Introduction to Graphs

Graphs!

Introduction to Graphs

Graphs!
Euler

Introduction to Graphs

Graphs!
- Euler
- Definitions: model.

Introduction to Graphs

Graphs!
- Euler
- Definitions: model.
- Euler Again!!

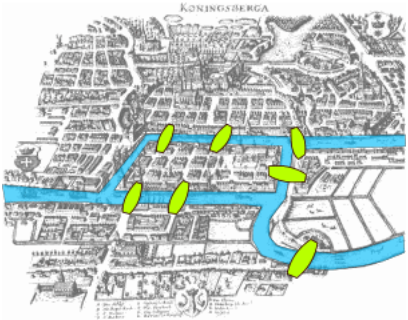
Introduction to Graphs

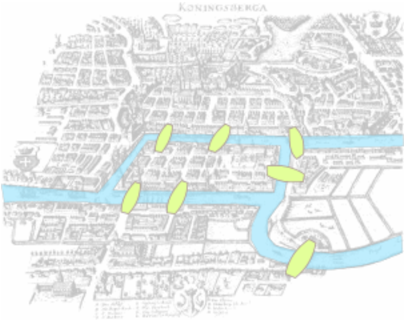
Graphs!
  Euler
  Definitions: model.
  Euler Again!!

Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

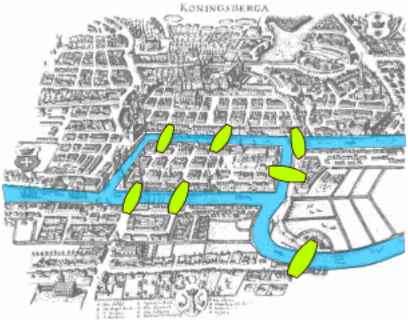


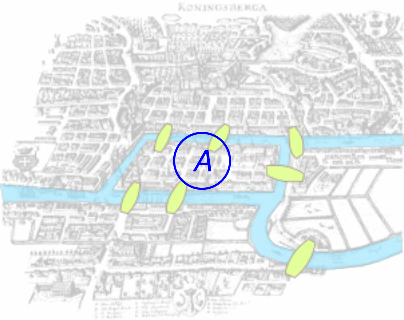


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

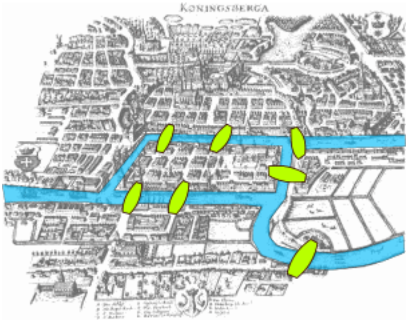


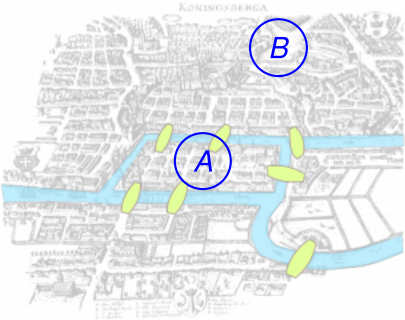


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

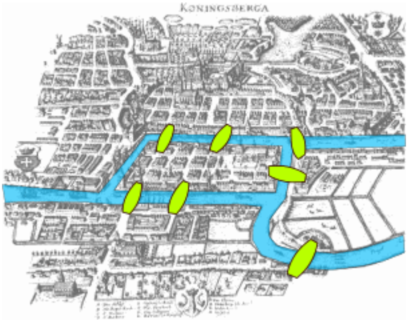


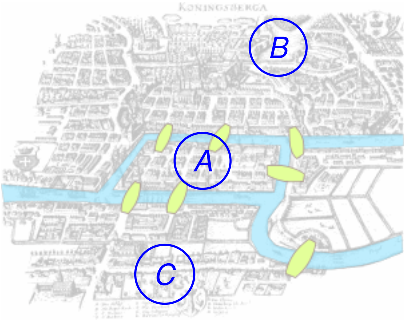


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

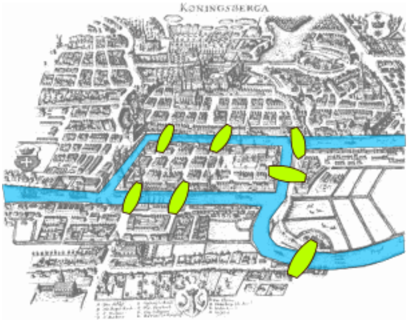


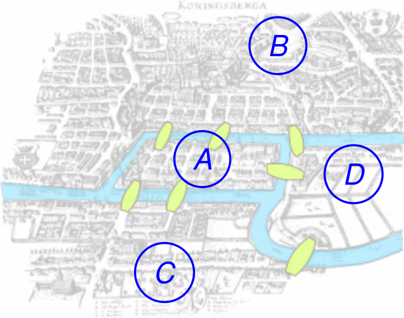


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

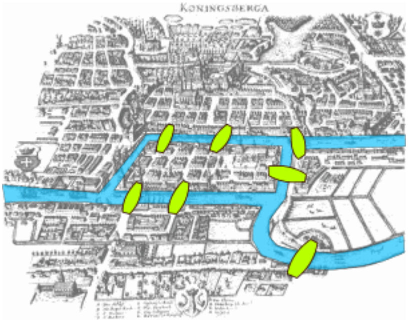


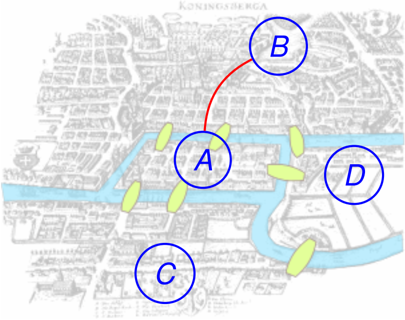


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

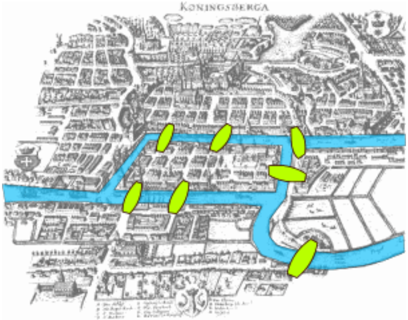


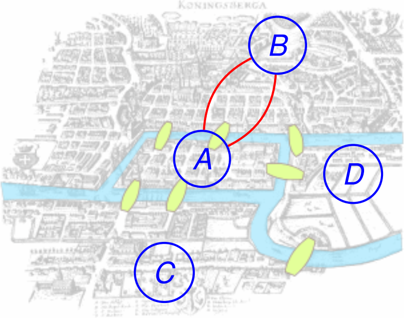


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

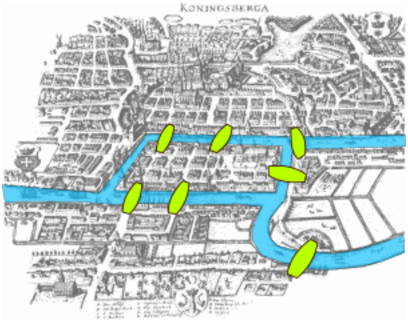


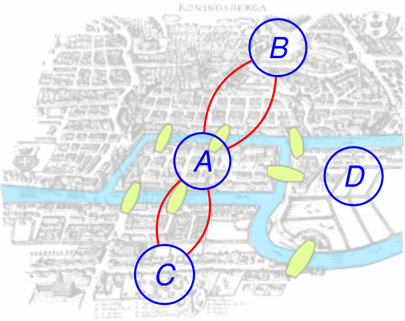


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

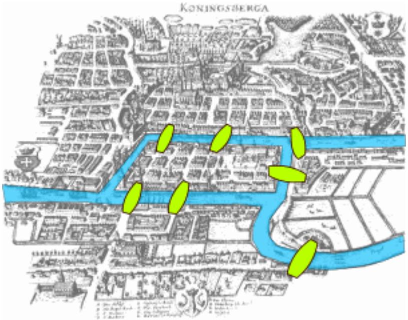


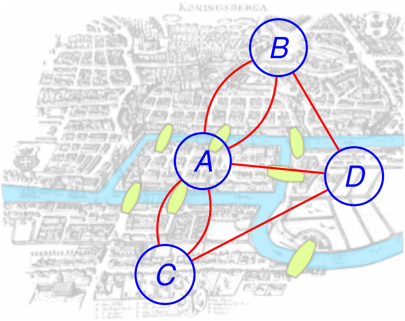


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

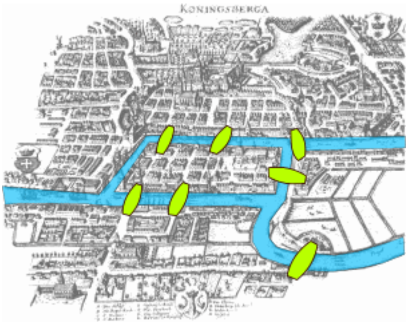


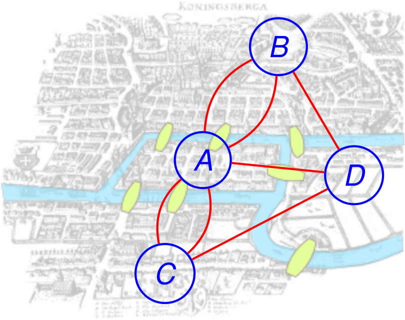


Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.



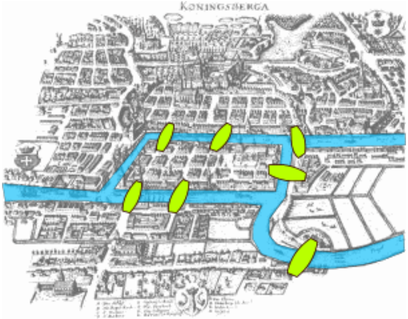


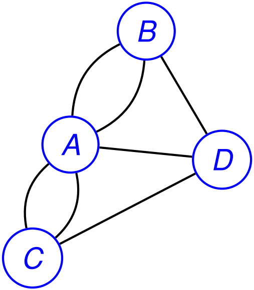
Can you draw a tour in the graph where you visit each edge once?

Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.



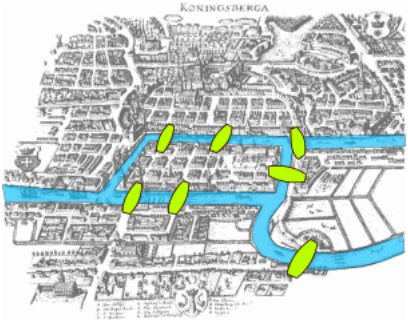


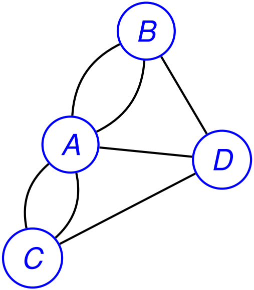
Can you draw a tour in the graph where you visit each edge once?   Yes?

Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.



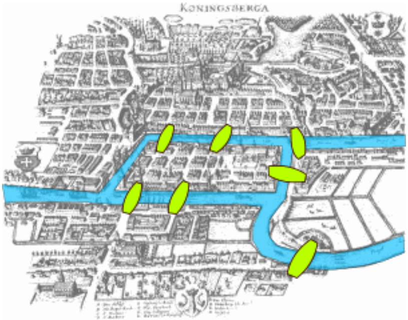


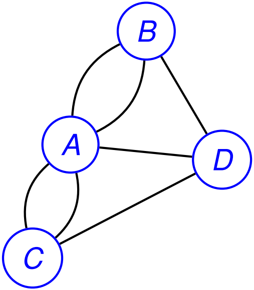
Can you draw a tour in the graph where you visit each edge once?   Yes?   No?

Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

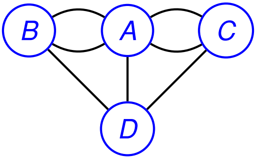




Can you draw a tour in the graph where you visit each edge once? Yes? No?

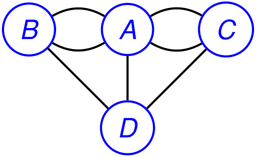
We will see!

Graphs: formally.



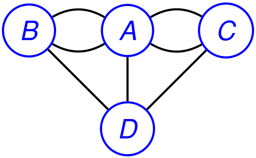
Graph:

Graphs: formally.



Graph: $G = (V, E)$.

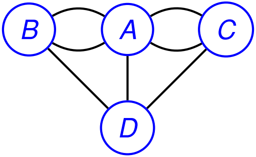
Graphs: formally.



Graph: $G = (V, E)$.

V - set of vertices.

Graphs: formally.

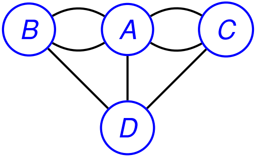


Graph: $G = (V, E)$.

V - set of vertices.

$\{A, B, C, D\}$

Graphs: formally.



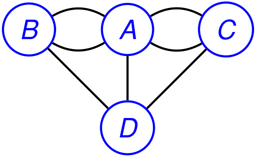
Graph: $G = (V, E)$.

V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ -

Graphs: formally.



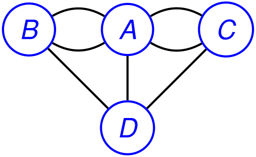
Graph: $G = (V, E)$.

V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ - set of edges.

Graphs: formally.



Graph: $G = (V, E)$.

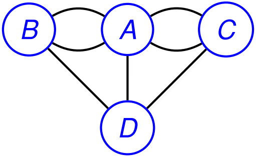
V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ - set of edges.

$\{\{A, B\}$

Graphs: formally.



Graph: $G = (V, E)$.

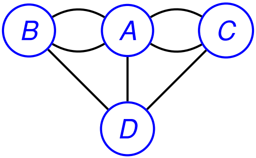
V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ - set of edges.

$\{\{A, B\}, \{A, B\}$

Graphs: formally.



Graph: $G = (V, E)$.

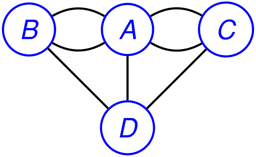
V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ - set of edges.

$\{\{A, B\}, \{A, B\}, \{A, C\},$

Graphs: formally.



Graph: $G = (V, E)$.

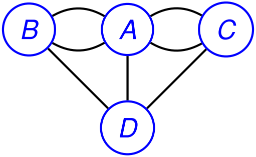
V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ - set of edges.

$\{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$.

Graphs: formally.



Graph: $G = (V,E)$.

V - set of vertices.

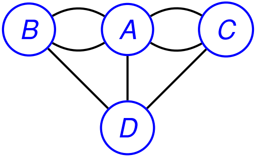
$\{A,B,C,D\}$

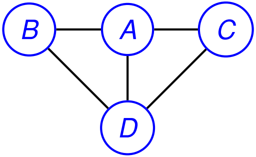
$E \subseteq V \times V$ - set of edges.

$\{\{A,B\},\{A,B\},\{A,C\},\{A,C\},\{B,D\},\{A,D\},\{C,D\}\}$.

For CS 70, usually simple graphs.

Graphs: formally.





Graph: $G = (V, E)$.

V - set of vertices.

$\{A, B, C, D\}$

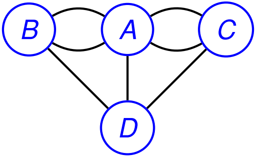
$E \subseteq V \times V$ - set of edges.

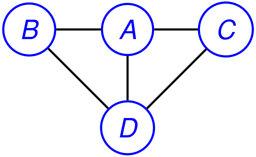
$\{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$.

For CS 70, usually simple graphs.

No parallel edges.

Graphs: formally.





Graph: $G = (V, E)$.

V - set of vertices.

$\{A, B, C, D\}$

$E \subseteq V \times V$ - set of edges.

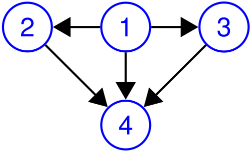
$\{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$.

For CS 70, usually simple graphs.

No parallel edges.

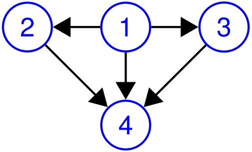
Multigraph above.

Directed Graphs



$G = (V, E)$.

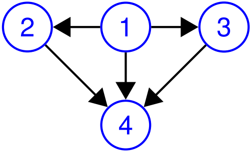
Directed Graphs



$G = (V, E)$.

V - set of vertices.

Directed Graphs

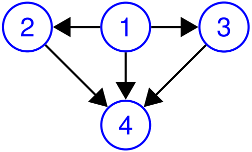


$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

Directed Graphs



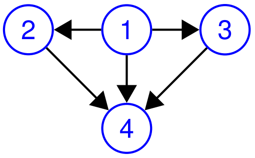
$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

Directed Graphs



$G = (V, E)$.

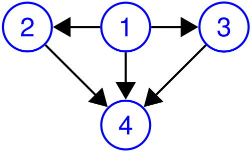
V - set of vertices.

$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1, 2),$

Directed Graphs



$G = (V, E)$.

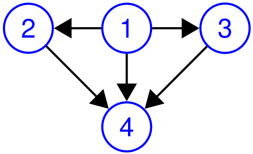
V - set of vertices.

$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1, 2), (1, 3),$

Directed Graphs



$G = (V, E)$.

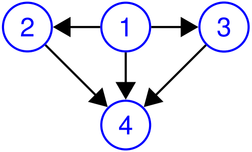
V - set of vertices.

$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1, 2), (1, 3), (1, 4),$

Directed Graphs



$G = (V, E)$.

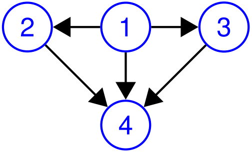
V - set of vertices.

$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

Directed Graphs



$G = (V, E)$.

V - set of vertices.

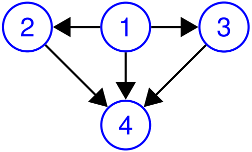
$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.

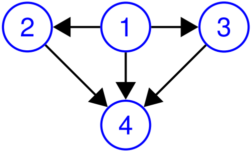
Directed Graphs



$G = (V, E)$.
V - set of vertices.
$\{1, 2, 3, 4\}$
E ordered pairs of vertices.
$\{(1,2), (1,3), (1,4), (2,4), (3,4)\}$

One way streets.
Tournament:

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

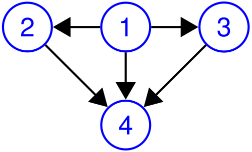
E ordered pairs of vertices.

$\{(1,2), (1,3), (1,4), (2,4), (3,4)\}$

One way streets.

Tournament: 1 beats 2,

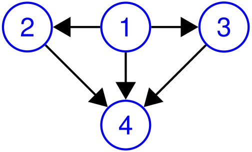
Directed Graphs



$G = (V, E)$.
V - set of vertices.
$\{1, 2, 3, 4\}$
E ordered pairs of vertices.
$\{(1,2), (1,3), (1,4), (2,4), (3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence:

Directed Graphs



$G = (V, E)$.

V - set of vertices.

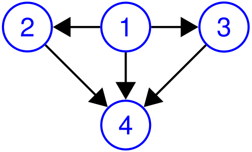
$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2,

Directed Graphs



$G = (V, E)$.

V - set of vertices.

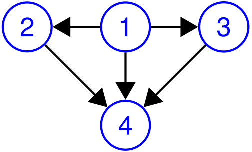
$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1,2), (1,3), (1,4), (2,4), (3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

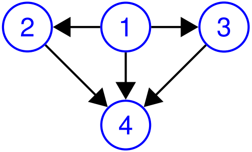
E ordered pairs of vertices.

$\{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network:

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

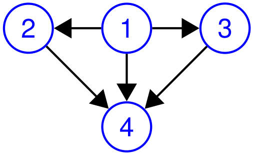
E ordered pairs of vertices.

$\{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed?

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

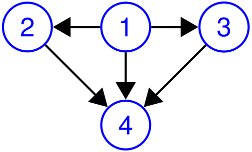
E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed? Undirected?

Directed Graphs

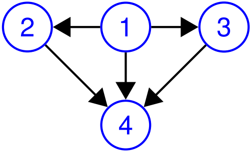


$G = (V, E)$.
V - set of vertices.
$\{1, 2, 3, 4\}$
E ordered pairs of vertices.
$\{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed? Undirected?
Friends.

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

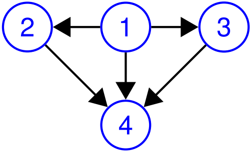
E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed? Undirected?
Friends. Undirected.

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

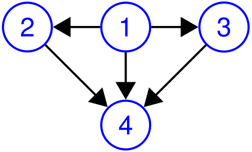
E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed? Undirected?
Friends. Undirected.
Likes.

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

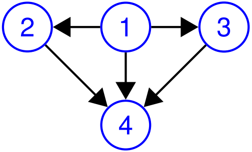
E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed? Undirected?
Friends. Undirected.
Likes. Directed.

Directed Graphs



$G = (V, E)$.

V - set of vertices.

$\{1, 2, 3, 4\}$

E ordered pairs of vertices.

$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

One way streets.
Tournament: 1 beats 2, ...
Precedence: 1 is before 2, ..

Social Network: Directed? Undirected?
Friends. Undirected.
Likes. Directed.

Graph Concepts and Definitions.

Graph: $G = (V, E)$

Graph Concepts and Definitions.

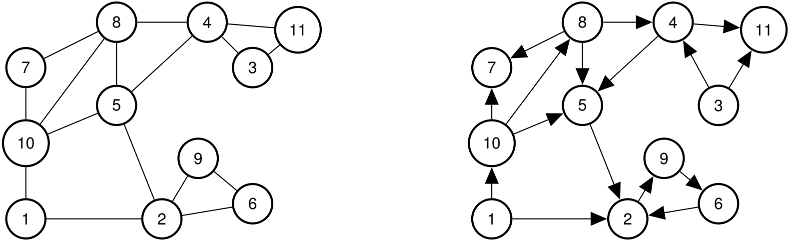
Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree

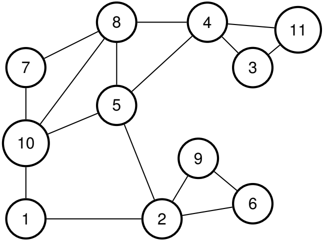


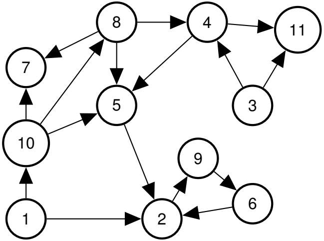
Neighbors of 10?

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



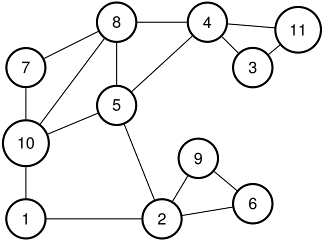


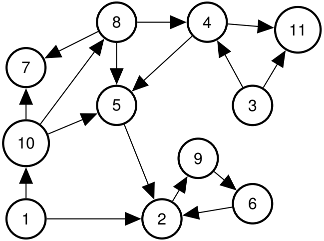
Neighbors of 10? 1,

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



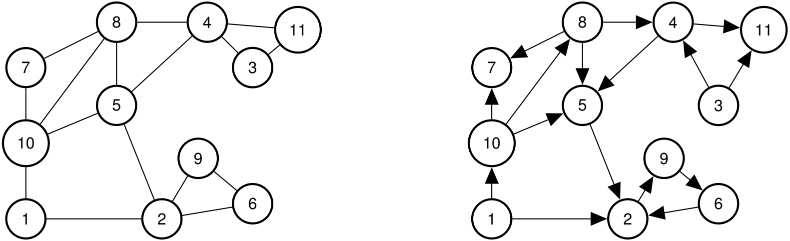


Neighbors of 10? 1,5,

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree

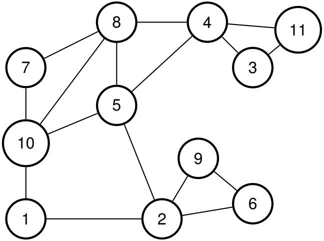


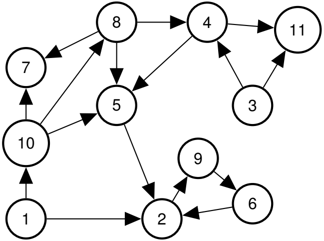
Neighbors of 10? 1,5,7,

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



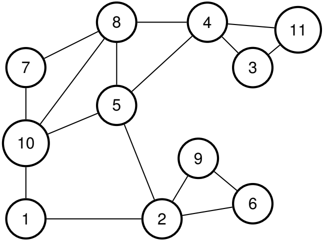


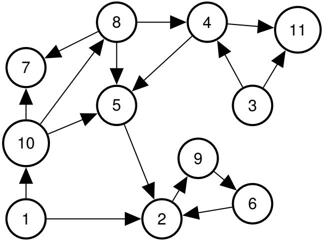
Neighbors of 10? 1,5,7, 8.

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





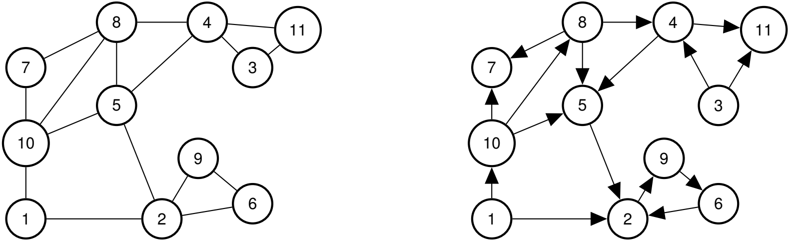
Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



Neighbors of 10? 1,5,7, 8.

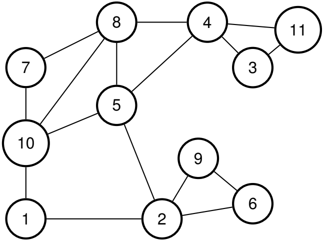
u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

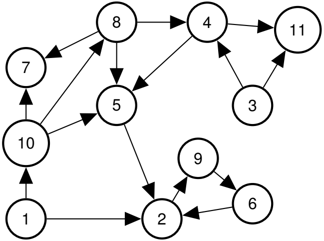
Edge $(10, 5)$ is incident to

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

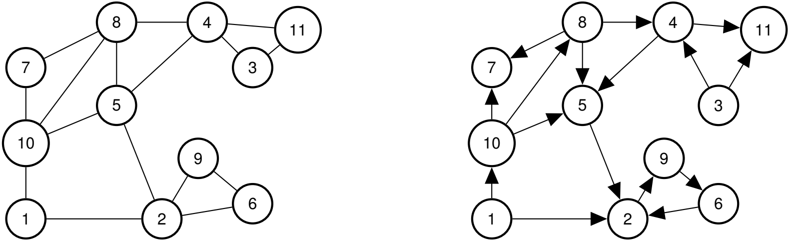
Edge (u, v) is incident to u and v.

Degree of vertex 1?

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

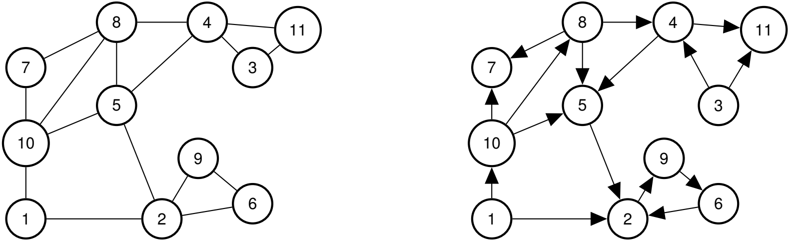
Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

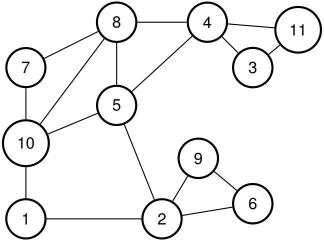
Degree of vertex 1? 2

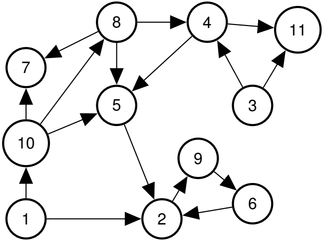
Degree of vertex u is number of incident edges.

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

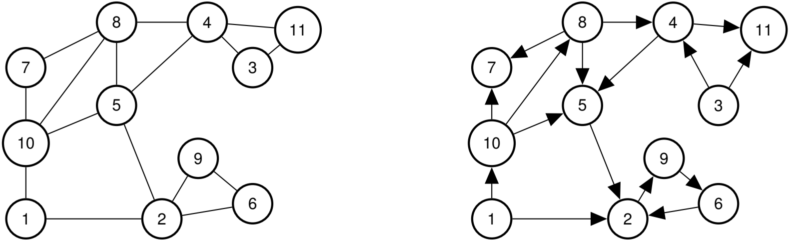
Degree of vertex u is number of incident edges.

Equals number of neighbors in simple graph.

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

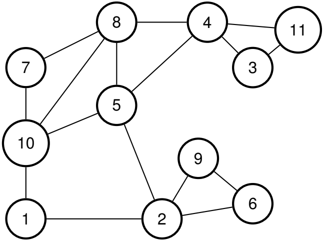
Degree of vertex u is number of incident edges.

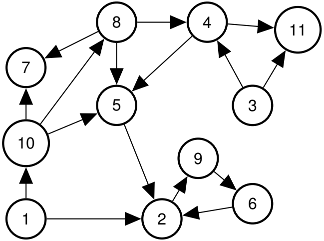
Equals number of neighbors in simple graph.

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

Degree of vertex u is number of incident edges.

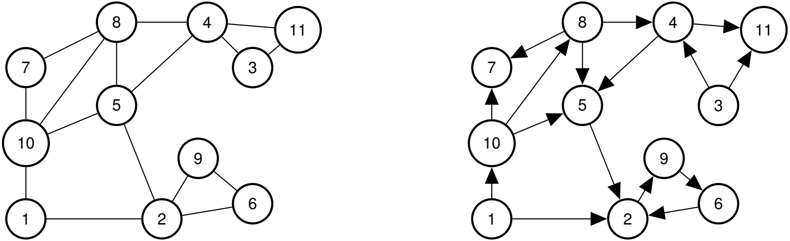
Equals number of neighbors in simple graph.

Directed graph?

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree



Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u,v) \in E$ (or if $(v,u) \in E$).

Edge $(10,5)$ is incident to vertex 10 and vertex 5.

Edge (u,v) is incident to u and v.

Degree of vertex 1? 2

Degree of vertex u is number of incident edges.

Equals number of neighbors in simple graph.

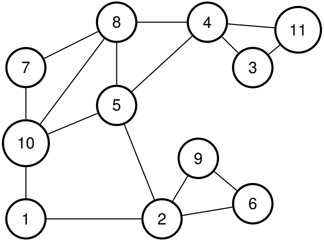
Directed graph?

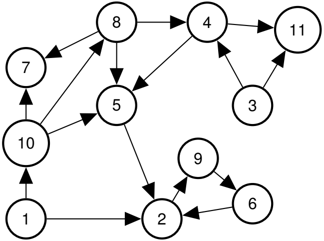
In-degree of 10?

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

Degree of vertex u is number of incident edges.

Equals number of neighbors in simple graph.

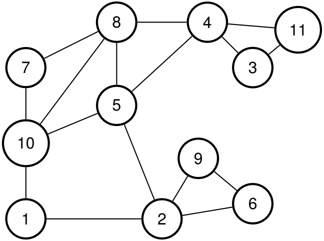
Directed graph?

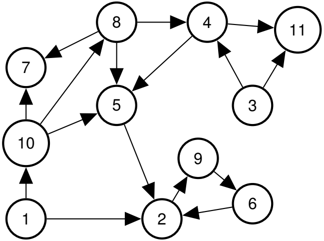
In-degree of 10? 1

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

Degree of vertex u is number of incident edges.

Equals number of neighbors in simple graph.

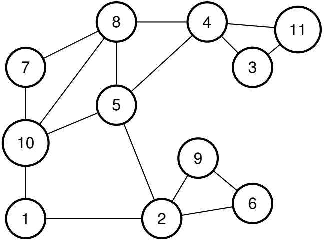
Directed graph?

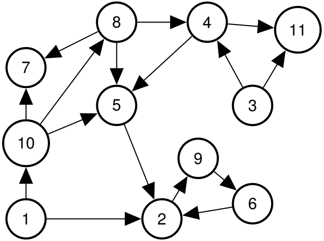
In-degree of 10? 1   Out-degree of 10?

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

Degree of vertex u is number of incident edges.

Equals number of neighbors in simple graph.

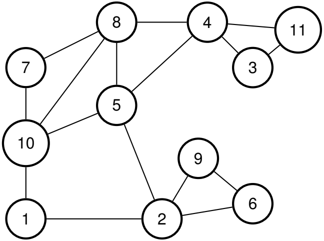
Directed graph?

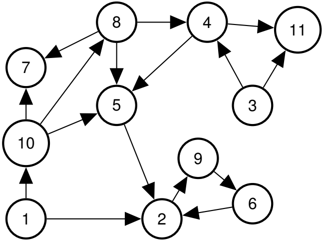
In-degree of 10? 1   Out-degree of 10? 3

Graph Concepts and Definitions.

Graph: $G = (V, E)$

neighbors, adjacent, degree, incident, in-degree, out-degree





Neighbors of 10? 1,5,7, 8.

u is neighbor of v if $(u, v) \in E$ (or if $(v, u) \in E$).

Edge $(10, 5)$ is incident to vertex 10 and vertex 5.

Edge (u, v) is incident to u and v.

Degree of vertex 1? 2

Degree of vertex u is number of incident edges.

Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10? 1   Out-degree of 10? 3

Quick Proof.

The sum of the vertex degrees is equal to

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)!

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

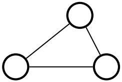
Not (A)! Triangle.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



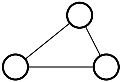
Not (B)!

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



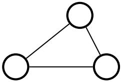
Not (B)! Triangle.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



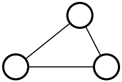
Not (B)! Triangle.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

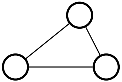
What could it be?

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

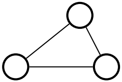
What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.

 Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

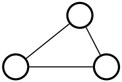
Could it always be...

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

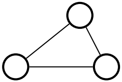
Could it always be...$2|E|$?

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

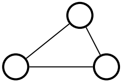
How many incidences does each edge contribute?

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

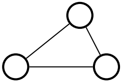
How many incidences does each edge contribute? 2.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

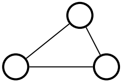
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

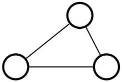
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree v?

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

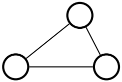
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree v? incidences contributed to v!

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

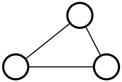
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree v? incidences contributed to v!
sum of degrees is total incidences

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

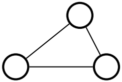
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree v? incidences contributed to v!
sum of degrees is total incidences ... or $2|E|$.

Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) Something else?

Not (A)! Triangle.



Not (B)! Triangle.

What could it be? For triangle number of edges is 3, the sum of degrees is 6.

Could it always be...$2|E|$?

How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree v? incidences contributed to v!
sum of degrees is total incidences ... or $2|E|$.

Thm: Sum of vertex degrees is $2|E|$.

Proof of “handshake” lemma.

Lemma: The sum of degrees is $2|E|$, for a graph $G = (V, E)$.

The number of edge-vertex incidences for an edge e is 2.

The total number of edge-vertex incidences is $2|E|$.

The sum of degrees is $2|E|$.

Proof of "handshake" lemma.

Lemma: The sum of degrees is $2|E|$, for a graph $G = (V, E)$.

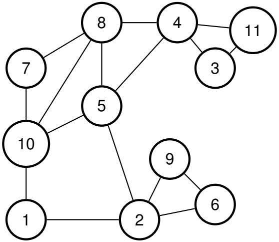
The number of edge-vertex incidences for an edge e is 2.

The total number of edge-vertex incidences is $2|E|$.

The sum of degrees is $2|E|$.

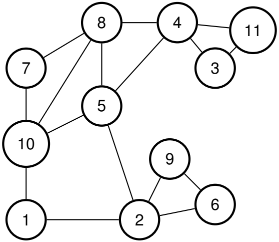
Handshake lemma: sum of number of handshakes of each person is twice the number of handshakes.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

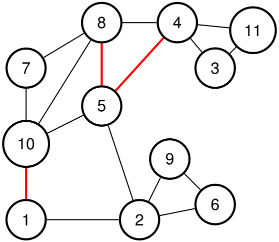
Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path?

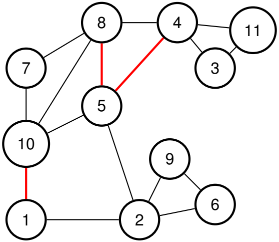
Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1, 10\}$, $\{8, 5\}$, $\{4, 5\}$?

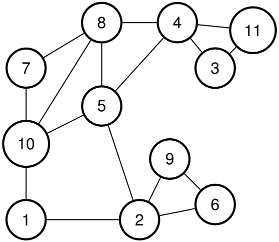
Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!

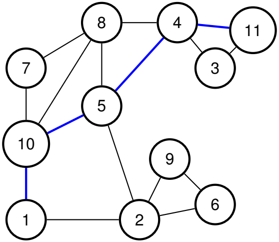
Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1, 10\}$, $\{8, 5\}$, $\{4, 5\}$? No!
Path?

Paths, walks, cycles, tour.

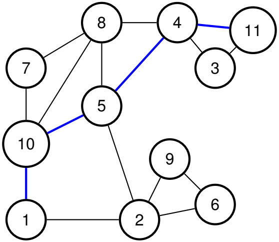


A path in a graph is a sequence of edges.

Path? $\{1, 10\}$, $\{8, 5\}$, $\{4, 5\}$? No!

Path? $\{1, 10\}$, $\{10, 5\}$, $\{5, 4\}$, $\{4, 11\}$?

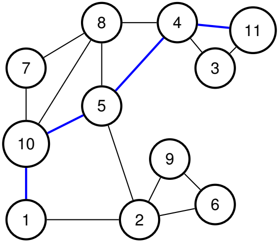
Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Paths, walks, cycles, tour.

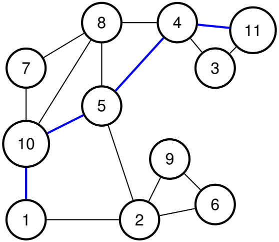


A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

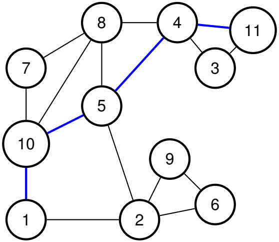
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!

Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Quick Check!

Paths, walks, cycles, tour.



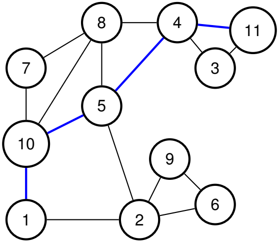
A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Quick Check! Length of path?

Paths, walks, cycles, tour.



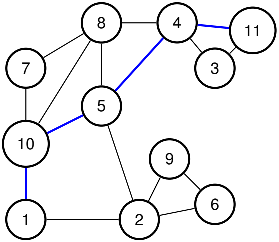
A path in a graph is a sequence of edges.

Path? $\{1, 10\}$, $\{8, 5\}$, $\{4, 5\}$? No!
Path? $\{1, 10\}$, $\{10, 5\}$, $\{5, 4\}$, $\{4, 11\}$? Yes!

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Quick Check! Length of path? k vertices

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

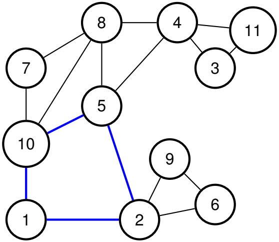
Path? $\{1,10\}, \{8,5\}, \{4,5\}$? No!

Path? $\{1,10\}, \{10,5\}, \{5,4\}, \{4,11\}$? Yes!

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Quick Check! Length of path? k vertices or $k-1$ edges.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!

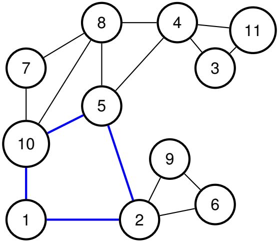
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!

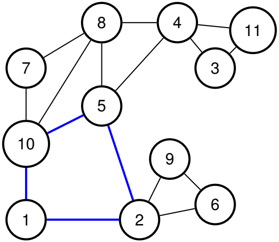
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle?

Paths, walks, cycles, tour.



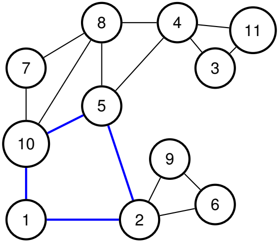
A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!

Paths, walks, cycles, tour.



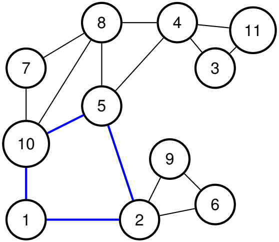
A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually *simple*.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!

Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

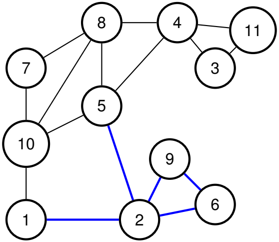
Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!

Path is usually *simple*. No repeated vertex!

Paths, walks, cycles, tour.



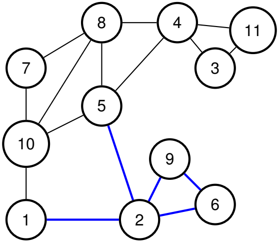
A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually *simple*. No repeated vertex!
Walk is sequence of edges with possible repeated vertex or edge.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

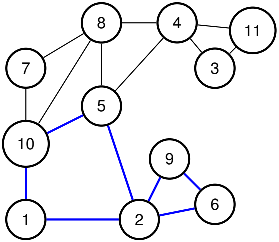
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!

Path is usually *simple*. No repeated vertex!

Walk is sequence of edges with possible repeated vertex or edge.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Quick Check! Length of path? k vertices or $k-1$ edges.

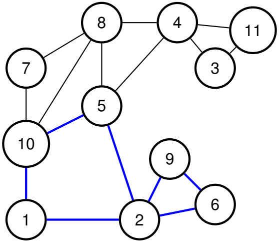
Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!

Path is usually *simple*. No repeated vertex!

Walk is sequence of edges with possible repeated vertex or edge.

Tour is walk that starts and ends at the same node.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.

Quick Check! Length of path? k vertices or $k-1$ edges.

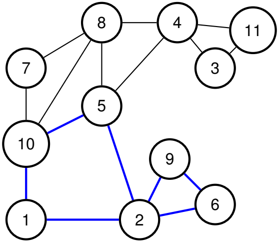
Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!

Path is usually *simple*. No repeated vertex!

Walk is sequence of edges with possible repeated vertex or edge.

Tour is walk that starts and ends at the same node.

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

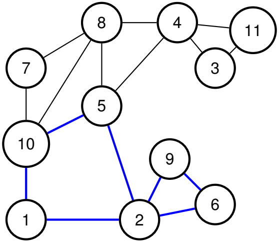
Path: $(v_1,v_2),(v_2,v_3),\ldots(v_{k-1},v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually *simple*. No repeated vertex!
Walk is sequence of edges with possible repeated vertex or edge.

Tour is walk that starts and ends at the same node.

Quick Check!

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

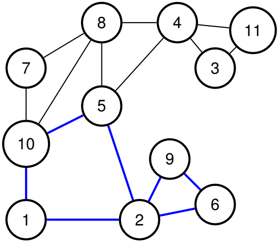
Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually *simple*. No repeated vertex!
Walk is sequence of edges with possible repeated vertex or edge.

Tour is walk that starts and ends at the same node.

Quick Check!
Path is to Walk as Cycle is to ??

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

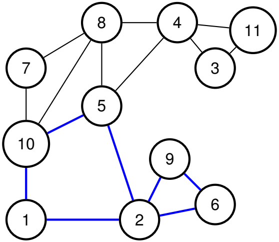
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually *simple*. No repeated vertex!
Walk is sequence of edges with possible repeated vertex or edge.
Tour is walk that starts and ends at the same node.

Quick Check!
Path is to Walk as Cycle is to ?? Tour!

Paths, walks, cycles, tour.



A path in a graph is a sequence of edges.

Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

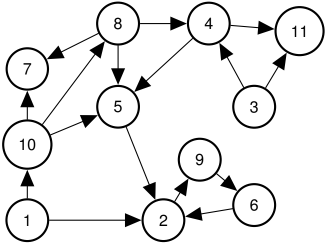
Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Quick Check! Length of path? k vertices or $k-1$ edges.

Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually *simple*. No repeated vertex!
Walk is sequence of edges with possible repeated vertex or edge.

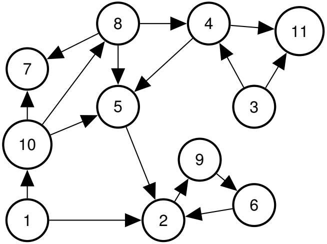
Tour is walk that starts and ends at the same node.

Quick Check!
Path is to Walk as Cycle is to ?? Tour!

Directed Paths.

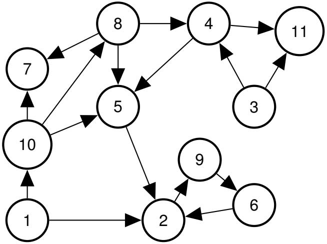


Directed Paths.



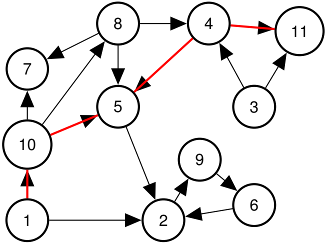
Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Directed Paths.



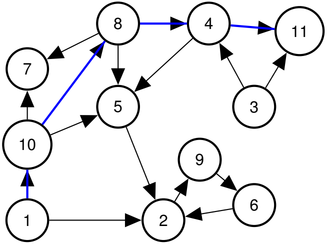
Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Directed Paths.



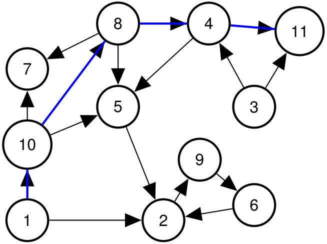
Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Directed Paths.



Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

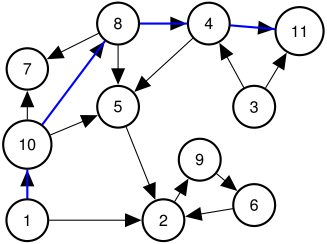
Directed Paths.



Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Paths,

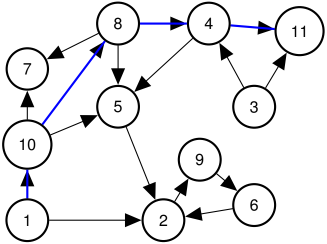
Directed Paths.



Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Paths, walks,

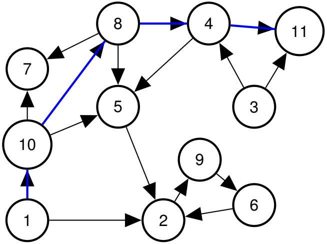
Directed Paths.



Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

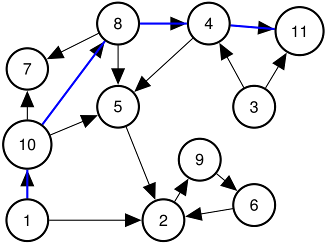
Paths, walks, cycles,

Directed Paths.



Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Paths, walks, cycles, tours

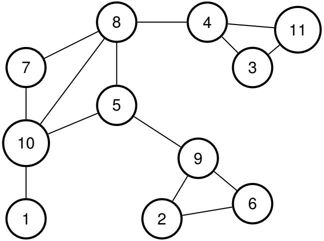
Directed Paths.



Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

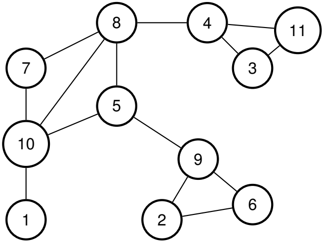
Paths, walks, cycles, tours ... are analogous to undirected now.

Connectivity



u and *v* are connected if there is a path between *u* and *v*.

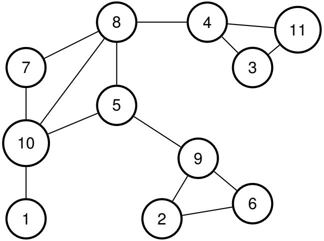
Connectivity



u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

Connectivity

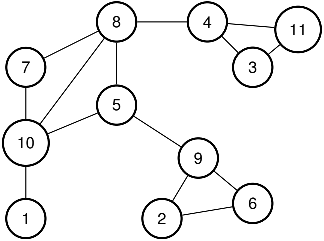


u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex *x* is connected to every other vertex.

Connectivity

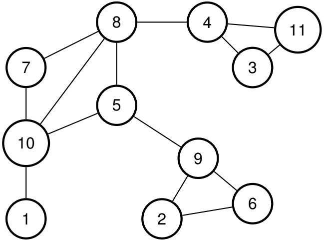


u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex *x* is connected to every other vertex.
Is graph connected?

Connectivity

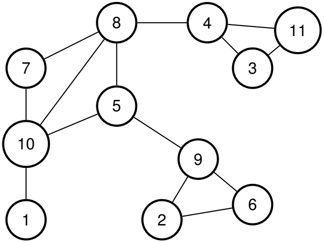


u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex *x* is connected to every other vertex.
Is graph connected? Yes?

Connectivity

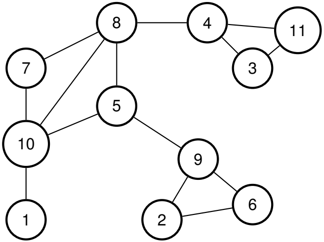


u and v are connected if there is a path between u and v.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex x is connected to every other vertex.
Is graph connected? Yes? No?

Connectivity



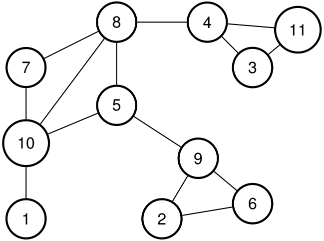
u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex *x* is connected to every other vertex.
Is graph connected? Yes? No?

Proof:

Connectivity



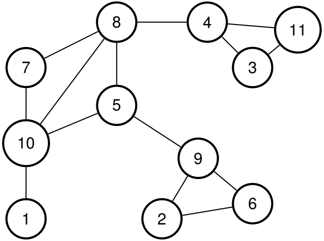
u and v are connected if there is a path between u and v.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex x is connected to every other vertex.
Is graph connected? Yes? No?

Proof: Use path from u to x and then from x to v.

Connectivity



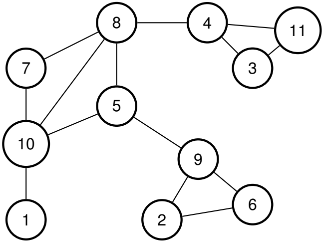
u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

If one vertex *x* is connected to every other vertex.
Is graph connected? Yes? No?

Proof: Use path from *u* to *x* and then from *x* to *v*. □

Connectivity



u and *v* are connected if there is a path between *u* and *v*.

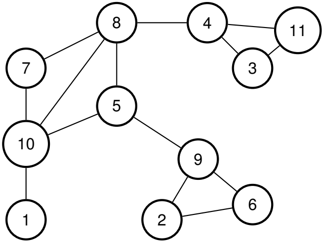
A connected graph is a graph where all pairs of vertices are connected.

If one vertex *x* is connected to every other vertex.
Is graph connected? Yes? No?

Proof: Use path from *u* to *x* and then from *x* to *v*. □

May not be simple!

Connectivity



u and v are connected if there is a path between u and v.

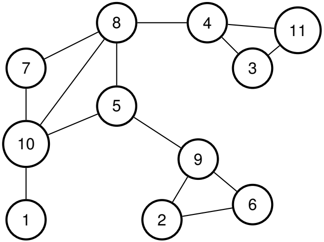
A connected graph is a graph where all pairs of vertices are connected.

If one vertex x is connected to every other vertex.
Is graph connected? Yes? No?

Proof: Use path from u to x and then from x to v. □

May not be simple!
Either modify definition to walk.

Connectivity



u and v are connected if there is a path between u and v.

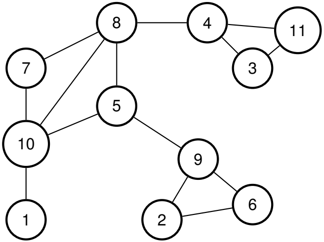
A connected graph is a graph where all pairs of vertices are connected.

If one vertex x is connected to every other vertex.
Is graph connected? Yes? No?

Proof: Use path from u to x and then from x to v. □

May not be simple!
Either modify definition to walk.
Or cut out cycles.

Connectivity



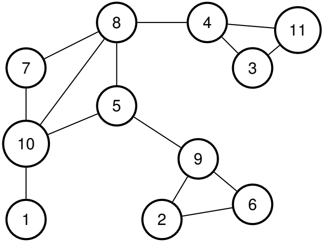
u and *v* are connected if there is a path between *u* and *v*.

A connected graph is a graph where all pairs of vertices are connected.

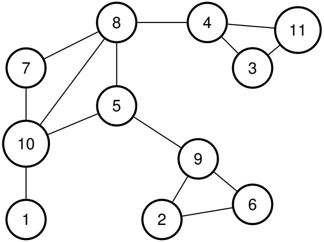
If one vertex *x* is connected to every other vertex.
Is graph connected? Yes? No?

Proof: Use path from *u* to *x* and then from *x* to *v*. □

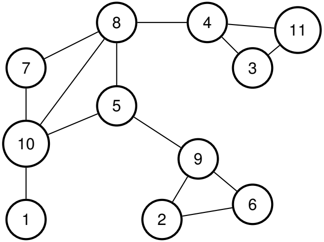
May not be simple!
Either modify definition to walk.
Or cut out cycles.



Is graph above connected?

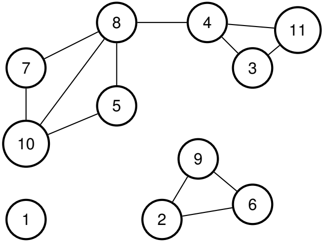


Is graph above connected? Yes!



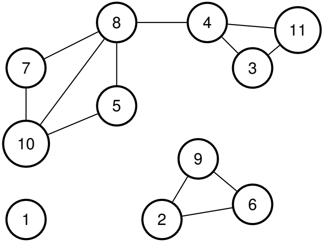
Is graph above connected? Yes!

How about now?



Is graph above connected? Yes!

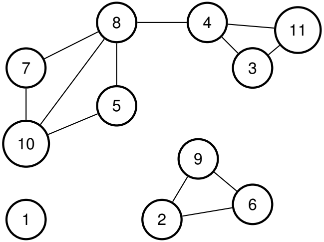
How about now? No!



Is graph above connected? Yes!

How about now? No!

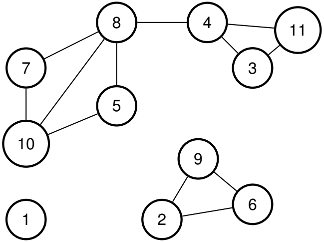
Connected Components?



Is graph above connected? Yes!

How about now? No!

Connected Components? $\{1\}, \{10, 7, 5, 8, 4, 3, 11\}, \{2, 9, 6\}$.
A connected component is a maximal set of connected nodes in a graph.
Quick Check: Is $\{10, 7, 5\}$ a connected component?



Is graph above connected? Yes!

How about now? No!

Connected Components? $\{1\},\{10,7,5,8,4,3,11\},\{2,9,6\}$.

A connected component is a maximal set of connected nodes in a graph.

Quick Check: Is $\{10,7,5\}$ a connected component? No.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. $\square$

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\Longrightarrow$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. □

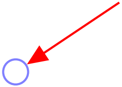
When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. $\square$



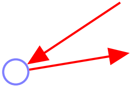
When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. □



When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

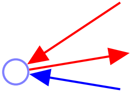
Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\Longrightarrow$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. $\square$



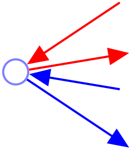
When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. □



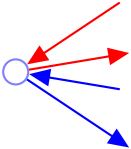
Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex *v* on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore *v* has even degree. □



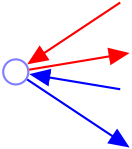
When you enter, you leave.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\Longrightarrow$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. □



When you enter, you leave.
For starting node,

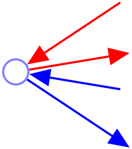
Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.

Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. $\square$



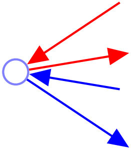
When you enter, you leave.
For starting node, tour leaves first

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. $\square$



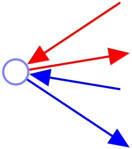
When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

Finally..back to Euler!

An Eulerian Tour is a tour that visits each edge exactly once.

Theorem: Any undirected graph has an Eulerian tour if and only if all vertices have even degree and is connected.
Proof of only if: Eulerian $\implies$ connected and all even degree.

Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex v on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore v has even degree. $\square$



When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm.

Finding a tour!

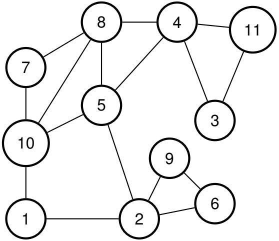
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.

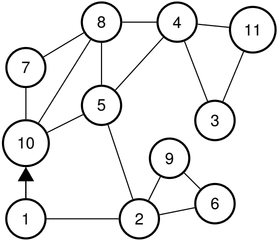
1. Take a walk starting from v (1)



Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

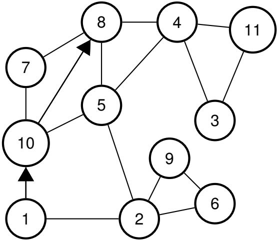
1. Take a walk starting from v (1)



Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

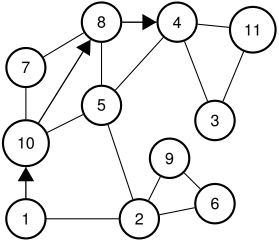
1. Take a walk starting from v (1)



Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

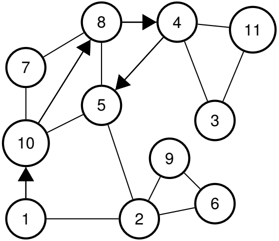
1. Take a walk starting from v (1)



Finding a tour!

Proof of if: Even + connected $\Longrightarrow$ Eulerian Tour.
We will give an algorithm. First by picture.

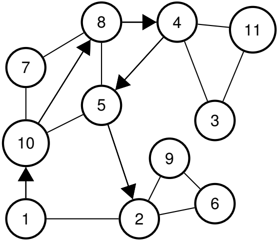
1. Take a walk starting from v (1)



Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

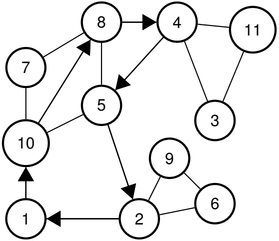
1. Take a walk starting from v (1)



Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



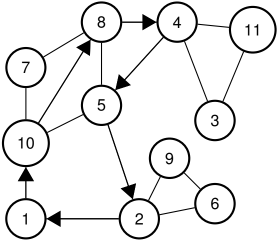
1. Take a walk starting from v (1)

... till you get back to v.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



1. Take a walk starting from v (1)

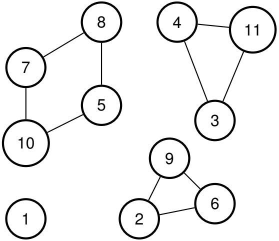
... till you get back to v.

2. Remove tour, C.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



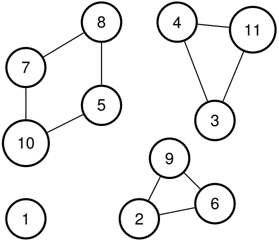
1. Take a walk starting from v (1)

... till you get back to v.

2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.

Finding a tour!

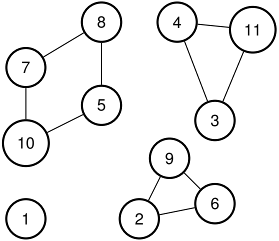
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.

Finding a tour!

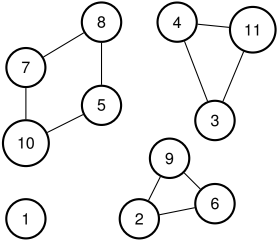
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \dots, G_k$ be connected components.
   Each is touched by C.
   Why?

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

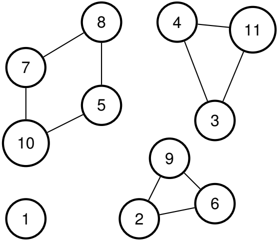


1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



1. Take a walk starting from v (1)

... till you get back to v.

2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.

    Each is touched by C.

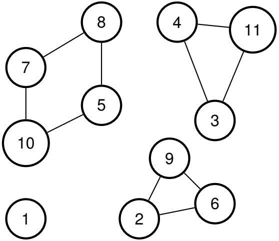
        Why? G was connected.

    Let v_i be (first) node in G_i touched by C.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



1. Take a walk starting from v (1)

... till you get back to v.

2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.

   Each is touched by C.

   Why? G was connected.

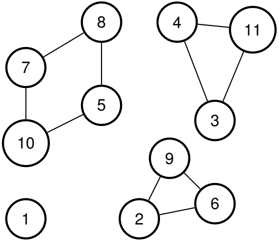
   Let v_i be (first) node in G_i touched by C.

   Example: $v_1 = 1$,

Finding a tour!

Proof of if: Even + connected $\Longrightarrow$ Eulerian Tour.

We will give an algorithm. First by picture.



1. Take a walk starting from v (1)

... till you get back to v.

2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.

   Each is touched by C.

   Why? G was connected.

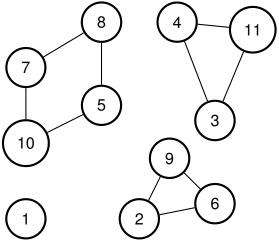
   Let v_i be (first) node in G_i touched by C.

   Example: $v_1 = 1$, $v_2 = 10$,

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



1. Take a walk starting from v (1)

... till you get back to v.

2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.

   Each is touched by C.

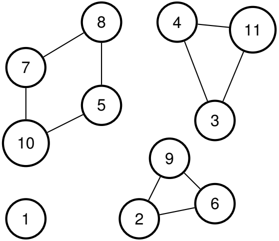
   Why? G was connected.

   Let v_i be (first) node in G_i touched by C.

   Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$,

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

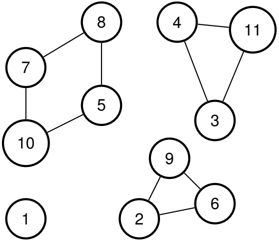


1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

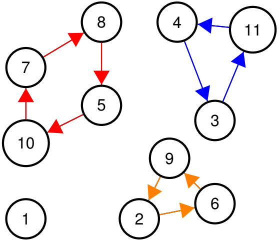
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
   Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
   Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i

Finding a tour!

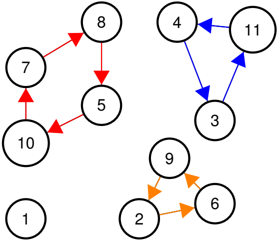
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i

Finding a tour!

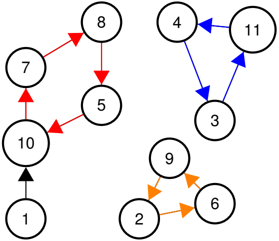
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
   Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
   Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.

Finding a tour!

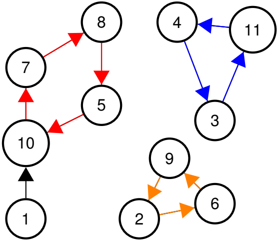
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.
   1,10

Finding a tour!

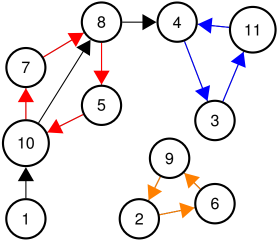
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.
   1,10,7,8,5,10

Finding a tour!

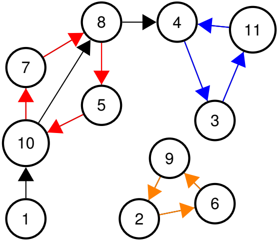
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.
   1,10,7,8,5,10 ,8,4

Finding a tour!

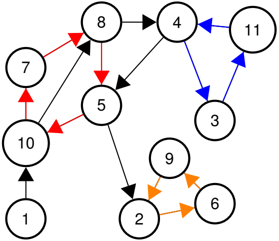
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.
   1,10,7,8,5,10 ,8,4,3,11,4

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

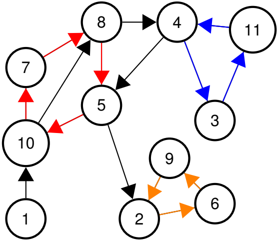


1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.
   1,10,7,8,5,10 ,8,4,3,11,4 5,2

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.



1. Take a walk starting from v (1)

... till you get back to v.

2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.

   Each is touched by C.

   Why? G was connected.

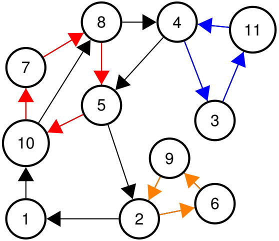
   Let v_i be (first) node in G_i touched by C.

   Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.

   1,10,7,8,5,10 ,8,4,3,11,4 5,2,6,9,2

Finding a tour!

Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.



1. Take a walk starting from v (1)
... till you get back to v.
2. Remove tour, C.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by C.
      Why? G was connected.
   Let v_i be (first) node in G_i touched by C.
    Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from v_i
5. Splice together.
   1,10,7,8,5,10 ,8,4,3,11,4 5,2,6,9,2 and to 1!

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!

Proof of Claim: Even degree. If enter, can leave

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!

Proof of Claim: Even degree. If enter, can leave except for v. $\square$

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.
Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\implies$

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\implies$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\implies$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. □

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\implies$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. $\square$

3. Find tour T_i of G_i

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. $\square$

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. $\square$

3. Find tour T_i of G_i starting/ending at v_i.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\implies$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. □

3. Find tour T_i of G_i starting/ending at v_i.

4. Splice T_i into C where v_i first appears in C.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. □

3. Find tour T_i of G_i starting/ending at v_i.

4. Splice T_i into C where v_i first appears in C.

Visits every edge once:
Visits edges in C

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. □

3. Find tour T_i of G_i starting/ending at v_i.

4. Splice T_i into C where v_i first appears in C.

Visits every edge once:
Visits edges in C exactly once.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\implies$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. □

3. Find tour T_i of G_i starting/ending at v_i.

4. Splice T_i into C where v_i first appears in C.

Visits every edge once:
Visits edges in C exactly once.
By induction for all other edges by induction on G_i.

General case: Recursive algorithm, proof by induction.

1. Take a walk from arbitrary node v, until you get back to v.

Claim: Do get back to v!
Proof of Claim: Even degree. If enter, can leave except for v. □

2. Remove cycle, C, from G.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$.

Let v_i be first vertex of C that is in G_i.
Why is there a v_i in C?
G was connected $\Longrightarrow$
a vertex in G_i must be incident to a removed edge in C.

Claim: Each vertex in each G_i has even degree and is connected.
Prf: Tour C has even incidences to any vertex v. □

3. Find tour T_i of G_i starting/ending at v_i.

4. Splice T_i into C where v_i first appears in C.

Visits every edge once:
Visits edges in C exactly once.
By induction for all other edges by induction on G_i. □

Summary

Graphs.

Summary

Graphs.
  Basics.

Summary

Graphs.

- Basics.

Connectivity.

Summary

Graphs.
  Basics.
Connectivity.
Algorithm for Eulerian Tour.

Summary

Graphs.
- Basics.

Connectivity.

Algorithm for Eulerian Tour.

Summary

Graphs.
  Basics.
Connectivity.
Algorithm for Eulerian Tour.