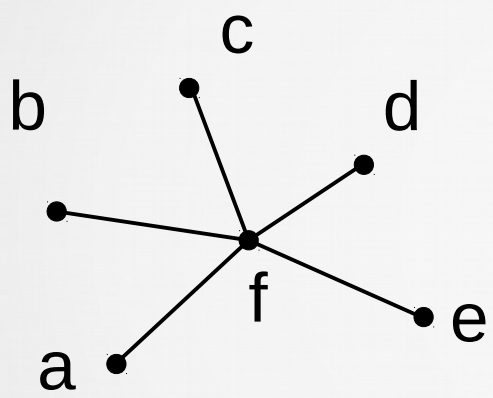
To *disconnect a graph*: remove vertices or edges, or both, to produce a disconnected graph.

Example: any two computers in a network can communicate if and only if the graph of this network is connected.

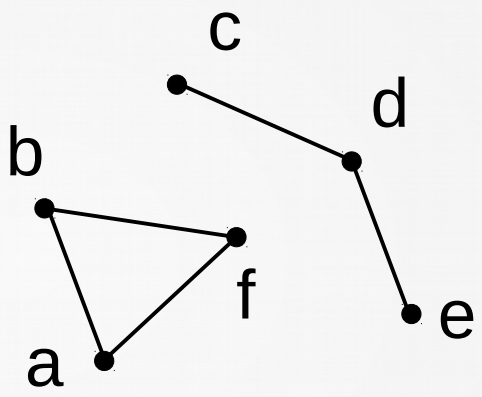
10.4 *Connectivity*

Connectedness in undirected graphs:

Examples:



connected graph



disconnected graph

10.4 *Connectivity*

Connectedness in undirected graphs:

[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

Proof: Let G = (V,E) be undirected connected graph,and let $u,v \in V$.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

Proof: Let G = (V,E) be undirected connected graph,and let $u,v \in V$.
G is connected, therefore there is at least one path between u and v.
Let $x_0 = u, x_1, \ldots, x_n = v$ be a path between vertices u and v of least length.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

Proof: Let G = (V,E) be undirected connected graph,and let $u,v \in V$.
G is connected, therefore there is at least one path between u and v.
Let $x_0 = u, x_1, \ldots, x_n = v$ be a path between vertices u and v of least length. This path must be simple.

10.4 *Connectivity*

Connectedness in undirected graphs:

[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

Proof: Let G = (V,E) be undirected connected graph,and let $u,v \in V$.
G is connected, therefore there is at least one path between u and v.
Let $x_0 = u, x_1, \ldots, x_n = v$ be a path between vertices u and v of least length. This path must be simple.
Assume it is not so.

10.4 *Connectivity*

Connectedness in undirected graphs:

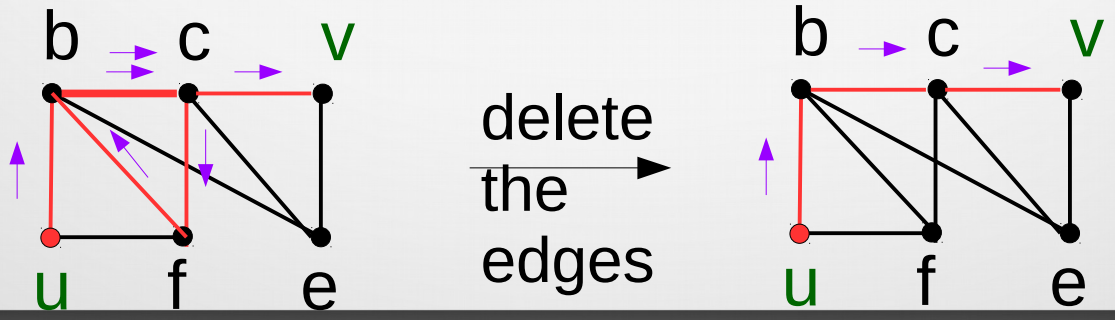
[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

Proof: Let G = (V,E) be undirected connected graph,and let $u,v \in V$.

G is connected, therefore there is at least one path between u and v.

Let $x_0 = u, x_1, \ldots, x_n = v$ be a path between vertices u and v of least length. This path must be simple.

Assume it is not so. Then, for some i,j with $0 \le i < j$, $x_i = x_j$.



10.4 *Connectivity*

Connectedness in undirected graphs:

[Theorem] there is a simple path between any pair of distinct vertices of an undirected connected graph.

Proof: Let G = (V,E) be undirected connected graph,and let $u,v \in V$.
G is connected, therefore there is at least one path between u and v.
Let $x_0 = u, x_1, \ldots, x_n = v$ be a path between vertices u and v of least length. This path must be simple.
Assume it is not so. Then, for some i,j with $0 \le i < j$, $x_i = x_j$.
So let's delete the edges corresponding to the sequence $x_i,\ldots,x_{j-1}$ from the path – we will get the path $x_1,\ldots,x_{i-1},x_j,\ldots,x_n$ of a smaller length – this contradicts our assumption.
Therefore, the path $x_0 = u, x_1, \ldots, x_n = v$ is simple.

q.e.d.

10.4 *Connectivity*

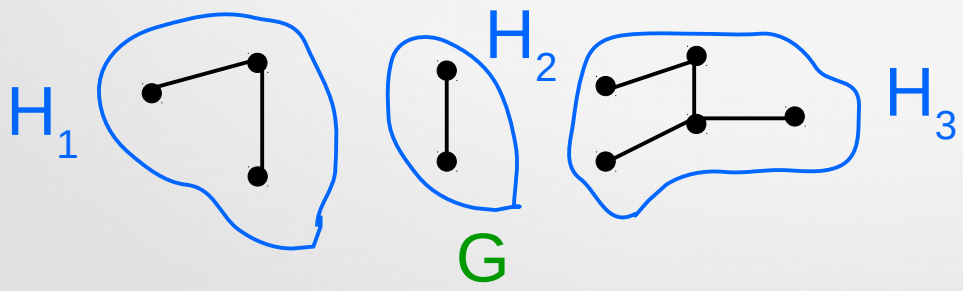
Connected components

[Def] a *subgraph of a graph* G = (V,E) is a graph H=(W,F) where $W \subseteq V$, ad $F \subseteq E$.

Graph H is a *proper subgraph* of G is $H \neq G$.

[Def] A *connected component* of a graph G is a connected subgraph of G that is not a proper subgraph of another connected subgraph of G.

i.e. maximal connected subgraph of G.



graph G and its connected components H_1, H_2, and H_3.

10.4 *Connectivity*

How connected is the graph ?

Imagine a graph representing computer network.

If it is connected, then any two computers can communicate.

However, we need to know how reliable the network is.

If one communication link fails, will all computer still be able to communicate with each other?

10.4 *Connectivity*

How connected is the graph ?

If a vertex and all incident edges are removed from the graph, and the produced subgraph has more connected components, then such a vertex is called *cut vertex* or *articulation point*.

The removal of a cut vertex from a connected graph produces unconnected subgraph.

10.4 *Connectivity*

How connected is the graph ?

If an edge removed from the graph produces a subgraph with more connected components than in the original graph, then such an edge is called *cut edge* or *bridge*.

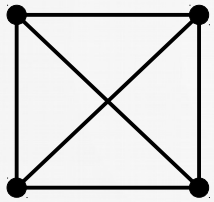
In a computer network graph, a cut vertex and a cut edge represent an essential router and an essential link that cannot fail for all computers to be able to communicate.

Not all graphs have cut vertices or cut edge. Such graphs are called *nonseparable graphs*.

10.4 *Connectivity*

How connected is the graph ?

Example: Find the cut vertices and cut edges in the graph below.

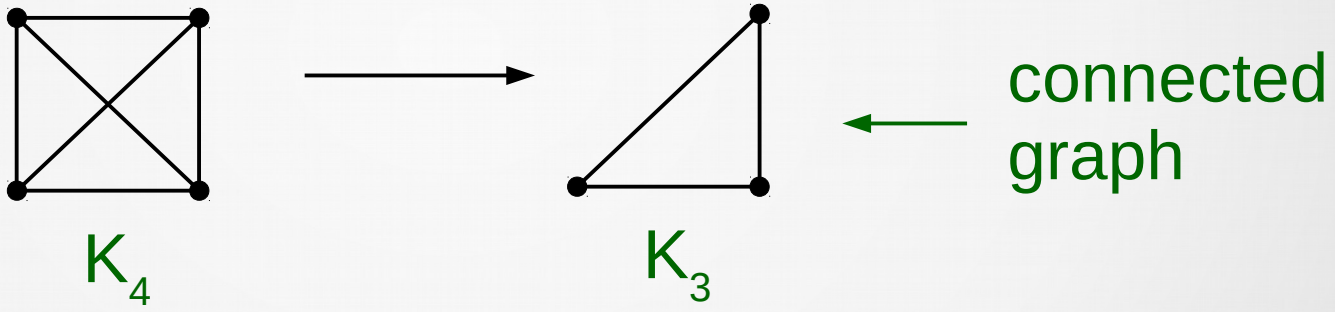


K_4

10.4 *Connectivity*

How connected is the graph ?

Example: Find the cut vertices and cut edges in the graph below.



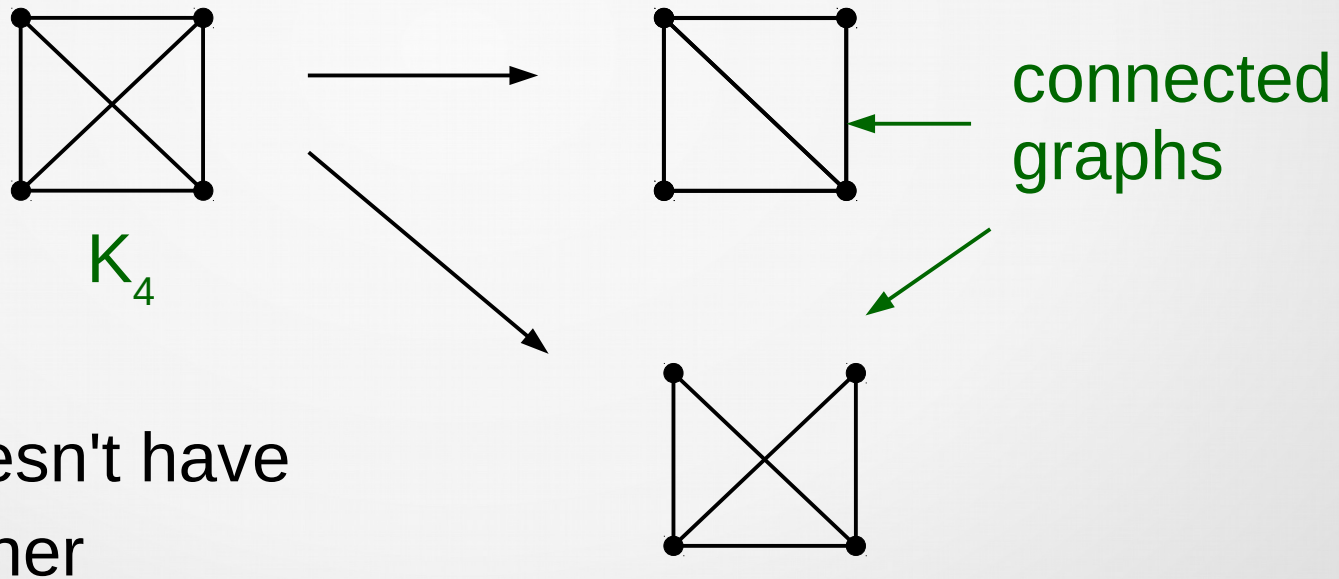
K_4 K_3

Hence K_4 doesn't have cut vertices

10.4 *Connectivity*

How connected is the graph ?

Example: Find the cut vertices and cut edges in the graph below.

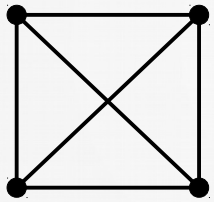


Hence K_4 doesn't have cut edges either

10.4 *Connectivity*

How connected is the graph ?

Example: Find the cut vertices and cut edges in the graph below.



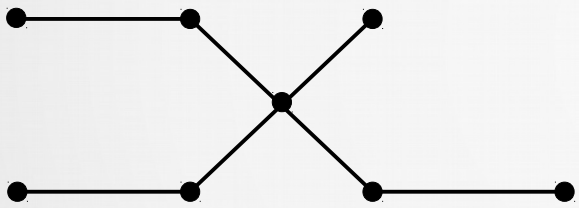
K_4

nonseparable graph

10.4 *Connectivity*

How connected is the graph ?

Example: Find the cut vertices and cut edges in the graph below.



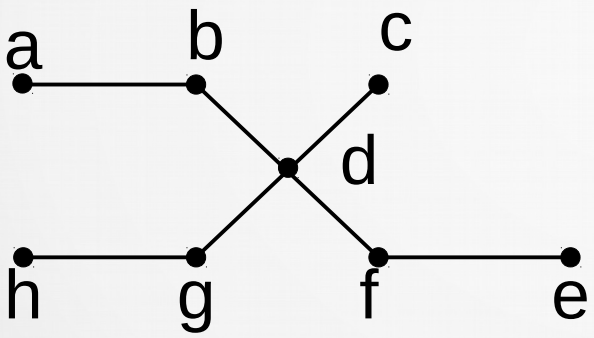
G

10.4 *Connectivity*

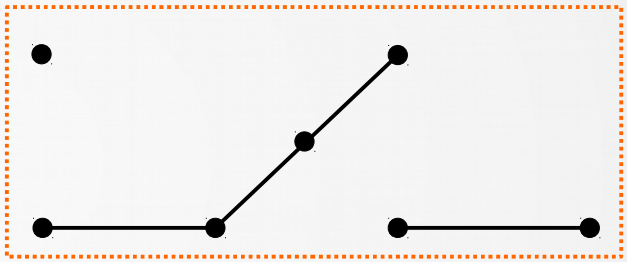
How connected is the graph ?

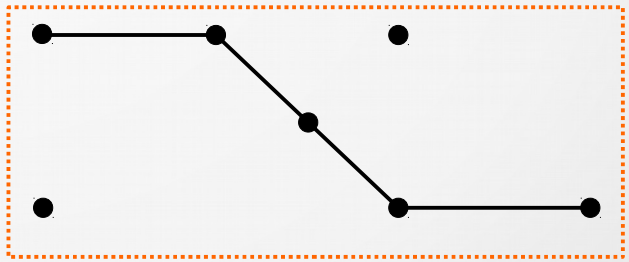
Example: Find the cut vertices and cut edges in the graph below.

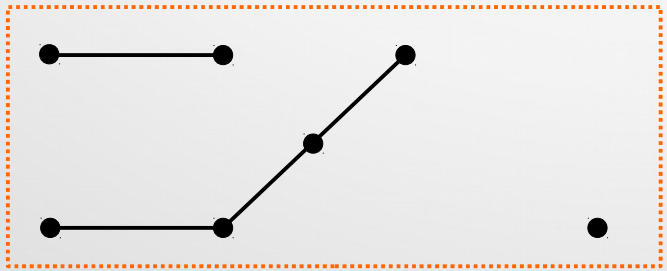
cut vertices: b,c,d,f,g

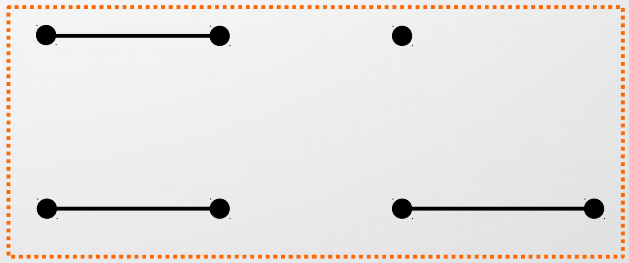


G







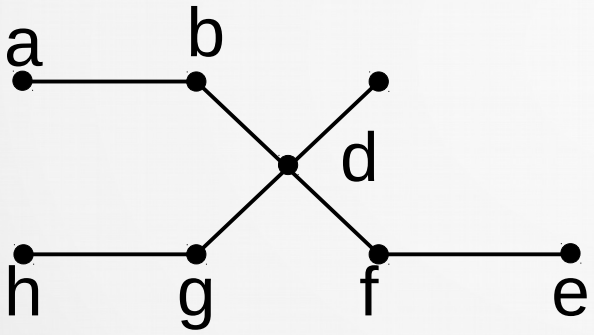


10.4 *Connectivity*

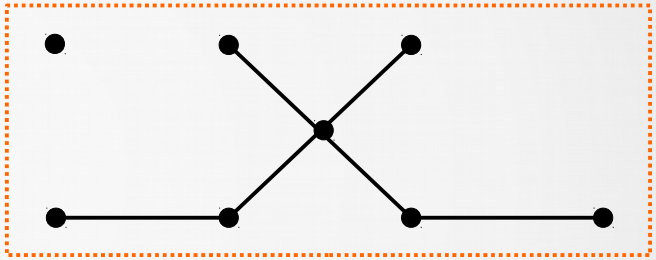
How connected is the graph ?

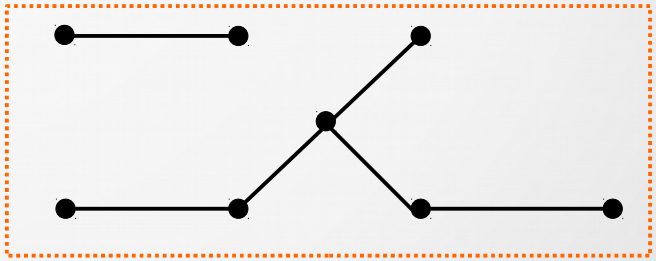
Example: Find the cut vertices and cut edges in the graph below.

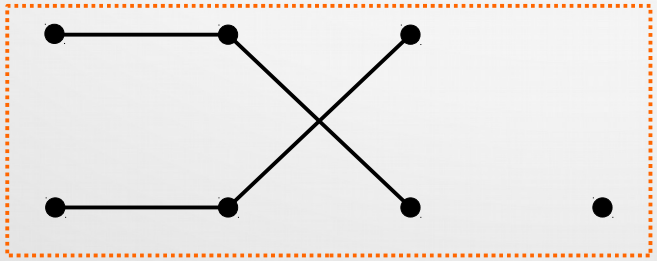
cut edges: (a,b), (b,d), (c,d), (f,e),

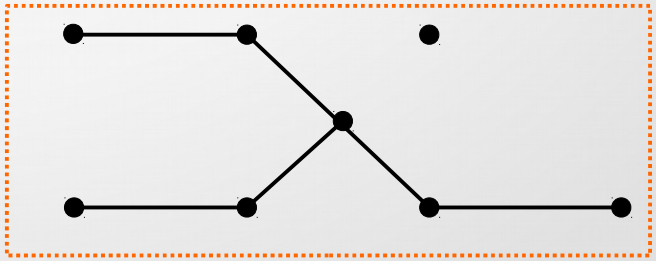


G









10.4 *Connectivity*

How connected is the graph ?

Example: Find the cut vertices and cut edges in the graph below.

cut edges: ..., (f,d), (d,g), (g,h)