# Lecture 5: Graphs.

Graphs!

# Lecture 5: Graphs.

Graphs!
Euler

# Lecture 5: Graphs.

Graphs!
- Euler
- Definitions: model.

# Lecture 5: Graphs.

Graphs!

- Euler
- Definitions: model.
- Fact!

# Lecture 5: Graphs.

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

# Lecture 5: Graphs.

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

# Lecture 5: Graphs.

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

Planar graphs.

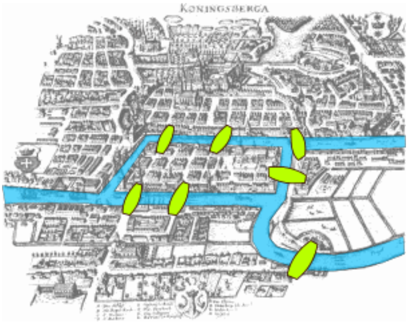
# Lecture 5: Graphs.

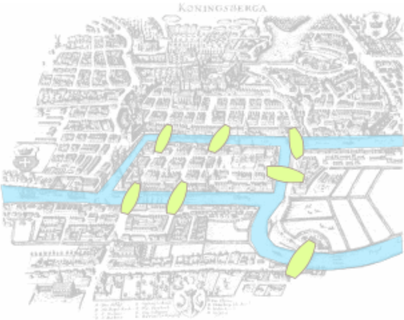
Graphs!
  Euler
  Definitions: model.
  Fact!
  Euler Again!!
Planar graphs.
  Euler Again!!!!

# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

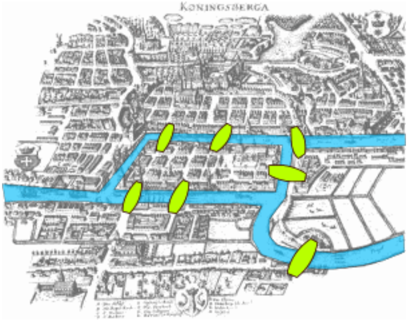


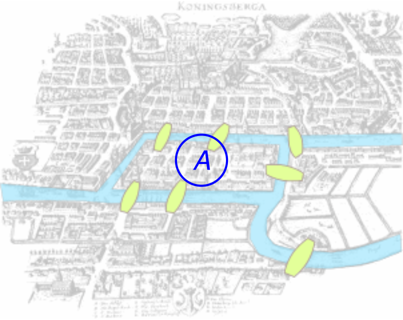



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

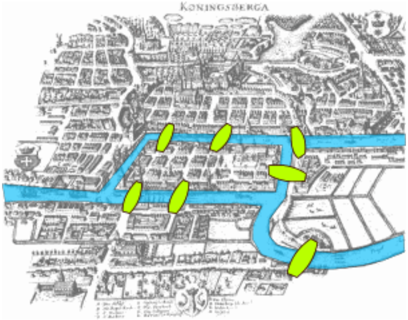


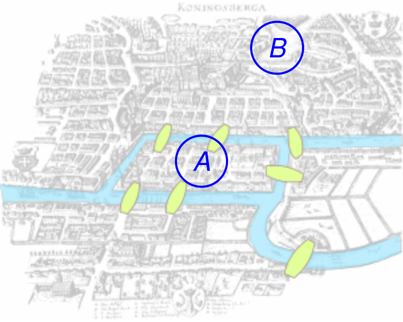



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

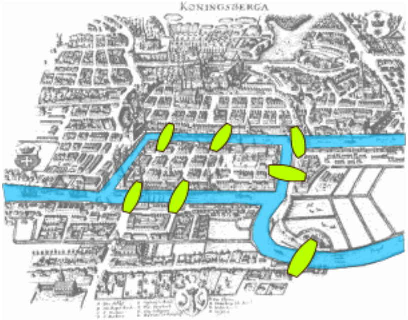


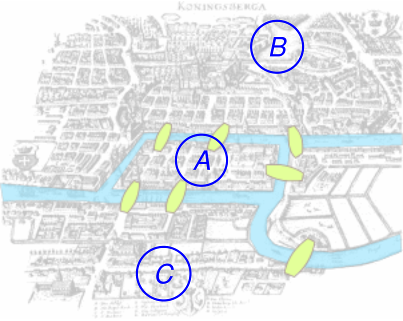



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

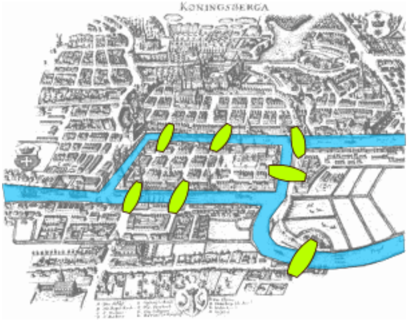


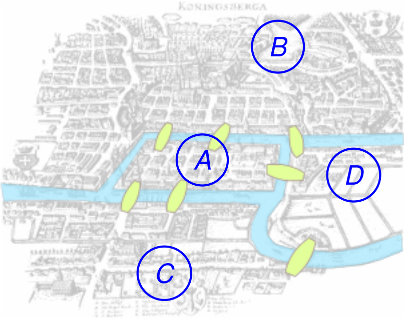



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

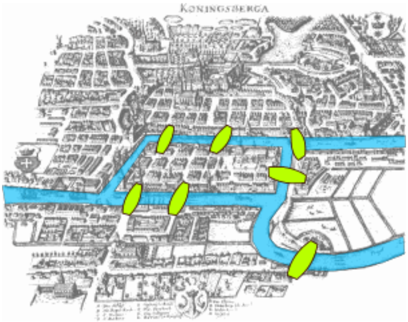


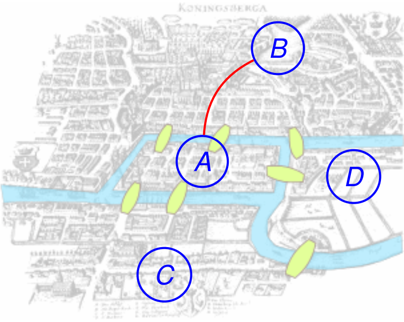



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

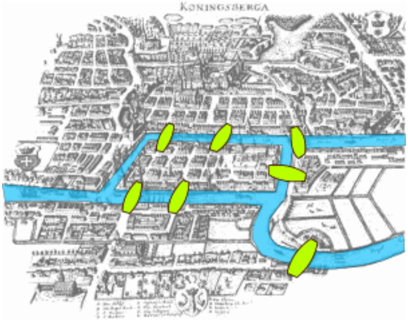


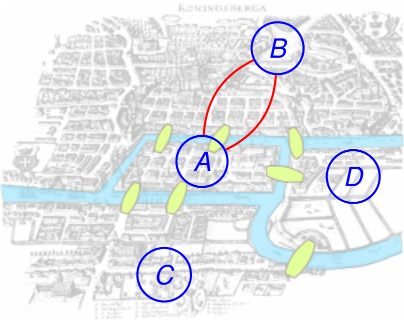


# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

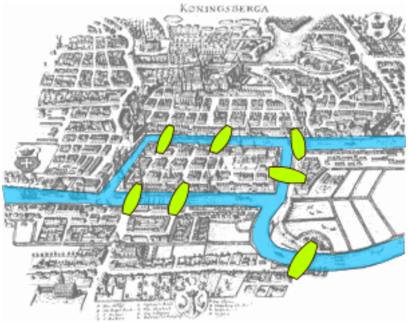


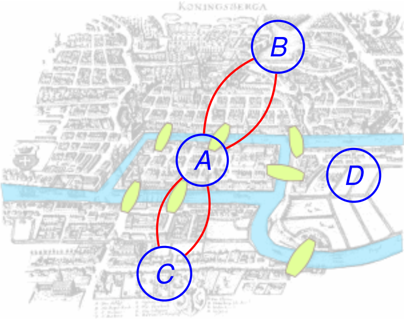



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

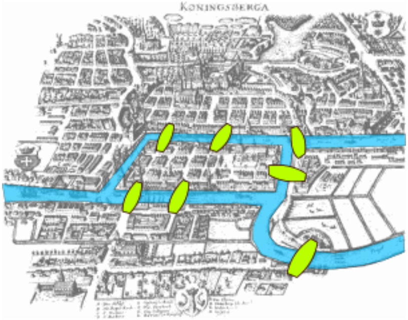


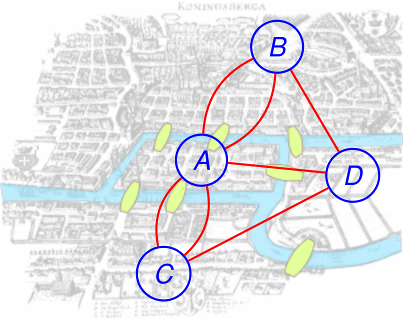



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.

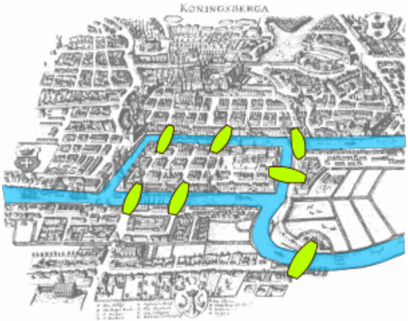


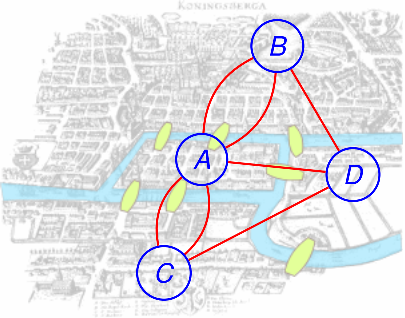



# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.



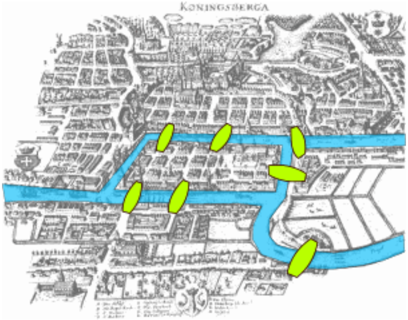



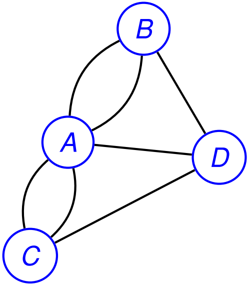

Can you draw a tour in the graph where you visit each edge once?

# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.



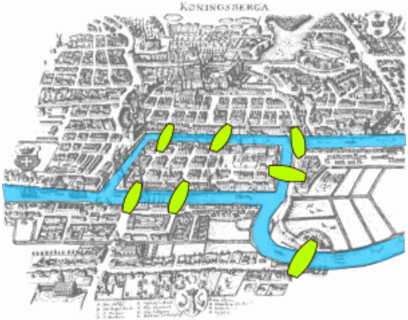


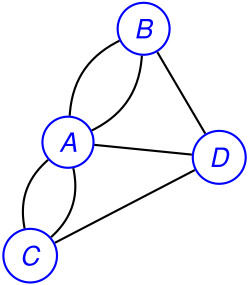

Can you draw a tour in the graph where you visit each edge once?
Yes?

# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuşcă - License.



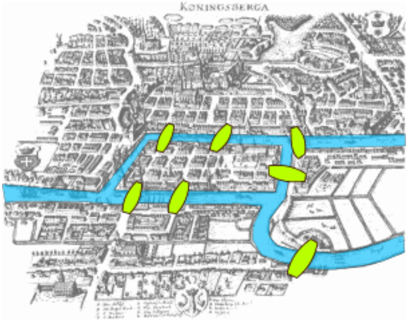



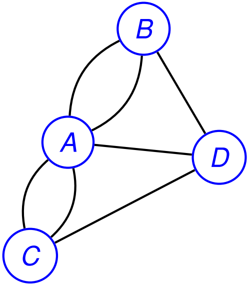

Can you draw a tour in the graph where you visit each edge once?
Yes?  No?

# Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

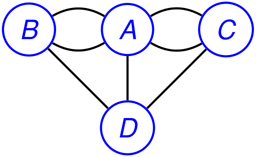
“Konigsberg bridges” by Bogdan Giuşcă - License.



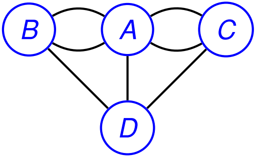

Can you draw a tour in the graph where you visit each edge once?
Yes? No?
We will see!

# Graphs: formally.



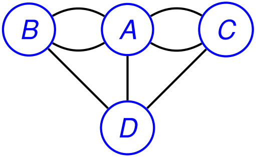

Graph:

# Graphs: formally.




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

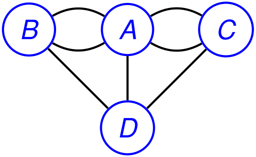
# Graphs: formally.




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

$V$ - set of vertices.

# Graphs: formally.

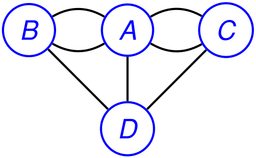



Graph: $G = (V, E)$.

$V$ - set of vertices.

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

# Graphs: formally.



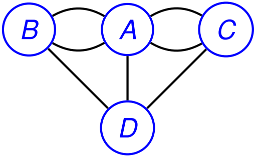

Graph: $G = (V, E)$.

$V$ - set of vertices.

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

$E \subseteq V \times V$ -

# Graphs: formally.



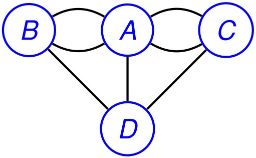

Graph: $G = (V, E)$.

$V$ - set of vertices.

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

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

# Graphs: formally.




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

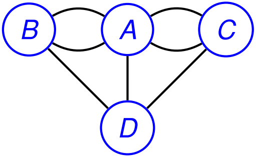
$V$ - set of vertices.

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

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

$\{\{A, B\}$

# Graphs: formally.




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

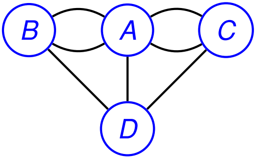
$V$ - set of vertices.

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

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

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

# Graphs: formally.




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

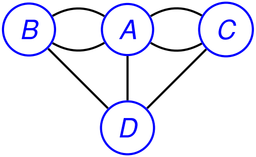
$V$ - set of vertices.

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

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

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

# Graphs: formally.




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

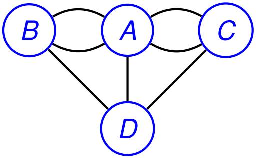
$V$ - set of vertices.

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

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

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

# Graphs: formally.




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

$V$ - set of vertices.

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

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

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

For CS 70, usually simple graphs.

# Graphs: formally.


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

$V$ - set of vertices.

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

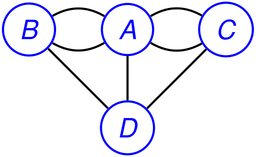
$E \subseteq V \times V$ - set of edges.

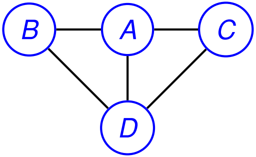
$\{\{A,B\},\{A,B\},\{A,C\},\{B,C\},\{B,D\},\{B,D\},\{C,D\}\}$.

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.

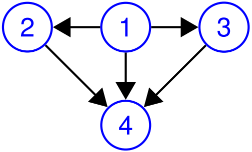
$\{\{A, B\}, \{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}, \{B, D\}, \{C, D\}\}$.

For CS 70, usually simple graphs.

No parallel edges.

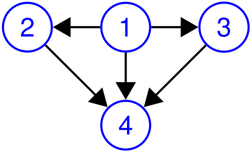
Multigraph above.

# Directed Graphs




$G = (V, E)$.

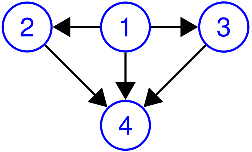
# Directed Graphs




$G = (V, E)$.

$V$ - set of vertices.

# Directed Graphs

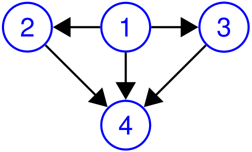



$G = (V, E)$.

$V$ - set of vertices.

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

# Directed Graphs



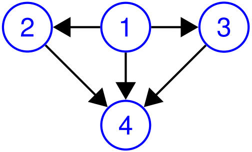

$G = (V, E)$.

$V$ - set of vertices.

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

$E$ ordered pairs of vertices.

# Directed Graphs




$G = (V, E)$.

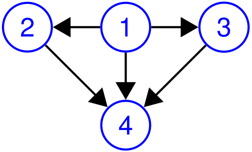
$V$ - set of vertices.

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

$E$ ordered pairs of vertices.

$\{(1, 2),$

# Directed Graphs




$G = (V, E)$.

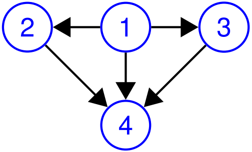
$V$ - set of vertices.

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

$E$ ordered pairs of vertices.

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

# Directed Graphs




$G = (V, E)$.

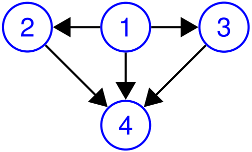
$V$ - set of vertices.

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

$E$ ordered pairs of vertices.

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

# Directed Graphs




$G = (V, E)$.

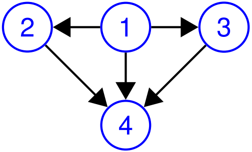
$V$ - set of vertices.

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

$E$ ordered pairs of vertices.

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

# Directed Graphs




$G = (V, E)$.

$V$ - set of vertices.

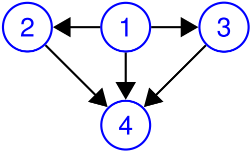
$\{1, 2, 3, 4\}$

$E$ ordered pairs of vertices.

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

One way streets.

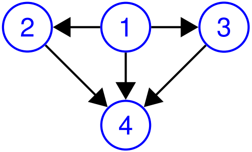
# Directed Graphs




$G = (V, E)$.
$V$ - 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.

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

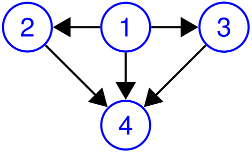
$E$ ordered pairs of vertices.

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

One way streets.

Tournament: 1 beats 2,

# Directed Graphs




$G = (V, E)$.

$V$ - set of vertices.

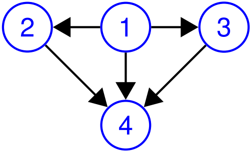
$\{1, 2, 3, 4\}$

$E$ 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.

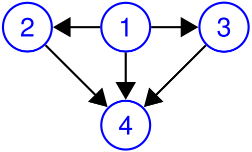
$\{1, 2, 3, 4\}$

$E$ 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.

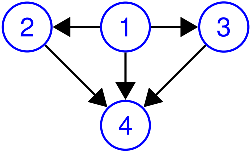
$\{1, 2, 3, 4\}$

$E$ 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.

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

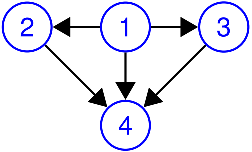
$E$ ordered pairs of vertices.

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

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

Social Network:

# Directed Graphs




$G = (V, E)$.

$V$ - set of vertices.

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

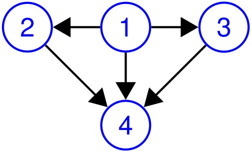
$E$ ordered pairs of vertices.

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

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

Social Network: Directed?

# Directed Graphs




$G = (V, E)$.

$V$ - set of vertices.

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

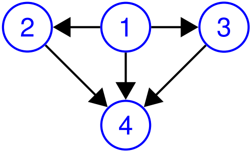
$E$ ordered pairs of vertices.

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

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.

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

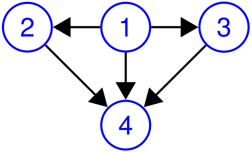
$E$ ordered pairs of vertices.

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

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

Social Network: Directed? Undirected?
Friends.

# Directed Graphs

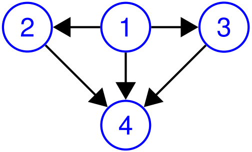



$G = (V, E)$.
$V$ - set of vertices.
$\{1, 2, 3, 4\}$
$E$ ordered pairs of vertices.
$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

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

Social Network: Directed? Undirected?
Friends. Undirected.

# Directed Graphs

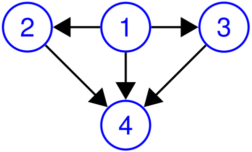



$G = (V, E)$.
$V$ - set of vertices.
$\{1, 2, 3, 4\}$
$E$ ordered pairs of vertices.
$\{(1,2),(1,3),(1,4),(2,4),(3,4)\}$

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

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

# Directed Graphs

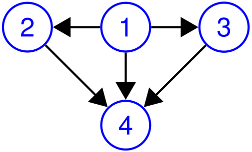



$G = (V, E)$.
$V$ - set of vertices.
$\{1, 2, 3, 4\}$
$E$ ordered pairs of vertices.
$\{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\}$

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: $G = (V, E)$

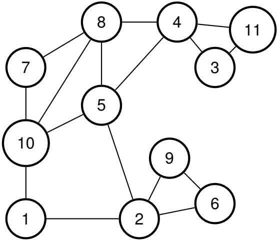
# Graph Concepts and Definitions.

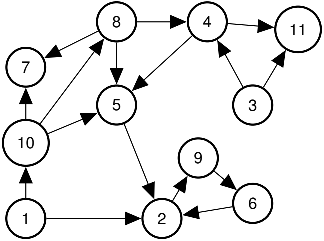
Graph: $G = (V, E)$
neighbors, adjacent, degree, incident, in-degree, out-degree

# Graph Concepts and Definitions.

Graph: $G = (V, E)$

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



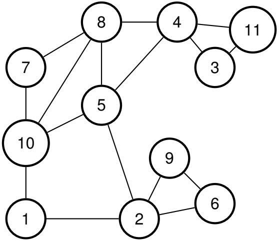



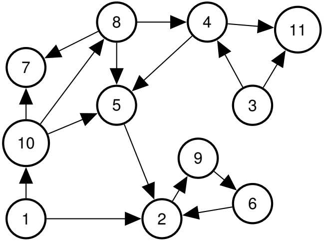

Neighbors of 10?

# Graph Concepts and Definitions.

Graph: $G = (V, E)$

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



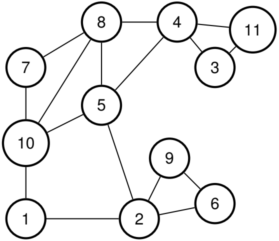



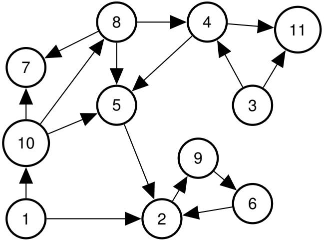

Neighbors of 10? 1,

# Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

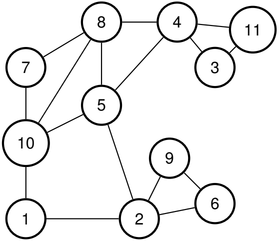


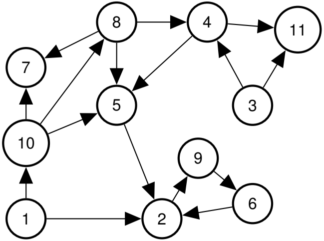




Neighbors of 10? 1,5,

# Graph Concepts and Definitions.

Graph: $G = (V, E)$
neighbors, adjacent, degree, incident, in-degree, out-degree



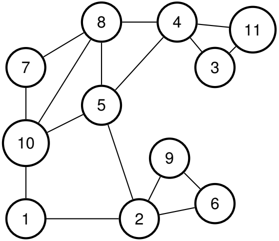



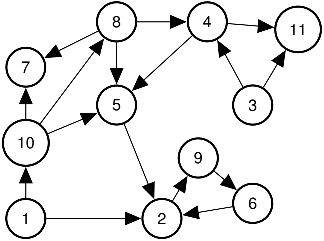

Neighbors of 10? 1,5,7,

# Graph Concepts and Definitions.

Graph: $G = (V, E)$

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



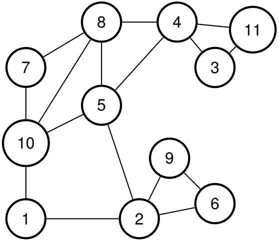



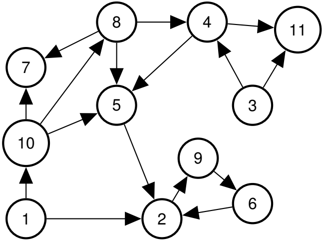

Neighbors of 10? 1,5,7, 8.

# Graph Concepts and Definitions.

Graph: $G = (V, E)$

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



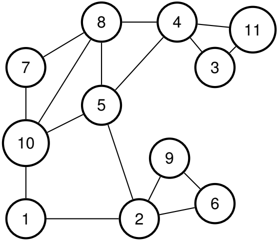

Neighbors of 10? 1,5,7, 8.

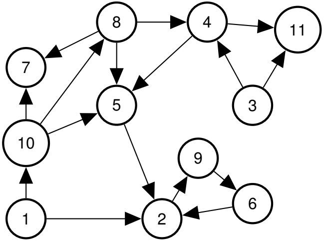
$u$ is neighbor of $v$ if $(u, v) \in E$.

# Graph Concepts and Definitions.

Graph: $G = (V, E)$

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



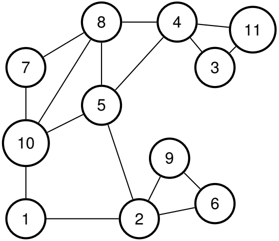

Neighbors of 10? 1,5,7, 8.

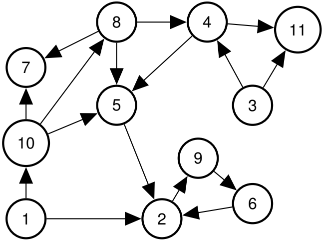
$u$ is neighbor of $v$ if $(u, v) \in E$.

Edge $(10, 5)$ is incident to

# Graph Concepts and Definitions.

Graph: $G = (V, E)$
neighbors, adjacent, degree, incident, in-degree, out-degree

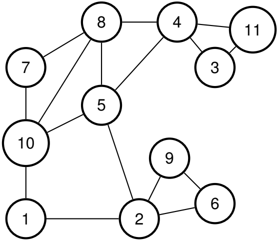


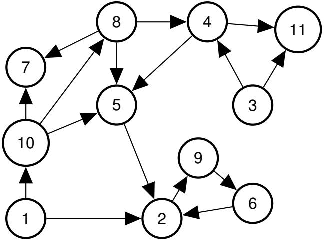




Neighbors of 10? 1,5,7, 8.
$u$ is neighbor of $v$ if $(u, v) \in E$.
Edge $(10, 5)$ is incident to vertex 10 and vertex 5.
Edge $(u, v)$ is incident to $u$ and $v$.
Degree of vertex 1?

# Graph Concepts and Definitions.

Graph: $G = (V, E)$
neighbors, adjacent, degree, incident, in-degree, out-degree



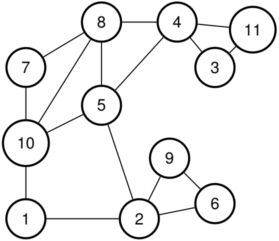



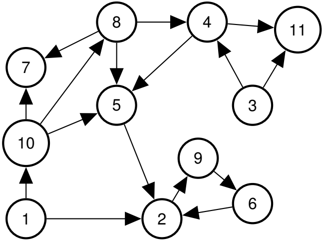

Neighbors of 10? 1,5,7, 8.
$u$ is neighbor of $v$ if $(u, v) \in E$.
Edge $(10, 5)$ is incident to vertex 10 and vertex 5.
Edge $(u, v)$ is incident to $u$ and $v$.
Degree of vertex 1? 2

# 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 $(u, v) \in E$.

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

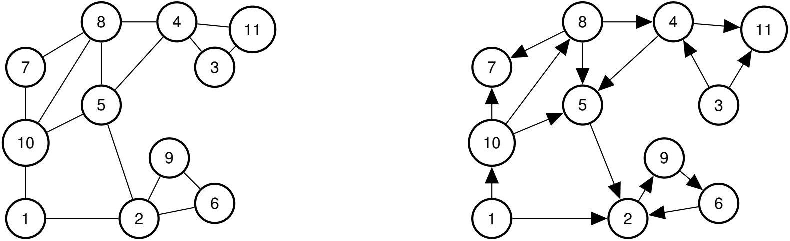
Edge $(u, v)$ is incident to $u$ and $v$.

Degree of vertex 1? 2

Degree of vertex $u$ is number of incident edges.

# Graph Concepts and Definitions.

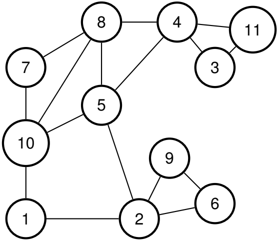
Graph: $G = (V, E)$
neighbors, adjacent, degree, incident, in-degree, out-degree

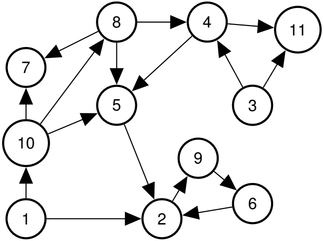



Neighbors of 10? 1,5,7, 8.
$u$ is neighbor of $v$ if $(u, v) \in E$.
Edge $(10, 5)$ is incident to vertex 10 and vertex 5.
Edge $(u, v)$ is incident to $u$ and $v$.
Degree of vertex 1? 2
Degree of vertex $u$ is number of incident edges.
Equals number of neighbors in simple graph.

# Graph Concepts and Definitions.

Graph: $G = (V, E)$
neighbors, adjacent, degree, incident, in-degree, out-degree



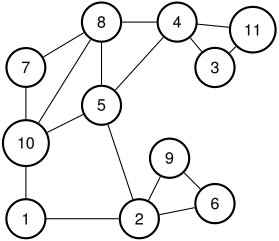



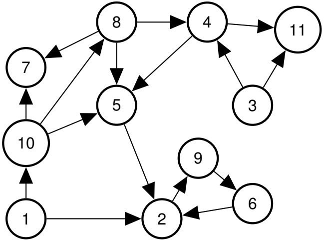

Neighbors of 10? 1,5,7, 8.
$u$ is neighbor of $v$ if $(u,v) \in E$.
Edge $(10,5)$ is incident to vertex 10 and vertex 5.
Edge $(u,v)$ is incident to $u$ and $v$.
Degree of vertex 1? 2
Degree of vertex $u$ is number of incident edges.
Equals number of neighbors in simple graph.
**Directed graph:**

# 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 $(u, v) \in E$.

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.

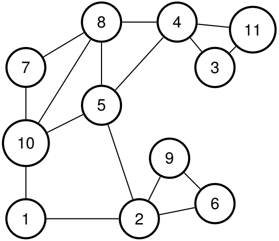
Equals number of neighbors in simple graph.

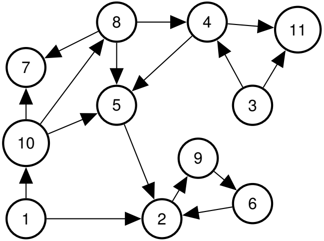
**Directed graph:** In-degree of 10?

# 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 $(u, v) \in E$.

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.

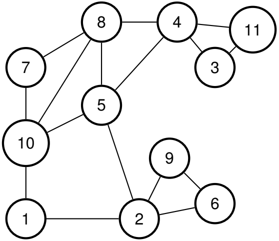
Equals number of neighbors in simple graph.

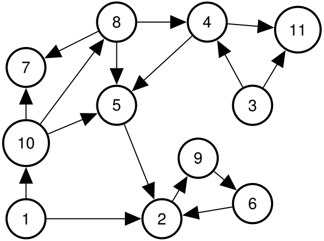
**Directed graph:** In-degree of 10? 1

# 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 $(u, v) \in E$.

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.

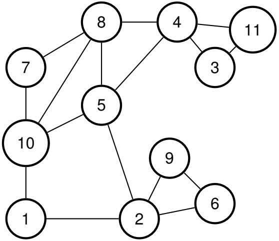
Equals number of neighbors in simple graph.

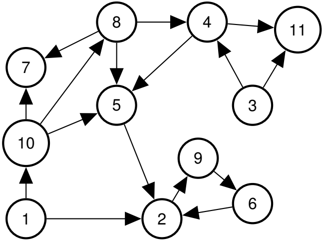
**Directed graph:** In-degree of 10? 1    Out-degree of 10?

# 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 $(u, v) \in E$.

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.

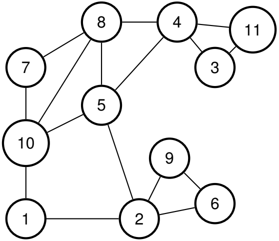
Equals number of neighbors in simple graph.

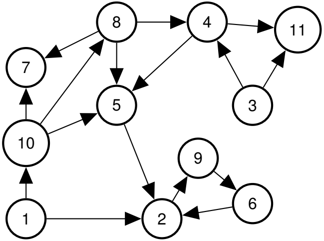
**Directed graph:** In-degree of 10? 1    Out-degree of 10? 3

# 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 $(u, v) \in E$.

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) What?

# 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) What?

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) What?

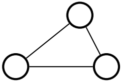
Not (A)! Triangle.

# 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) What?

Not (A)! Triangle.



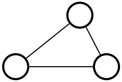
Not (B)!

# 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) What?

Not (A)! Triangle.



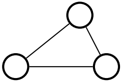
Not (B)! Triangle.

# 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) What?

Not (A)! Triangle.



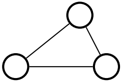
Not (B)! Triangle.

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

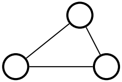
What?

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

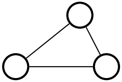
What? For triangle number of edges is 3, the sum of degrees is 6.

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

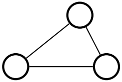
Could it always be...

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

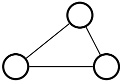
Could it always be...$2|E|$? ..

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

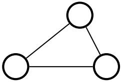
Could it always be...$2|E|$? ..or $2|V|$?

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

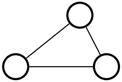
How many incidences does each edge contribute?

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

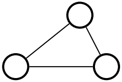
How many incidences does each edge contribute? 2.

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

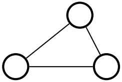
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

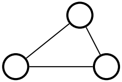
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree $v$?

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

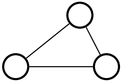
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree $v$? incidences contributed to $v$!

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

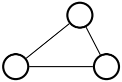
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree $v$? incidences contributed to $v$!
sum of degrees is total incidences

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

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

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

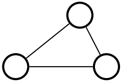
How many incidences does each edge contribute? 2.
$2|E|$ incidences are contributed in total!
What is degree $v$? incidences contributed to $v$!
sum of degrees is total incidences ... or $2|E|$.

# 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) What?

Not (A)! Triangle.



Not (B)! Triangle.

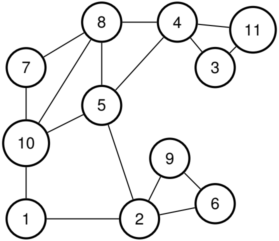
What? For triangle number of edges is 3, the sum of degrees is 6.

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

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 $2|E|$.

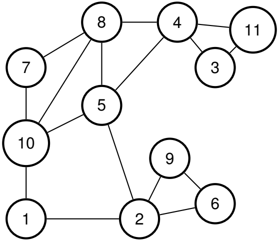
**Thm:** Sum of vertex degress is $2|E|$.

# Paths, walks, cycles, tour.




A path in a graph is a sequence of edges.

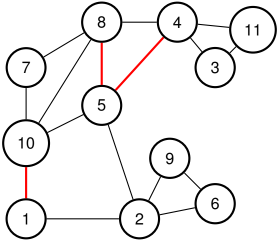
# Paths, walks, cycles, tour.




A path in a graph is a sequence of edges.

Path?

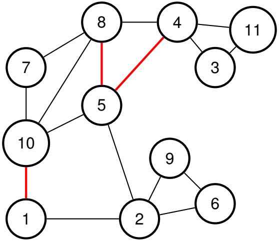
# Paths, walks, cycles, tour.




A path in a graph is a sequence of edges.

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

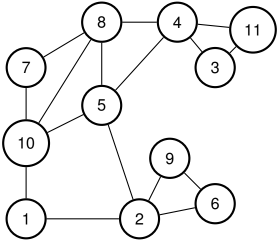
# Paths, walks, cycles, tour.




A path in a graph is a sequence of edges.

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

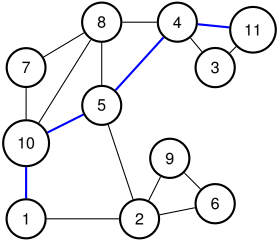
# Paths, walks, cycles, tour.




A path in a graph is a sequence of edges.

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

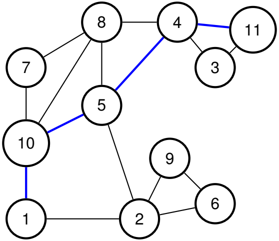
# Paths, walks, cycles, tour.




A path in a graph is a sequence of edges.

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

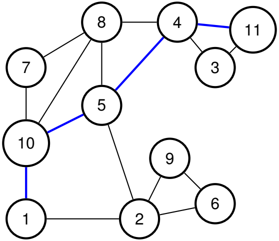
# Paths, walks, cycles, tour.




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.

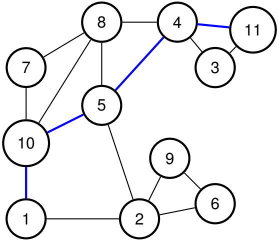



A path in a graph is a sequence of edges.

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.



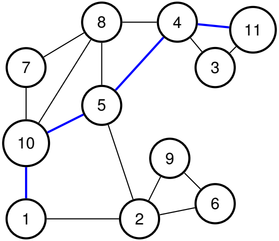

A path in a graph is a sequence of edges.

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.

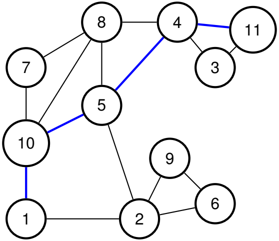



A path in a graph is a sequence of edges.

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.

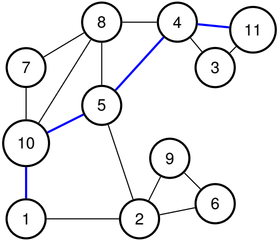
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$ ? No!

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.



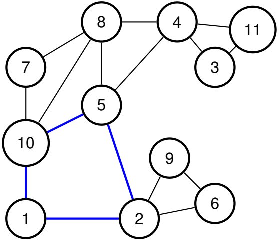

A path in a graph is a sequence of edges.

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.

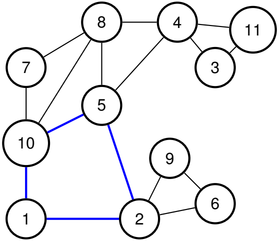
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$ ? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

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.

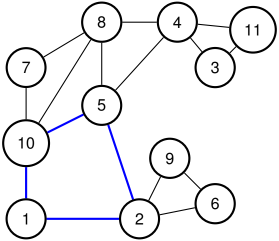
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$ ? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

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.

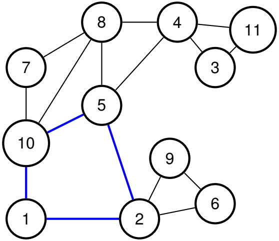
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$ ? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

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.



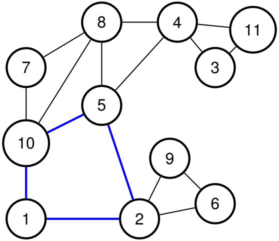

A path in a graph is a sequence of edges.

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.



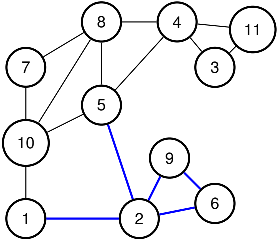

A path in a graph is a sequence of edges.

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.

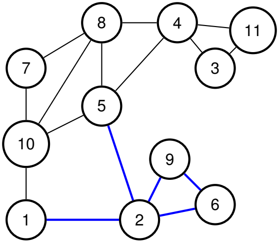
Path? $\{1,10\}$, $\{8,5\}$, $\{4,5\}$ ? No!
Path? $\{1,10\}$, $\{10,5\}$, $\{5,4\}$, $\{4,11\}$? Yes!

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

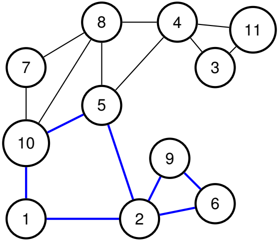
Quick Check! Length of path? $k$ vertices or $k-1$ edges.

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!

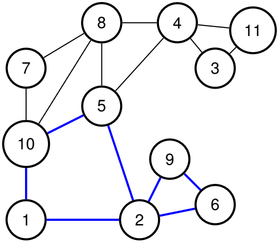
Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Quick Check! Length of path? $k$ vertices or $k-1$ edges.

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.

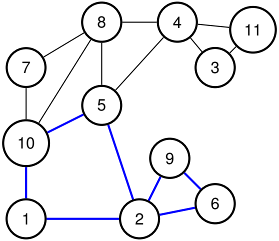
Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!

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!

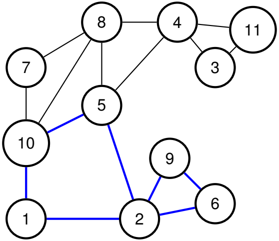
Path is usually simple. No repeated vertex!

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.

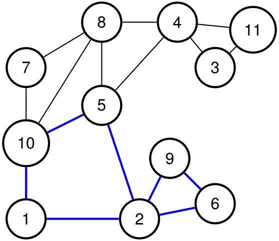
Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually simple. No repeated vertex!

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.

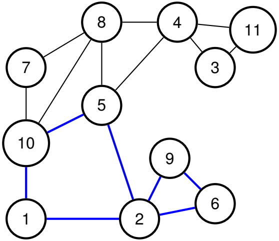
Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually simple. No repeated vertex!

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.

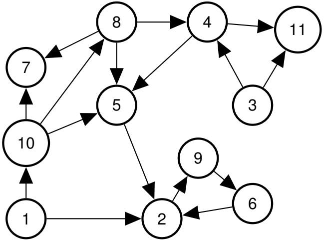
Cycle: Path with $v_1 = v_k$. Length of cycle? $k-1$ vertices and edges!
Path is usually simple. No repeated vertex!

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

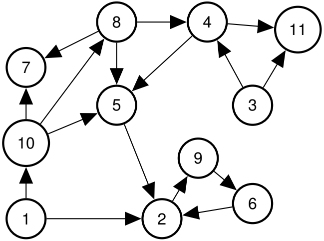
Tour is walk that starts and ends at the same node.

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

# Directed Paths.

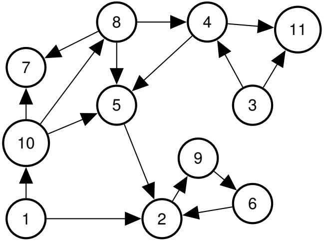


# Directed Paths.



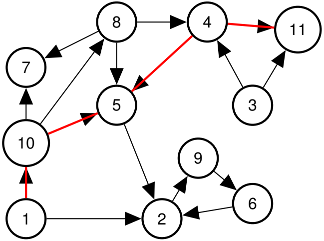

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

# Directed Paths.



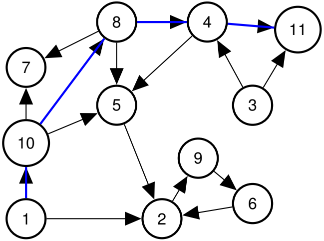

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

# Directed Paths.



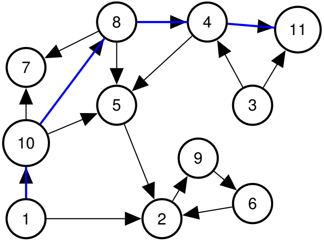

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

# Directed Paths.



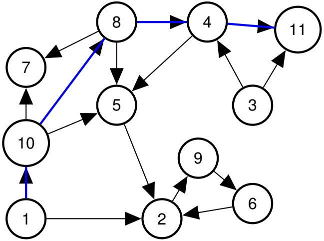

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

# Directed Paths.



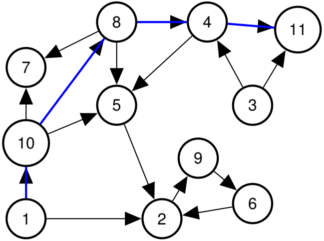

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Paths,

# Directed Paths.



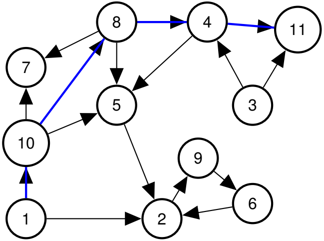

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Paths, walks,

# Directed Paths.




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

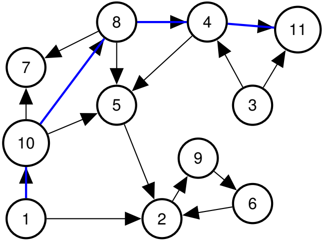
# Directed Paths.




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

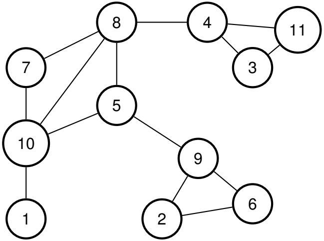
Paths, walks, cycles, tours

# Directed Paths.



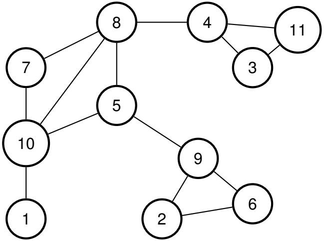

Path: $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.
Paths, walks, cycles, tours ... are analagous to undirected now.

# Connectivity




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

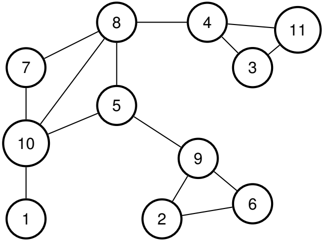
# Connectivity




$u$ 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

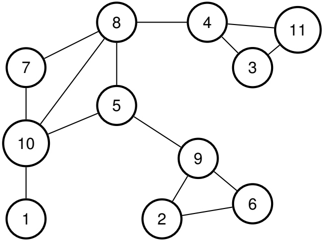



*u* and *v* are connected if there is a path between *u* and *v*.

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

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

# Connectivity

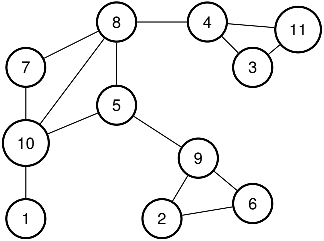



$u$ and $v$ are connected if there is a path between $u$ and $v$.

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

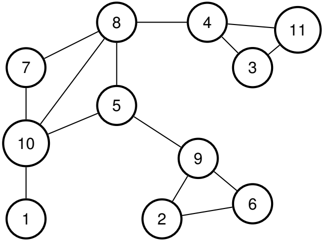



*u* and *v* are connected if there is a path between *u* and *v*.

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

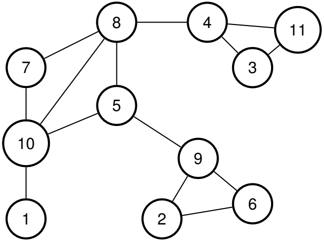



$u$ and $v$ are connected if there is a path between $u$ and $v$.

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



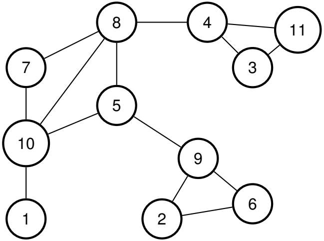

$u$ and $v$ are connected if there is a path between $u$ and $v$.

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



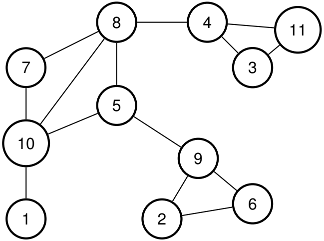

$u$ and $v$ are connected if there is a path between $u$ and $v$.

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



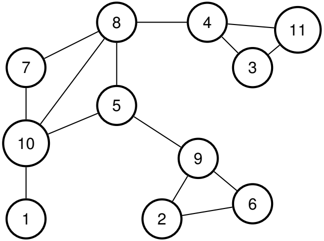

$u$ and $v$ are connected if there is a path between $u$ and $v$.

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 $v$.

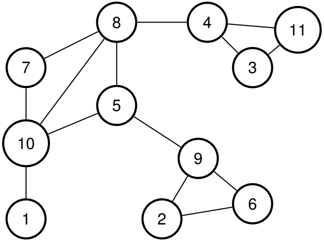
A connected graph is a graph where all pairs of vertices are connected.

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 $v$. $\square$

May not be simple!

# Connectivity




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

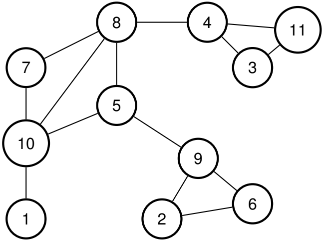
A connected graph is a graph where all pairs of vertices are connected.

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 $v$. $\square$

May not be simple!
Either modify definition to walk.

# Connectivity




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

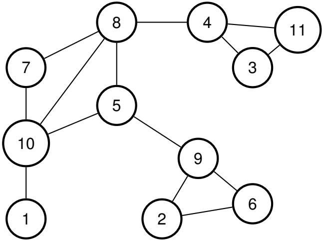
A connected graph is a graph where all pairs of vertices are connected.

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 $v$. $\square$

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 $v$.

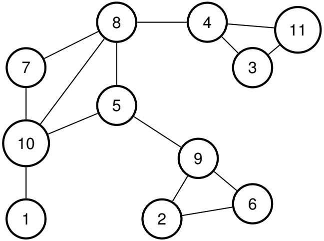
A connected graph is a graph where all pairs of vertices are connected.

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 $v$. $\square$

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 $v$.

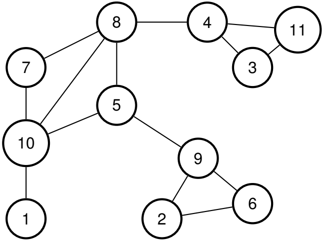
A connected graph is a graph where all pairs of vertices are connected.

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 $v$. $\square$

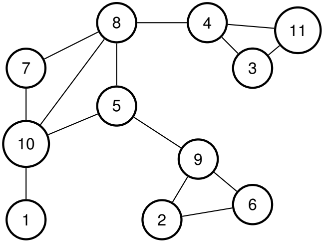
May not be simple!
Either modify definition to walk.
Or cut out cycles. .

# Understanding Definition.



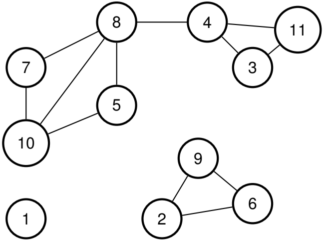

Is graph above connected?

# Understanding Definition.




Is graph above connected? Yes!

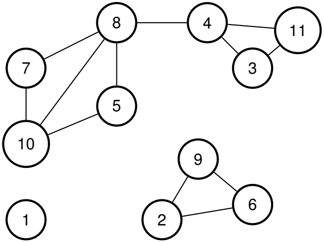
# Understanding Definition.




Is graph above connected? Yes!

How about now?

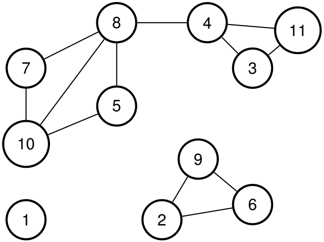
# Understanding Definition.




Is graph above connected? Yes!

How about now? No!

# Understanding Definition.

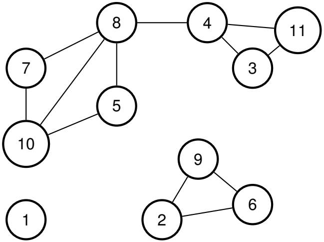



Is graph above connected? Yes!

How about now? No!

Connected Components?

# Understanding Definition.

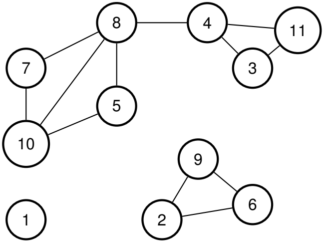



Is graph above connected? Yes!

How about now? No!

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

# Understanding Definition.

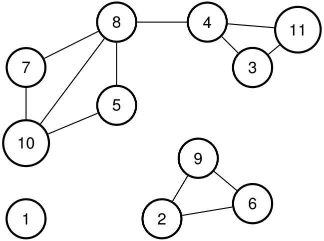



Is graph above connected? Yes!

How about now? No!

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

# Understanding Definition.



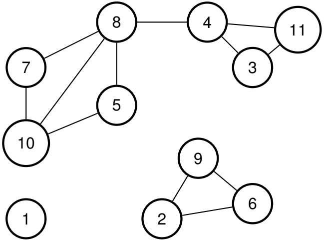

Is graph above connected? Yes!

How about now? No!

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

Connected component - maximal set of connected vertices.

# Understanding Definition.



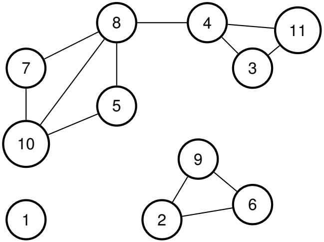

Is graph above connected? Yes!

How about now? No!

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

Connected component - maximal set of connected vertices.

# Understanding Definition.




Is graph above connected? Yes!

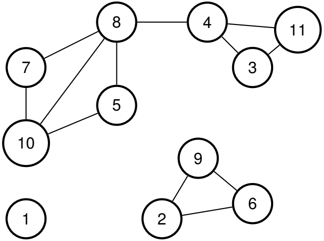
How about now? No!

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

Connected component - maximal set of connected vertices.

Quick Check: Is $\{10, 7, 5\}$ a connected component?

# Understanding Definition.




Is graph above connected? Yes!

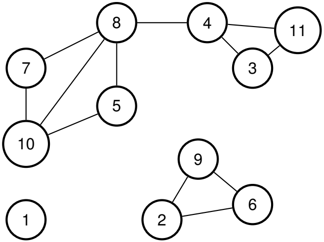
How about now? No!

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

Connected component - maximal set of connected vertices.

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

# Understanding Definition.




Is graph above connected? Yes!

How about now? No!

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

Connected component - maximal set of connected vertices.

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

Not maximal.

# 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.

# 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.

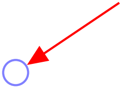
**Proof of only if: Eulerian $\implies$ connected and all even degree.**
Eulerian Tour is connected so graph is connected.
Tour enters and leaves vertex $v$ on each visit.
Uses two incident edges per visit. Tour uses all incident edges.
Therefore $v$ has even degree. $\square$

# 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 $\implies$ 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. $\square$



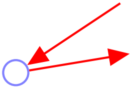
When you enter,

# 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. $\square$



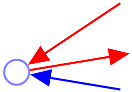
When you enter, you leave.

# 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. $\square$



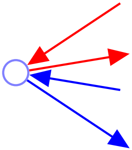
When you enter, you leave.

# 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. $\square$



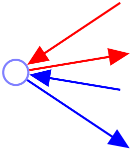
When you enter, you leave.

# 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. $\square$



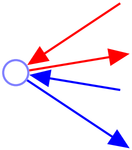
When you enter, you leave.
For starting node,

# 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. $\square$



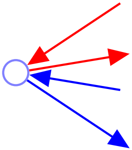
When you enter, you leave.
For starting node, tour leaves first

# 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. $\square$



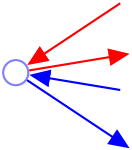
When you enter, you leave.
For starting node, tour leaves first ....then enters at end.

# 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 $\Longrightarrow$ 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.

# Finding a tour!

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

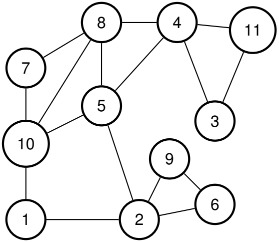
# Finding a tour!

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

# Finding a tour!

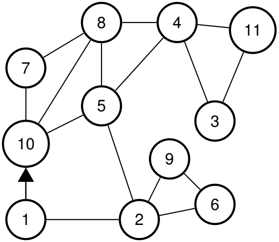
**Proof of if: Even + connected $\Longrightarrow$ Eulerian Tour.**
We will give an algorithm. First by picture.

1. Take a walk starting from $v$ (1) on "unused" edges



# Finding a tour!

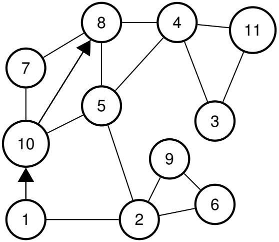
**Proof of if: Even + connected $\Longrightarrow$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges

# Finding a tour!

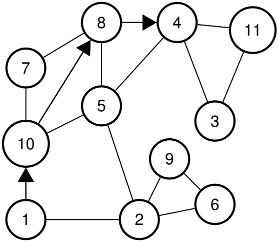
**Proof of if: Even + connected $\Longrightarrow$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges

# Finding a tour!

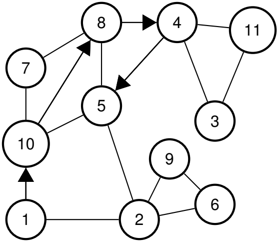
**Proof of if: Even + connected $\Longrightarrow$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges

# Finding a tour!

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

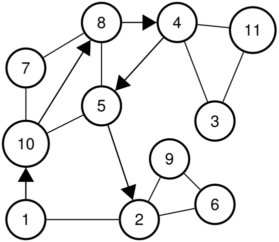



1. Take a walk starting from $v$ (1) on "unused" edges

# Finding a tour!

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

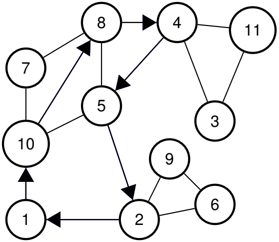
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges

# Finding a tour!

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

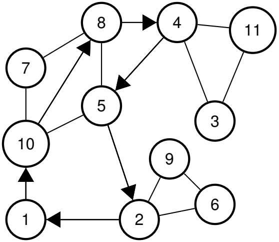



1. Take a walk starting from $v$ (1) on "unused" edges
... till you get back to $v$.

# Finding a tour!

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

We will give an algorithm. First by picture.

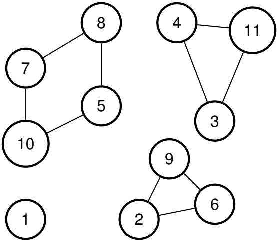



1. Take a walk starting from $v$ (1) on “unused” edges
... till you get back to $v$.
2. Remove tour, $C$.

# Finding a tour!

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

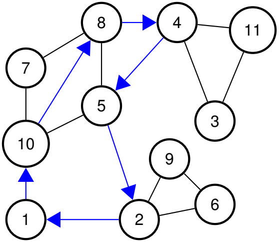
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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 $\Longrightarrow$ Eulerian Tour.**
We will give an algorithm. First by picture.

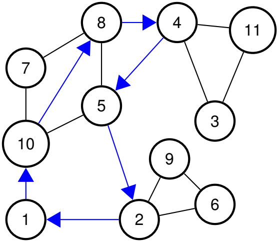



1. Take a walk starting from $v$ (1) on "unused" edges
... till you get back to $v$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by $C$.

# 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) on "unused" edges

... till you get back to $v$.

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

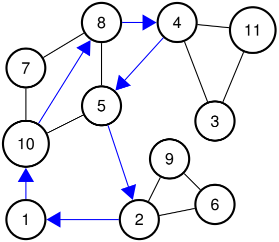
   Each is touched by $C$.

   Why?

# Finding a tour!

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

We will give an algorithm. First by picture.

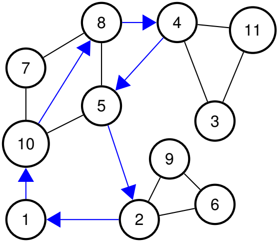



1. Take a walk starting from $v$ (1) on "unused" edges
... till you get back to $v$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by $C$.
      Why? $G$ was connected.

# 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) on "unused" edges

... till you get back to $v$.

2. Remove tour, $C$.

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

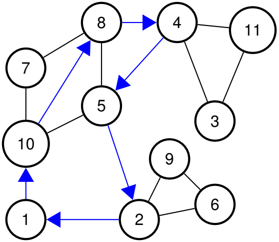
   Each is touched by $C$.

   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.

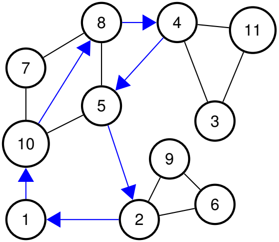



1. Take a walk starting from $v$ (1) on "unused" edges
... till you get back to $v$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components.
Each is touched by $C$.
Why? $G$ was connected.
Let $v_i$ be (first) node in $G_i$ touched by $C$.
Example: $v_1 = 1$,

# Finding a tour!

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

We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges

... 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.

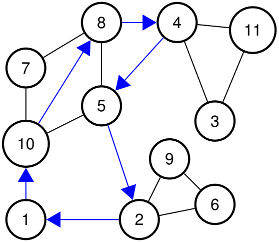
   Let $v_i$ be (first) node in $G_i$ touched by $C$.

   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) on "unused" edges

... till you get back to $v$.

2. Remove tour, $C$.

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

   Each is touched by $C$.

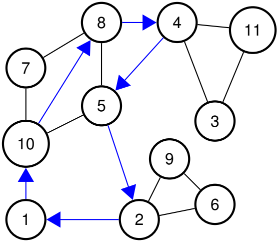
   Why? $G$ was connected.

   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!

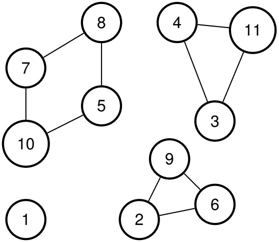
**Proof of if: Even + connected $\implies$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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!

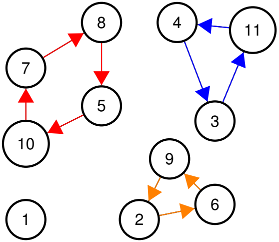
**Proof of if: Even + connected $\implies$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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.

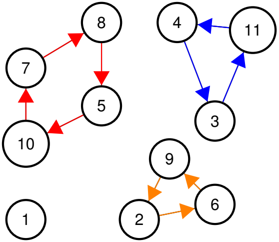



1. Take a walk starting from $v$ (1) on "unused" edges
... till you get back to $v$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by $C$.
      Why? $G$ was connected.
   Let $v_i$ be (first) node in $G_i$ touched by $C$.
   Example: $v_1 = 1$, $v_2 = 10$,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

... till you get back to $v$.

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

   Each is touched by $C$.

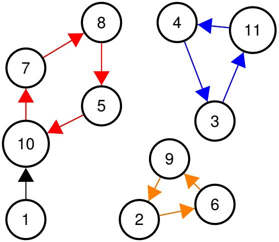
   Why? $G$ was connected.

   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!

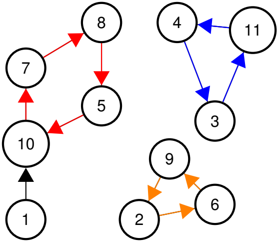
**Proof of if: Even + connected $\implies$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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!

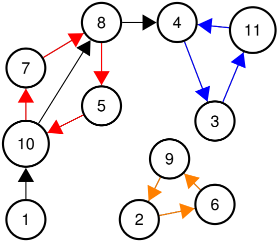
**Proof of if: Even + connected $\implies$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on “unused” edges
... 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.

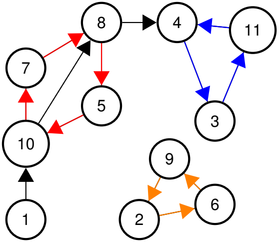



1. Take a walk starting from $v$ (1) on "unused" edges
... till you get back to $v$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by $C$.
   Why? $G$ was connected.
   Let $v_i$ be (first) node in $G_i$ touched by $C$.
   Example: $v_1 = 1$, $v_2 = 10$, $v_3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

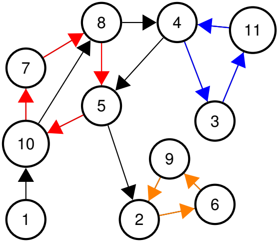
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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!

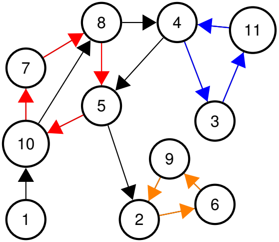
**Proof of if: Even + connected $\implies$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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!

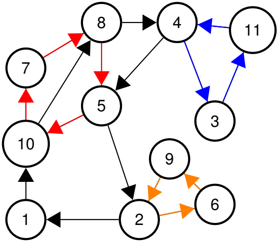
**Proof of if: Even + connected $\implies$ Eulerian Tour.**
We will give an algorithm. First by picture.




1. Take a walk starting from $v$ (1) on "unused" edges
... 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) on "unused" edges
... 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!

# Recursive/Inductive Algorithm.

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

# Recursive/Inductive Algorithm.

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

**Claim:** Do get back to $v$!

# Recursive/Inductive Algorithm.

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.

# Recursive/Inductive Algorithm.

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

# Recursive/Inductive Algorithm.

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 $v$.

# Recursive/Inductive Algorithm.

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 $v$. $\square$

# Recursive/Inductive Algorithm.

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 $G$.

# Recursive/Inductive Algorithm.

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!)

# Recursive/Inductive Algorithm.

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 $G_1, \ldots, G_k$.

# Recursive/Inductive Algorithm.

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 $G_i$.

# Recursive/Inductive Algorithm.

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 $C$?

# Recursive/Inductive Algorithm.

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 $\implies$

# Recursive/Inductive Algorithm.

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 $C$.

# Recursive/Inductive Algorithm.

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 $C$.

# Recursive/Inductive Algorithm.

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**

# Recursive/Inductive Algorithm.

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.**

# Recursive/Inductive Algorithm.

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 $v$.

# Recursive/Inductive Algorithm.

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 $v$. $\square$

# Recursive/Inductive Algorithm.

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 $G_i$

# Recursive/Inductive Algorithm.

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 $v_i$.

# Recursive/Inductive Algorithm.

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 $G_i$ starting/ending at $v_i$. Induction.

# Recursive/Inductive Algorithm.

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 $G_i$ starting/ending at $v_i$. Induction.
4. Splice $T_i$ into $C$ where $v_i$ first appears in $C$.

# Recursive/Inductive Algorithm.

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 $G_i$ starting/ending at $v_i$. Induction.
4. Splice $T_i$ into $C$ where $v_i$ first appears in $C$.

Visits every edge once:
Visits edges in $C$

# Recursive/Inductive Algorithm.

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 $G_i$ starting/ending at $v_i$. Induction.
4. Splice $T_i$ into $C$ where $v_i$ first appears in $C$.

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

# Recursive/Inductive Algorithm.

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 $v_i$. Induction.
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 edges in each $G_i$.

# Recursive/Inductive Algorithm.

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 $G_i$ starting/ending at $v_i$. Induction.
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 edges in each $G_i$. $\square$

# Administration Time!

Well admin time!

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

Test Option: don't have to do homework.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

Test Option: don't have to do homework. Yes!!

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth:

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
    - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth: mostly test,

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth: mostly test, both options!

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth: mostly test, both options!

- Variance mostly in exams for A/B range.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth: mostly test, both options!

- Variance mostly in exams for A/B range.
  - most homework students get near perfect scores on homework.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth: mostly test, both options!

- Variance mostly in exams for A/B range.
  - most homework students get near perfect scores on homework.

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

Test Option: don't have to do homework. Yes!!

Should do homework. No need to write up.

Homework Option: have to do homework. Bummer!

The truth: mostly test, both options!

Variance mostly in exams for A/B range.

most homework students get near perfect scores on homework.

How will I do?

# Administration Time!

Well admin time!

Must choose homework option or test only: soon after recieving hw 1 scores.

- Test Option: don't have to do homework. Yes!!
  - Should do homework. No need to write up.
- Homework Option: have to do homework. Bummer!

The truth: mostly test, both options!

- Variance mostly in exams for A/B range.
  - most homework students get near perfect scores on homework.

How will I do?

Mostly up to you.

# A few details.

Welcome to turn in homework in test-only option.

# A few details.

Welcome to turn in homework in test-only option.

Changes:

# A few details.

Welcome to turn in homework in test-only option.

Changes:
80% max on homework.

# A few details.

Welcome to turn in homework in test-only option.

Changes:
  80% max on homework.
  Test Only Reflection.

# A few details.

Welcome to turn in homework in test-only option.

Changes:
  80% max on homework.
  Test Only Reflection.

Professor Rao.

# A few details.

Welcome to turn in homework in test-only option.

Changes:

80% max on homework.

Test Only Reflection.

Professor Rao. What should I do?

# A few details.

Welcome to turn in homework in test-only option.

Changes:
  80% max on homework.
  Test Only Reflection.

Professor Rao. What should I do?

Whatever you think best for you.

# TA/Professor Ramchandran opinion

# TA/Professor Ramchandran opinion

Do your homework!

# TA/Professor Ramchandran opinion

Do your homework! Darnit.

# TA/Professor Ramchandran opinion

Do your homework! Darnit.

Make them do their homework!

# TA/Professor Ramchandran opinion

Do your homework! Darnit.

Make them do their homework! It is good for them.

# TA/Professor Ramchandran opinion

Do your homework! Darnit.

- Make them do their homework! It is good for them.
- Even if it makes little difference on test performance.

# TA/Professor Ramchandran opinion

Do your homework! Darnit.
- Make them do their homework! It is good for them.
- Even if it makes little difference on test performance.
  - E.g. One test prep method: grind through old exams.

# TA/Professor Ramchandran opinion

- Do your homework! Darnit.
  - Make them do their homework! It is good for them.
  - Even if it makes little difference on test performance.
    - E.g. One test prep method: grind through old exams.
  - You still learn.

# TA/Professor Ramchandran opinion

- Do your homework! Darnit.
  - Make them do their homework! It is good for them.
  - Even if it makes little difference on test performance.
    - E.g. One test prep method: grind through old exams.
  - You still learn.
  - Tests not necessarily a measure of learning.

# TA/Professor Ramchandran opinion

- Do your homework! Darnit.
  - Make them do their homework! It is good for them.
  - Even if it makes little difference on test performance.
    - E.g. One test prep method: grind through old exams.
  - You still learn.
  - Tests not necessarily a measure of learning.
  - Indispensable learning tool: go toe-to-toe with problems!

# TA/Professor Ramchandran opinion

- Do your homework! Darnit.
  - Make them do their homework! It is good for them.
  - Even if it makes little difference on test performance.
    - E.g. One test prep method: grind through old exams.
  - You still learn.
  - Tests not necessarily a measure of learning.
  - Indispensable learning tool: go toe-to-toe with problems!

# TA/Professor Ramchandran opinion

Do your homework! Darnit.

- Make them do their homework! It is good for them.
- Even if it makes little difference on test performance.
  - E.g. One test prep method: grind through old exams.
- You still learn.
- Tests not necessarily a measure of learning.
- Indispensable learning tool: go toe-to-toe with problems!

HWs force learning depth and discipline....

# TA/Professor Ramchandran opinion

Do your homework! Darnit.
- Make them do their homework! It is good for them.
- Even if it makes little difference on test performance.
  - E.g. One test prep method: grind through old exams.
- You still learn.
- Tests not necessarily a measure of learning.
- Indispensable learning tool: go toe-to-toe with problems!

HWs force learning depth and discipline....
- like a gourmet meal vs. fast-food

# TA/Professor Ramchandran opinion

Do your homework! Darnit.

- Make them do their homework! It is good for them.
- Even if it makes little difference on test performance.
  - E.g. One test prep method: grind through old exams.
- You still learn.
- Tests not necessarily a measure of learning.
- Indispensable learning tool: go toe-to-toe with problems!

HWs force learning depth and discipline....

- like a gourmet meal vs. fast-food (cramming for test-only)....