Lecture 6: Graphs.

Graphs!
Graphs!
Euler
Graphs!
Euler
Definitions: model.
Lecture 6: Graphs.

Graphs!
Euler
Definitions: model.
Euler Again!!
Graphs!
Euler
Definitions: model.
Euler Again!!
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giușcă - License.
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giusca - License.
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

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 Giuscă - 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?

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 Giusăcă - License.

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

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

$V$ - set of vertices.

\{A, B, C, D\}

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

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

For CS 70, usually simple graphs.

No parallel edges.

Multigraph above.
Graphs: formally.

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

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

$E = \{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$. For CS 70, usually simple graphs. No parallel edges. Multigraph above.
Graphs: formally.

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

- $V$ - set of vertices.
- $E$ - set of edges.

For CS 70, usually simple graphs. No parallel edges. Multigraph above.
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.

For CS 70, usually simple graphs.
No parallel edges.
Multigraph above.
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.
  - $\{\{A, B\}, \{A, B\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$

For CS 70, usually simple graphs.
- No parallel edges.
- Multigraph above.
Graphs: formally.

Graph: $G = (V, E)$.
- $V$ - set of vertices.
  - $\{A, B, C, D\}$
- $E \subseteq V \times V$ - set of edges.
  - $\{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$. 
Graphs: formally.

Graph: $G = (V, E)$.
- $V$ - set of vertices.
  - $\{A, B, C, D\}$
- $E \subseteq V \times V$ - set of edges.
  - $\{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$.

For CS 70, usually simple graphs.
Graphs: formally.

Graph: $G = (V, E)$.
- $V$ - set of vertices.
  - $\{A, B, C, D\}$
- $E \subseteq V \times V$ - set of edges.
  - $\{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\}$.

For CS 70, usually simple graphs.
No parallel edges.
Graph: $G = (V, E)$.

$V$ - set of vertices.

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

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

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

For CS 70, usually simple graphs.

No parallel edges.

Multigraph above.
Directed Graphs

$G = (V, E)$.

- $V$: 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.
Directed Graphs

\( G = (V, E) \).

- \( V \) - set of vertices.
  \( \{1, 2, 3, 4\} \)

One way streets.

Tournament:
- 1 beats 2, ...

Precedence:
- 1 is before 2, ..

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

\[ G = (V, E). \]
\[ V - \text{set of vertices.} \]
\[ \{1, 2, 3, 4\} \]
\[ E \text{ ordered pairs of vertices.} \]
Directed Graphs

\[ G = (V, E). \]

\( V \) - set of vertices.
\[ \{1, 2, 3, 4\} \]

\( E \) ordered pairs of vertices.
\[ \{(1, 2), \} \]

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.  
$\{(1, 2), (1, 3), \ldots\}$
Directed Graphs

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

One way streets.
Tournament:
- 1 beats 2,
- ...
Precedence:
- 1 is before 2,
- ..
Social Network:
- Directed?
- Undirected?
  - Friends. Undirected.
  - Likes. Directed.
Directed Graphs

\[ G = (V, E). \]

\[ V - \text{set of vertices.} \]
\[ \{1, 2, 3, 4\} \]

\[ E - \text{ordered pairs of vertices.} \]
\[ \{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\} \]
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

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

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?
Undirected?
Friends.
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?
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

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

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: \( G = (V, E) \)

Neighbors, adjacent, degree, incident, in-degree, out-degree

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

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

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

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

Degree of vertex 1: 2

Degree of vertex \( u \) is number of incident edges. Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10: 1

Out-degree of 10: 3
Graph Concepts and Definitions.

Graph: \( G = (V, E) \)

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

Neighbors of 10?

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

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

Degree of vertex 1?

Degree of vertex $u$ is number of incident edges.
Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10?

Out-degree of 10?
Graph Concepts and Definitions.

Graph: \( G = (V, E) \)

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

Neighbors of 10? 1,
Graph Concepts and Definitions.

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

Neighbors of 10? 1, 5,
Graph Concepts and Definitions.

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

Neighbors of 10? 1, 5, 7,
Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

Neighbors of 10? 1, 5, 7, 8.
Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

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

$u$ is neighbor of $v$ if $(u, v) \in E$ (or if $(v, u) \in E$).
Graph: $G = (V, E)$

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

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

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

Edge $(10, 5)$ is incident to
Graph Concepts and Definitions.

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

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

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

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

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

Degree of vertex 1?

Degree of vertex $u$ is number of incident edges.
Equals number of neighbors in simple graph.

Directed graph?
Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

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

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

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

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

Degree of vertex 1? 2
Graph Concepts and Definitions.

Graph: \( G = (V, E) \)

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

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

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

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

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

Degree of vertex 1? 2

Degree of vertex \( u \) is number of incident edges.
Graph Concepts and Definitions.

Graph: \( G = (V, E) \)

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

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

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

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

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

Degree of vertex 1? 2

Degree of vertex \( u \) is number of incident edges.

Equals number of neighbors in simple graph.
Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

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

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

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

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

Degree of vertex 1? 2

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

 Equals number of neighbors in simple graph.
Graph Concepts and Definitions.

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

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

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

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

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

Degree of vertex 1? 2

Degree of vertex \( u \) is number of incident edges.

Equals number of neighbors in simple graph.

Directed graph?
Graph Concepts and Definitions.

Graph: $G = (V, E)$

eighbors, adjacent, degree, incident, in-degree, out-degree

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

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

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

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

Degree of vertex 1? 2

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

Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10?
Graph Concepts and Definitions.

Graph: \( G = (V, E) \)

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

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

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

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

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

Degree of vertex 1? 2

Degree of vertex \( u \) is number of incident edges.

Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10? 1
Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

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

Degree of vertex 1? 2
- Degree of vertex $u$ is number of incident edges.
- Equals number of neighbors in simple graph.

Directed graph?
- In-degree of 10? 1
- Out-degree of 10? 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 neighborhood of $v$ if $(u, v) \in E$ (or if $(v, u) \in E$).

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

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

Degree of vertex 1? 2

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

Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10? 1   Out-degree of 10? 3
Graph Concepts and Definitions.

Graph: $G = (V, E)$

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

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

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

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

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

Degree of vertex 1? 2

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

Equals number of neighbors in simple graph.

Directed graph?

In-degree of 10? 1  Out-degree of 10? 3
Quick Proof.

The sum of the vertex degrees is equal to

(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 $2|E|$?

How many incidences does each edge contribute?

2.

$2|E|$ incidences are contributed in total!

What is degree $v$ incidences contributed to $v$!

The sum of degrees is total incidences... or $2|E|$.

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

$2|E|$?

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 $2|E|$. 
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)!

Triangle. What?

For triangle number of edges is 3, the sum of degrees is 6. Could it always be $2|E|$?

How many incidences does each edge contribute? 2.

$2|E|$ incidences are contributed in total!

What is degree $v$ incidences contributed to $v$!

Sum of degrees is total incidences... or $2|E|$.

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

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

How many incidences does each edge contribute? $2$.

$2|E|$ incidences are contributed in total!

What is degree $v$? incidences contributed to $v$!

Sum of degrees is total incidences... or $2|E|$.

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

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

What is degree $v$ incidences contributed to $v$! Sum of degrees is total incidences... or $2|E|$.

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

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.

What? For triangle number of edges is 3, the sum of degrees is 6.
Could it always be $2|E|$?

How many incidences does each edge contribute? 2.
Quick Proof.

The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$.
(B) the total number of edges, $|E|$.
(C) 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... $2|E|$?

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... $2|E|$?

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 $2|E|$?

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... $2|E|$?

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

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

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

What is degree $v$? incidences contributed to $v$!

sum of degrees is total incidences ... or $2|E|$.

Thm: Sum of vertex degrees is $2|E|$. 
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... \( 2|E| \) ?

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.

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!
A path in a graph is a sequence of edges.

Path?

Path?

Path:

Quick Check!

Length of path?

Cycle: Path with \( v_1 = v_k \).

Length of cycle?

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\} ?

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!
A path in a graph is a sequence of edges.

Path? \{1, 10\}, \{8, 5\}, \{4, 5\} ? No!
A path in a graph is a sequence of edges.

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

Path?
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!

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!
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: \((v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\).
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!
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?
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
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 \(v_1 = v_k\).
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!
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  \textit{simple}. 

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!
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.
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.
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 \textit{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!
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 ??
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!
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 \textit{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.

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

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

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

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

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

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

Paths,
Directed Paths.

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

Paths, walks,
Directed Paths.

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

Paths, walks, cycles,
Directed Paths.

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

Paths, walks, cycles, tours
Directed Paths.

Path: \((v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)\).
Paths, walks, cycles, tours ... are analogous to undirected now.
$u$ 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 the graph connected? Yes? No? Proof: Use path from $u$ to $x$ and then from $x$ to $v$. May not be simple! Either modify definition to walk. Or cut out cycles.
$u$ 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.
$u$ 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?
$u$ 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?
$u$ 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:
$u$ 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 $v$. 
$u$ and $v$ are connected if there is a path between $u$ and $v$.

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

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

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

May not be simple! Either modify definition to walk. Or cut out cycles.
$u$ and $v$ are connected if there is a path between $u$ and $v$.

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

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

Is graph connected? Yes? No?

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

May not be simple!
$u$ and $v$ are connected if there is a path between $u$ and $v$.

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

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

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

May not be simple! Either modify definition to walk.
$u$ and $v$ are connected if there is a path between $u$ and $v$.

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

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

Is graph connected? Yes? No?

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

May not be simple!
Either modify definition to walk.
Or cut out cycles.
$u$ and $v$ are connected if there is a path between $u$ and $v$.

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

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

Is graph connected? Yes? No?

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

May not be simple!
Either modify definition to walk.
Or cut out cycles.
Is graph above connected?

Yes!

How about now?

No!

Connected Components?

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

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

Quick Check: Is {10, 7, 5} a connected component? No.
Is graph above connected? Yes!

Connected Components:

\{1, 2, 9, 6\}, \{10, 7, 5, 8, 4, 3, 11\}.
Is graph above connected? Yes!
How about now?

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

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

Quick Check: Is \{10, 7, 5\} a connected component? No.
Is graph above connected? Yes!
How about now? No!

Connected Components?

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

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

Quick Check: Is \{10, 7, 5\} a connected component? No.
Is graph above connected? Yes!
How about now? No!

Connected Components?
Is graph above connected? Yes!

How about now? No!

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

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

Quick Check: Is \{10, 7, 5\} a connected component?
Is graph above connected? Yes!

How about now? No!

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

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

Quick Check: Is {10, 7, 5} a connected component? No.
Finally..back to Euler!

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

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

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

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

When you enter, you leave.
For starting node, tour leaves first ....then enters at end.
Finally..back to Euler!

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

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

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

Eulerian Tour is connected so graph is connected.

Tour enters and leaves vertex $v$ on each visit.

Uses two incident edges per visit. Tour uses all incident edges.

Therefore $v$ has even degree.

When you enter, you leave.

For starting node, tour leaves first ....then enters at end.
An Eulerian Tour is a tour that visits each edge exactly once.

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

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

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

When you enter, you leave.
For starting node, tour leaves first ....then enters at end.
An Eulerian Tour is a tour that visits each edge exactly once.

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

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

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

When you enter, you leave.
For starting node, tour leaves first ....then enters at end.
Finally..back to Euler!

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

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

**Proof of only if:** \( \text{Eulerian} \implies \text{connected and all even degree.} \)

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

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

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

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

When you enter, you leave.
An Eulerian Tour is a tour that visits each edge exactly once.

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

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

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

When you enter, you leave.

For starting node,
Finally..back to Euler!

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

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

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

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

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

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

**Proof of only if:** Eulerian \( \implies \) connected and all even degree.

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

When you enter, you leave.

For starting node, tour leaves first...then enters at end.
An Eulerian Tour is a tour that visits each edge exactly once.

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

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

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

When you enter, you leave.

For starting node, tour leaves first ... then enters at end.
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.

1. Take a walk starting from $v_1$...
till you get back to $v_1$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components.
   Each is touched by $C$.
   Why? $G$ was connected.
4. Let $v_i$ be (first) node in $G_i$ touched by $C$.
   Example: $v_1 = 1, v_2 = 10, v_3 = 4, v_4 = 2$.
5. Recurse on $G_1, \ldots, G_k$ starting from $v_i$.
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

1. Take a walk starting from $v(1)$
Finding a tour!

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

1. Take a walk starting from $v$ (1)

[Diagram of a graph with nodes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and edges connecting them.]
Finding a tour!

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

1. Take a walk starting from $v(1)$
Finding a tour!

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

1. Take a walk starting from $v$ (1)
Finding a tour!

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

1. Take a walk starting from $v (1)$
Finding a tour!

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

We will give an algorithm. First by picture.

1. Take a walk starting from $v(1)$
Finding a tour!

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

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

![Graph diagram]
Finding a tour!

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

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

1,10,7,8,5,10,8,4,3,11,4,5,2,6,9,2,1!
Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.

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

4. Recurse on $G_1, \ldots, G_k$ starting from $v_i$.
5. Splice together.
Finding a tour!

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

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

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

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

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

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

We will give an algorithm. First by picture.

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

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

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

Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$. 

1, 10, 7, 8, 5, 10, 8, 4, 3, 11, 4, 5, 2, 6, 9, 2 and to 1!
Finding a tour!

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

1. Take a walk starting from $v$ (1) ... till you get back to $v$.
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) ... 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.
Proof of if: Even + connected $\implies$ Eulerian Tour.

We will give an algorithm. First by picture.

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

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

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

\begin{center}
\begin{tikzpicture}
\node[circle,draw] (8) at (0,0) {8};
\node[circle,draw] (5) at (1,0) {5};
\node[circle,draw] (10) at (1,-1) {10};
\node[circle,draw] (7) at (-1,-1) {7};
\node[circle,draw] (2) at (0,-2) {2};
\node[circle,draw] (3) at (1,-2) {3};
\node[circle,draw] (9) at (2,-1) {9};
\node[circle,draw] (11) at (2,0) {11};
\node[circle,draw] (4) at (1,1) {4};
\draw (8) -- (5) -- (10) -- (7) -- (2) -- (3) -- (9) -- (4) -- (11) -- (3) -- (5);
\end{tikzpicture}
\end{center}
Finding a tour!

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

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

We will give an algorithm. First by picture.

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

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

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

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

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

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

We will give an algorithm. First by picture.

1. Take a walk starting from $v\ (1)$ till you get back to $v$.
2. Remove tour, $C$.
3. Let $G_1, \ldots, G_k$ be connected components. Each is touched by $C$.
   Why? $G$ was connected.
   Let $v_i$ be (first) node in $G_i$ touched by $C$.
   Example: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, $v_4 = 2$.
4. Recurse on $G_1, \ldots, G_k$ starting from $v_i$
5. Splice together.
   $1,10,7,8,5,10,8,4$
Proof of if: Even + connected $\implies$ Eulerian Tour.
We will give an algorithm. First by picture.

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

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

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

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

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

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

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

1. Take a walk from arbitrary node $v$, until you get back to $v$. 
General case: Recursive algorithm, proof by induction.

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

**Claim:** Do get back to \( v \)!

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 \( \Rightarrow \) a vertex in \( G_i \) must be incident to a removed edge in \( C \).

**Claim:** Each vertex in each \( G_i \) has even degree and is connected.

**Prf:** Tour \( C \) has even incidences to any vertex \( v \).

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

4. Splice \( T_i \) into \( C \) where \( v_i \) first appears in \( C \).

Visits every edge once:

- Visits edges in \( C \) exactly once.
- By induction for all other edges by induction on \( G_i \).
General case: Recursive algorithm, proof by induction.

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

**Claim:** Do get back to \( v \)!

**Proof of Claim:** Even degree.
General case: Recursive algorithm, proof by induction.

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

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

**Proof of Claim:** Even degree. If enter, can leave
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)

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

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

Why is there a $v_i$ in $C$?

$G$ was connected $\implies$ a vertex in $G_i$ must be incident to a removed edge in $C$.

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

**Prf:** Tour $C$ has even incidences to any vertex $v$.

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

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

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

By induction for all other edges by induction on $G_i$. 
General case: Recursive algorithm, proof by induction.

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

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

**Proof of Claim:** Even degree. If enter, can leave except for $v$. □
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$. 
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)

Let components be $G_1, \ldots, G_k$. 
General case: Recursive algorithm, proof by induction.

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

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

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 \( G_i \).
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)

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

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

Why is there a $v_i$ in $C$?
General case: Recursive algorithm, proof by induction.

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

Claim: Do get back to $v$!

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)

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

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

Why is there a $v_i$ in $C$?

$G$ was connected $\implies$
General case: Recursive algorithm, proof by induction.

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

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

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

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

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

   Why is there a $v_i$ in $C$?
   
   $G$ was connected $\implies$ a vertex in $G_i$ must be incident to a removed edge in $C$. 
General case: Recursive algorithm, proof by induction.

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

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

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

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

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

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

   **Why is there a $v_i$ in $C$?**

   $G$ was connected $\implies$ a vertex in $G_i$ must be incident to a removed edge in $C$. 

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

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

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

   By induction for all other edges by induction on $G_i$. 
General case: Recursive algorithm, proof by induction.

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

**Claim:** Do get back to \( v \)!

**Proof of Claim:** Even degree. If enter, can leave except for \( v \).

2. Remove cycle, \( C \), from \( G \).

Resulting graph may be disconnected. (Removed edges!)

Let components be \( G_1, \ldots, G_k \).

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

Why is there a \( v_i \) in \( C \)?

\( G \) was connected \( \implies \) a vertex in \( G_i \) must be incident to a removed edge in \( C \).

**Claim:** Each vertex in each \( G_i \) has even degree
General case: Recursive algorithm, proof by induction.

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

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

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

   Let components be \( G_1, \ldots, G_k \).

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

   Why is there a \( v_i \) in \( C \)?
   
   \( G \) was connected \( \implies \) a vertex in \( G_i \) must be incident to a removed edge in \( C \).

   **Claim:** Each vertex in each \( G_i \) has even degree and is connected.
General case: Recursive algorithm, proof by induction.

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

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

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

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 $v$. 
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)

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

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

Why is there a $v_i$ in $C$?

$G$ was connected $\implies$ a vertex in $G_i$ must be incident to a removed edge in $C$.

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

**Prf:** Tour $C$ has even incidences to any vertex $v$. □
General case: Recursive algorithm, proof by induction.

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

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

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

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

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

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

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

3. Find tour $T_i$ of $G_i$
General case: Recursive algorithm, proof by induction.

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

**Claim:** Do get back to \( v \)!

**Proof of Claim:** Even degree. If enter, can leave except for \( v \).

2. Remove cycle, \( C \), from \( G \).

Resulting graph may be disconnected. (Removed edges!)

Let components be \( G_1, \ldots, G_k \).

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

Why is there a \( v_i \) in \( C \)?

\( G \) was connected \( \iff \) 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 \( v_i \).
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

   Resulting graph may be disconnected. (Removed edges!)

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

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

   Why is there a $v_i$ in $C$?

   $G$ was connected $\Rightarrow$ a vertex in $G_i$ must be incident to a removed edge in $C$.

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

   **Prf:** Tour $C$ has even incidences to any vertex $v$.

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

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

Visits every edge once:

Visits edges in $C$ exactly once.

By induction for all other edges by induction on $G_i$.  

General case: Recursive algorithm, proof by induction.

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

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

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

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

Let components be $G_1, \ldots, G_k$. Let $v_i$ be first vertex of $C$ that is in $G_i$.

**Why is there a $v_i$ in $C$?**

$G$ was connected $\implies$ a vertex in $G_i$ must be incident to a removed edge in $C$.

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

**Prf:** Tour $C$ has even incidences to any vertex $v$. □

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

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

Visits every edge once:

Visit edges in $C$
General case: Recursive algorithm, proof by induction.

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

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

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

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

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

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

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

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

   Visits every edge once:
   Visits edges in $C$ exactly once.
General case: Recursive algorithm, proof by induction.

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

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

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

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

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

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

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

**Prf:** Tour $C$ has even incidences to any vertex $v$.

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

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

Visits every edge once:
   \begin{itemize}
   \item Visits edges in $C$ exactly once.
   \item By induction for all other edges by induction on $G_i$.
   \end{itemize}
General case: Recursive algorithm, proof by induction.

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

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

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

2. Remove cycle, $C$, from $G$.

Resulting graph may be disconnected. (Removed edges!)

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

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

**Why is there a $v_i$ in $C$?**

$G$ was connected $\implies$ a vertex in $G_i$ must be incident to a removed edge in $C$.

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

**Prf:** Tour $C$ has even incidences to any vertex $v$.

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

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

Visits every edge once:

- Visits edges in $C$ exactly once.

By induction for all other edges by induction on $G_i$. 

\[ \square \]
Summary

Graphs.
Summary

Graphs.
Basics.
Summary

Graphs.
Basics.
Connectivity.

Algorithm for Eulerian Tour.
Summary

Graphs.
  Basics.
Connectivity.
Algorithm for Eulerian Tour.
Summary

Graphs.
Basics.
Connectivity.
Algorithm for Eulerian Tour.
Summary

Graphs.
Basics.
Connectivity.
Algorithm for Eulerian Tour.