Lecture Today.

To homework (score) or not to homework (score)
To homework (score) or not to homework (score)
Do proofs of optimality/pessimality again.
Lecture Today.

To homework (score) or not to homework (score)
Do proofs of optimality/pessimality again.
Graphs
Job Propose and Candidate Reject is optimal!

For jobs?

Theorem:
Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not:
there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ knocks $b$ off of $g$'s string on day $t$.

$\Rightarrow g$ prefers $b^*$ to $b$.

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\Rightarrow b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.

So $S$ is not a stable pairing.

Contradiction.

Notes:
$S$ - stable.
$(b^*, g^*) \in S$.

But $(b^*, g)$ is rogue couple!

Used Well-Ordering principle...

Induction.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem:
Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job $b$ does not get optimal candidate, $g$.
There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b \star$ knocks $b$ off of $g$’s string on day $t = \Rightarrow g$ prefers $b \star$ to $b$.

By choice of $t$, $b \star$ likes $g$ at least as much as optimal candidate.

$= \Rightarrow b \star$ prefers $g$ to its partner $g \star$ in $S$.

Rogue couple for $S$.

So $S$ is not a stable pairing.

Contradiction.

Notes:
$S$ - stable.
$(b \star, g \star) \in S$.

But $(b \star, g)$ is rogue couple!

Used Well-Ordering principle...

Induction.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not:

Let $t$ be the first day job $b^*$ gets rejected by its optimal candidate $g$ whom it is paired with in stable pairing $S$.

$b^* -$ knocks $b^*$ off of $g$'s string on day $t = \Rightarrow g$ prefers $b^*$ to $b$ by choice of $t$.

$b^*$ likes $g$ at least as much as optimal candidate.

$b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.

So $S$ is not a stable pairing.

Contradiction.

Notes:
$S$ - stable. $(b^*, g^*) \in S$.
But $(b^*, g)$ is a rogue couple!

Used Well-Ordering principle...

Induction.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$. 
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
  by its optimal candidate $g$ who it is paired with
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem**: Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
by its optimal candidate $g$ who it is paired with
in stable pairing $S$. 

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
by its optimal candidate $g$ who it is paired with
in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies$ $g$ prefers $b^*$ to $b$
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected 
- by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t \implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job \( b \) does not get optimal candidate, \( g \).

There is a stable pairing \( S \) where \( b \) and \( g \) are paired.

Let \( t \) be first day job \( b \) gets rejected
by its optimal candidate \( g \) who it is paired with
in stable pairing \( S \).

\( b^* \) - knocks \( b \) off of \( g \)'s string on day \( t \) \( \implies \) \( g \) prefers \( b^* \) to \( b \)

By choice of \( t \), \( b^* \) likes \( g \) at least as much as optimal candidate.

\( \implies b^* \) prefers \( g \) to its partner \( g^* \) in \( S \).
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
   by its optimal candidate $g$ who it is paired with
   in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies$ $g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$. 
Job Propose and Candidate Reject is optimal!
For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected 
by its optimal candidate $g$ who it is paired with 
in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies$ $g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
by its optimal candidate $g$ who it is paired with
in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
   by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$'s string on day $t \implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.

Notes:
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t \implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.

Notes: $S$ - stable.
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
   by its optimal candidate $g$ who it is paired with
   in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t \implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.

Notes: $S$ - stable. $(b^*, g^*) \in S$. 

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.

□

Notes: $S$ - stable. $(b^*, g^*) \in S$. But $(b^*, g)$
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job \( b \) does not get optimal candidate, \( g \).

There is a stable pairing \( S \) where \( b \) and \( g \) are paired.

Let \( t \) be first day job \( b \) gets rejected by its optimal candidate \( g \) who it is paired with in stable pairing \( S \).

\( b^* \) - knocks \( b \) off of \( g \)'s string on day \( t \) \( \implies \) \( g \) prefers \( b^* \) to \( b \)

By choice of \( t \), \( b^* \) likes \( g \) at least as much as optimal candidate.

\( \implies b^* \) prefers \( g \) to its partner \( g^* \) in \( S \).

Rogue couple for \( S \).
So \( S \) is not a stable pairing. Contradiction.

Notes: \( S \) - stable. \((b^*, g^*) \in S\). But \((b^*, g)\) is rogue couple!
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected by its optimal candidate $g$ who it is paired with in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t \implies g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.

So $S$ is not a stable pairing. Contradiction.

Notes: $S$ - stable. $(b^*, g^*) \in S$. But $(b^*, g)$ is rogue couple!

Used Well-Ordering principle...
Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

**Theorem:** Job Propose and Reject produces a job-optimal pairing.

**Proof:**
Assume not: there is a job $b$ does not get optimal candidate, $g$.

There is a stable pairing $S$ where $b$ and $g$ are paired.

Let $t$ be first day job $b$ gets rejected
by its optimal candidate $g$ who it is paired with
in stable pairing $S$.

$b^*$ - knocks $b$ off of $g$’s string on day $t$ $\implies$ $g$ prefers $b^*$ to $b$

By choice of $t$, $b^*$ likes $g$ at least as much as optimal candidate.

$\implies b^*$ prefers $g$ to its partner $g^*$ in $S$.

Rogue couple for $S$.
So $S$ is not a stable pairing. Contradiction.

Notes: S - stable. $(b^*, g^*) \in S$. But $(b^*, g)$ is rogue couple!
Used Well-Ordering principle...Induction.
How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T - pairing produced by JPR.

S - worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b∗) is pair.

g prefers b to b∗.

T is job optimal, so b prefers g to its partner in S.

(??, ?) is Rogue couple for S.

S is not stable.

Contradiction.

Notes: Not really induction.

Structural statement: Job optimality =⇒ Candidate pessimality.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

- **T** – pairing produced by JPR.
- **S** – worse stable pairing for candidate \( g \).

In **T**, \((g, b)\) is pair.

In **S**, \((g, b^*\) is pair.

\( g \) prefers \( b \) to \( b^* \).

**T** is job optimal, so \( b \) prefers \( g \) to its partner in **S**.

\((g, b)\) is Rogue couple for **S**.

**S** is not stable.

Contradiction.

**Notes:**
Not really induction.
Structural statement: Job optimality \( \Rightarrow \) Candidate pessimality.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

- $T$ – pairing produced by JPR.
- $S$ – worse stable pairing for candidate $g$.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$. 

How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

*T* – pairing produced by JPR.

*S* – worse stable pairing for candidate *g*.

In *T*, *(g, b)* is pair.

In *S*, *(g, b*) is pair.

*g* prefers *b* to *b*.

*T* is job optimal, so *b* prefers *g* to its partner in *S*.
How about for candidates?

**Theorem**: Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g, b)$ is Rogue couple for $S$. 
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g, b)$ is Rogue couple for $S$

$S$ is not stable.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g,b)$ is pair.

In $S$, $(g,b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g,b)$ is Rogue couple for $S$.

$S$ is not stable.

Contradiction.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g, b)$ is Rogue couple for $S$

$S$ is not stable.

**Contradiction.**
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g, b)$ is Rogue couple for $S$.

$S$ is not stable.

**Contradiction.**

Notes:
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g, b)$ is Rogue couple for $S$.

$S$ is not stable.

**Contradiction.**

Notes: Not really induction.
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

*T* – pairing produced by JPR.

*S* – worse stable pairing for candidate *g*.

In *T*, *(g, b)* is pair.

In *S*, *(g, b*) is pair.

*g* prefers *b* to *b*.

*T* is job optimal, so *b* prefers *g* to its partner in *S*.

*(g, b)* is Rogue couple for *S*

*S* is not stable.

**Contradiction.**

Notes: Not really induction.

  Structural statement: Job optimality
How about for candidates?

**Theorem:** Job Propose and Reject produces candidate-pessimal pairing.

$T$ – pairing produced by JPR.

$S$ – worse stable pairing for candidate $g$.

In $T$, $(g, b)$ is pair.

In $S$, $(g, b^*)$ is pair.

$g$ prefers $b$ to $b^*$.

$T$ is job optimal, so $b$ prefers $g$ to its partner in $S$.

$(g, b)$ is Rogue couple for $S$.

$S$ is not stable.

**Contradiction.**

Notes: Not really induction.

Structural statement: Job optimality $\Rightarrow$ Candidate pessimality.
Lecture 5: Graphs.

Graphs!
Lecture 5: Graphs.

Graphs!
Definitions: model.
Lecture 5: Graphs.

Graphs!
  Definitions: model.
  Fact!
Graphs!
Definitions: model.
Fact!
Lecture 5: Graphs.

Graphs!
Definitions: model.
Fact!
Planar graphs.
Lecture 5: Graphs.

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

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.
Map Coloring.

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.
Map Coloring.

Fewer Colors?

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.

Fewer Colors?
Map Coloring.

Yes! Three colors.
Map Coloring.

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.
Map Coloring.

Fewer Colors?  Yes! Three colors.

Fewer Colors?  Four colors required!

Theorem: Four colors enough.
Map Coloring.

Fewer Colors? Yes! Three colors.

Fewer Colors? Four colors required!

Theorem: Four colors enough.
Map Coloring.

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.
Map Coloring.

Fewer Colors? Yes! Three colors.

Fewer Colors? Four colors required!

Theorem: Four colors enough.
Map Coloring.

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.
Map Coloring.

Fewer Colors?

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.

Fewer Colors?
Map Coloring.

Yes! Three colors.

Four colors required!

Theorem: Four colors enough.
Map Coloring.

Four colors required!
Map Coloring.

Four colors required!

Theorem: Four colors enough.
Scheduling: coloring.
Scheduling: coloring.

Exam Slot 1.
Exam Slot 2.
Exam Slot 3.
Scheduling: coloring.
Scheduling: coloring.

Exam Slot 1.
Exam Slot 2.
Exam Slot 3.
Scheduling: coloring.

Diagram:

- 61B
- 61C
- 61A
- 70
- 170

Exam Slot 1.
Exam Slot 2.
Exam Slot 3.
Scheduling: coloring.
Scheduling: coloring.
Scheduling: coloring.
Scheduling: coloring.

Diagram:
- 61B
- 61C
- 61A
- 170
- 70

Exam Slot 1.
Exam Slot 2.
Exam Slot 3.
Scheduling: coloring.
Scheduling: coloring.
Scheduling: coloring.
Scheduling: coloring.
Scheduling: coloring.

Exam Slot 1.
Exam Slot 2.
Exam Slot 3.
Scheduling: coloring.

Exam Slot 1.
Exam Slot 2.
Exam Slot 3.
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: $G = (V, E)$.

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

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

Graph: \( G = (V, E) \).  
\( V \) - set of vertices.
Graphs: formally.

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

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

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

$V$ - set of vertices.

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

$E \subseteq V \times V$ - set of edges.
Graphs: formally.

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

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

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

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

$V$ - set of vertices.

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

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

$\{\{A, B\}, \{A, B\}, \{A, C\},$
Graphs: formally.

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

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

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

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

\( V \) - set of vertices. 
\( \{A, B, C, D\} \)

\( E \subseteq V \times V \) - set of edges.
\( \{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{B, D\}, \{A, D\}, \{C, D\}\} \).

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

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

$V$ - set of vertices.

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

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

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

For CS 70, usually simple graphs.

No parallel edges.

Multigraph above.
Directed Graphs

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

$G = (V, E)$.  
$V$ - set of vertices.
Directed Graphs

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

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

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.

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

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

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.
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.
  - Friends.
- Undirected.
  - Likes.
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:
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

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

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

Social Network: Directed? Undirected?
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

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

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

Social Network: Directed? Undirected?
Friends. 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 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?

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 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.

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.
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 \emph{neighbor} of \( v \) if \( \{u, v\} \in E \).

Edge \( \{10, 5\} \) is \emph{incident} to vertex 10 and vertex 5.
- Edge \( \{u, v\} \) is \emph{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
Sum of degrees?

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 $2|V|$?
The sum of the vertex degrees is equal to

(A) the total number of vertices, $|V|$. 
The sum of the vertex degrees is equal to

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

What?

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

Could it always be $2|E|$ or $2|V|$?
Sum of degrees?

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.
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.
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)!
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.
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.
Sum of degrees?

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?
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.
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...
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... $2|E|$? ..
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 of an Equality.

The sum of the vertex degrees is equal to ??
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
- edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
- degree of \(u\) number of edges incident to \(u\)

Let's count incidences in two ways.
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

- edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
- degree of \(u\) number of edges incident to \(u\)

Let's count incidences in two ways.
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

How many incidences does each edge contribute?
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.
  How many incidences does each edge contribute? 2.
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

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

The sum of the vertex degrees is equal to ??

Recall:
- edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
- degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

  How many incidences does each edge contribute? 2.
  Total Incidences?
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

How many incidences does each edge contribute? 2.
Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).
degree of \(u\) number of edges incident to \(u\)

Let's count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)?
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is **incident** to endpoints, \(u\) and \(v\).
degree of \(u\) number of edges **incident** to \(u\)

Let’s count incidences in two ways.

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

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)? Incidences corresponding to \(v\)!
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let's count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)? Incidences corresponding to \(v\)!
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)? Incidences corresponding to \(v\)!

Total Incidences?
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:

edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)? Incidences corresponding to \(v\)!

Total Incidences? The sum over vertices of degrees!
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let’s count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)? Incidences corresponding to \(v\)!

Total Incidences? The sum over vertices of degrees!
Quick Proof of an Equality.

The sum of the vertex degrees is equal to ??

Recall:
edge, \((u, v)\), is incident to endpoints, \(u\) and \(v\).

degree of \(u\) number of edges incident to \(u\)

Let's count incidences in two ways.

How many incidences does each edge contribute? 2.

Total Incidences? \(|E|\) edges, 2 each. \(\rightarrow 2|E|\)

What is degree \(v\)? Incidences corresponding to \(v\)!

Total Incidences? The sum over vertices of degrees!

**Thm:** Sum of vertex degrees is \(2|E|\).
A path in a graph is a sequence of edges.
A path in a graph is a sequence of edges.

Path?

A path in a graph is a sequence of edges.

Path?

Path?

Path?

Path?

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\} ?

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

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

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\}?
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!
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.
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?
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 simple.
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!
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.
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 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 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!
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.

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

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

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

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, walks,
Directed Paths.

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

Paths, walks, cycles,
Directed Paths.

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

Paths, walks, cycles, tours
Directed Paths.

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

Paths, walks, cycles, tours ... are analogous to undirected now.
Connectivity: undirected graph.

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

![Graph Image]
Connectivity: undirected graph.

\[ u \text{ and } v \text{ are connected if there is a path between } u \text{ and } v. \]

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

$u$ 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: undirected graph.

$u$ 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: undirected graph.

\[ u \text{ and } v \text{ are connected if there is a path between } u \text{ and } v. \]

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?

\[ \text{Yes?} \]
Connectivity: undirected graph.

$u$ 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: undirected graph.

$u$ 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: undirected graph.

$u$ 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: undirected graph.

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

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

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

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

May not be simple!
Either modify definition to walk.
Or cut out cycles.
Connectivity: undirected graph.

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

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

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

Is graph connected? Yes? No?

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

May not be simple!
Connectivity: undirected graph.

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

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

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

Is graph connected? Yes? No?

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

May not be simple!

Either modify definition to walk.
Connectivity: undirected graph.

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

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

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

Is graph connected? Yes? No?

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

May not be simple!
Either modify definition to walk.
Or cut out cycles.
Connectivity: undirected graph.

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

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

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

Is graph connected? Yes? No?

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

May not be simple!

Either modify definition to walk.

Or cut out cycles. 
Connectivity: undirected graph.

$u$ 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.
Connected Components: Quiz.

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.
Connected Components: Quiz.

Is graph above connected? Yes!

Connected Components?

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

Connected component - maximal set of connected vertices.

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

Is graph above connected? Yes!

How about now?

Connected component - maximal set of connected vertices.
Connected Components: Quiz.

Is graph above connected? Yes!
How about now? No!

Connected component - maximal set of connected vertices.

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

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\}. 
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.
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?
Connected Components: Quiz.

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.
Konigsberg bridges problem.

Can you make a tour visiting each bridge exactly once?

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

Can you make a tour visiting each bridge exactly once?

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

Can you make a tour visiting each bridge exactly once?

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

Can you make a tour visiting each bridge exactly once?

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

Can you make a tour visiting each bridge exactly once?

“Konigsberg bridges” by Bogdan Giuscă - License.
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” by Bogdan Giuscă - 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 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?

“Konigsberg bridges” by Bogdan Giuscă - 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?

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

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

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.
Eulerian Tour

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

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.
Eulerian Tour

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.
Eulerian Tour

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.
Eulerian Tour

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.
Eulerian Tour

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.
Eulerian Tour

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.
Eulerian Tour

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.
Eulerian Tour

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 $\Rightarrow$ 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,
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 can leave.
Eulerian Tour

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 can 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 $\Rightarrow$ 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 can leave.
Eulerian Tour

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 can leave.
For starting node,
Eulerian Tour

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 can leave.
For starting node, tour leaves first.
Eulerian Tour

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 can leave. For starting node, tour leaves first ....then enters at end.
Eulerian Tour

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 can leave.
For starting node, tour leaves first ....then enters at end.
Eulerian Tour

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 can leave.
For starting node, tour leaves first ....then enters at end.
Not The Hotel California.
Finding a tour!

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

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

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

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

![Graph Image]
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

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

\begin{center}
\begin{tikzpicture}
  \node[circle, draw] (1) at (0,0) {1};
  \node[circle, draw] (2) at (1,-1) {2};
  \node[circle, draw] (3) at (2,-1) {3};
  \node[circle, draw] (4) at (2,1) {4};
  \node[circle, draw] (5) at (1,1) {5};
  \node[circle, draw] (6) at (0,-1) {6};
  \node[circle, draw] (7) at (-1,0) {7};
  \node[circle, draw] (8) at (-1,1) {8};
  \node[circle, draw] (9) at (1,-2) {9};
  \node[circle, draw] (10) at (-2,-1) {10};
  \node[circle, draw] (11) at (2,0) {11};

  \draw (1) -- (2) -- (3);
  \draw (2) -- (4) -- (5);
  \draw (4) -- (6);
  \draw (5) -- (7) -- (8);
  \draw (7) -- (10);
  \draw (8) -- (11);
  \draw (10) -- (9);
  \draw (9) -- (3);
\end{tikzpicture}
\end{center}
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

1
2
3
4
5
6
7
8
9
10
11
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

![Diagram of a graph with nodes and edges labeled from 1 to 11, showing a path that starts at node 1 and ends at node 11, with intermediate nodes and edges connecting them.]
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
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

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) on “unused” edges
   ... till you get back to \(v\).

\[
\begin{align*}
8 & \rightarrow 4 & 11 \\
7 & \rightarrow 5 & 3 \\
10 & \rightarrow & 9 \\
1 & \rightarrow 2 & 6
\end{align*}
\]
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, $C$.

![Diagram of a graph with numbered nodes and arrows showing the path for an Eulerian tour.](image)
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.
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 $C$.
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?

![Graph diagram](image-url)
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.
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 $C$.
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: $v_1 = 1$,
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: $v_1 = 1$, $v_2 = 10$, $v_3 = 4$, 

![Diagram of a graph with nodes and edges labeled from 1 to 11. The walk starts at node 1, visits nodes 10, 5, 4, 11, 1, 2, 6, 9, 3, 5, and 11, and returns to node 1.](image-url)
Finding a tour!

Proof of if: Even + connected $\Rightarrow$ 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 $v_i$
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.
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) 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

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.

**Proof:** 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 $v$.  

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 $C$.

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 \( v \).
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 $G$.

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 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 \( v \).

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 $v$. □

2. Remove cycle, $C$, from $G$.
   Resulting graph may be disconnected. (Removed edges!)
   Let components be $G_1, \ldots, G_k$. 

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 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 $G_i$. 
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 $v$. □

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 $v$. □

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 $v$. 

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 $v_i$.

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 $v$.

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
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 \( \implies \) 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 \( v \).

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 $v$.  

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 \( v \).

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 \( G_i \)
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 $\implies$ 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 $v_i$. 
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 $\implies$ 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 $v_i$. 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 \( v \).

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 \( v_i \). 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 \( v \).

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 \( \Rightarrow \)
\hspace{1cm} 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:
\hspace{1cm} 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 $v$.  

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 $v_i$. 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 $\implies$ 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 $v_i$. 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 \( v \).

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 \( v_i \). 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 \( G_i \).
A Tree, a tree.

Graph $G = (V, E)$.
Binary Tree!

More generally.
Trees.

Definitions:

A connected graph without a cycle.

A connected graph with $|V| - 1$ edges.

A connected graph where any edge removal disconnects it.

A connected graph where any edge addition creates a cycle.

Some trees. No cycle and connected? Yes. $|V| - 1$ edges and connected? Yes. Removing any edge disconnects it. Harder to check. But yes. Adding any edge creates a cycle. Harder to check. But yes. To tree or not to tree!
Trees.

Definitions:

A connected graph without a cycle.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

- No cycle and connected? Yes.
- $|V| - 1$ edges and connected? Yes.
- Removing any edge disconnects it. Harder to check. But yes.
- Adding any edge creates cycle. Harder to check. But yes.

To tree or not to tree!
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

no cycle and connected?
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

no cycle and connected? Yes.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

![Tree 1](image1)
![Tree 2](image2)

no cycle and connected? Yes.
$|V| - 1$ edges and connected?
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

No cycle and connected? Yes.
$|V| - 1$ edges and connected? Yes.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

no cycle and connected? Yes.
$|V| - 1$ edges and connected? Yes.
removing any edge disconnects it.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

no cycle and connected? Yes.
$|V| - 1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check.
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

- No cycle and connected? Yes.
- $|V| - 1$ edges and connected? Yes.
- Removing any edge disconnects it. Harder to check. but yes.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with \(|V| - 1\) edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

![Tree 1](image1)
![Tree 2](image2)

no cycle and connected? Yes.
\(|V| - 1\) edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

no cycle and connected? Yes.
$|V| - 1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle. Harder to check.
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

![Trees](image)

- no cycle and connected? Yes.
- $|V| - 1$ edges and connected? Yes.
- removing any edge disconnects it. Harder to check. but yes.
- Adding any edge creates cycle. Harder to check. but yes.
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

![Tree 1](image1)

![Tree 2](image2)

no cycle and connected? Yes.
$|V| - 1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle. Harder to check. but yes.
Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V| - 1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.

![Trees](image)

no cycle and connected? Yes.
$|V| - 1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle. Harder to check. but yes.

To tree or not to tree!
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

<table>
<thead>
<tr>
<th>No cycle and connected?</th>
<th>Yes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>V</td>
</tr>
</tbody>
</table>

Removing any edge disconnects it. Harder to check, but yes.
Adding any edge creates cycle. Harder to check, but yes.

To tree or not to tree!
Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.

- No cycle and connected? Yes.
- $|V| - 1$ edges and connected? Yes.
- Removing any edge disconnects it. Harder to check. But yes.
- Adding any edge creates a cycle. Harder to check. But yes.

To tree or not to tree!
Equivalence of Definitions.

**Theorem:**
“$G$ connected and has $|V| - 1$ edges” $\equiv$
“$G$ is connected and has no cycles.”
Equivalence of Definitions.

**Theorem:**
“G connected and has $|V| - 1$ edges” $\equiv$

“G is connected and has no cycles.”

**Lemma:** If $v$ is degree 1 in connected graph $G$, $G - v$ is connected.

**Proof:**
For $x \neq v, y \neq v \in V$, 

Equivalence of Definitions.

**Theorem:**
“$G$ connected and has $|V| - 1$ edges” $\equiv$
“$G$ is connected and has no cycles.”

**Lemma:** If $v$ is degree 1 in connected graph $G$, $G - v$ is connected.

**Proof:**
For $x \neq v, y \neq v \in V$,
there is path between $x$ and $y$ in $G$ since connected.
Equivalence of Definitions.

**Theorem:**
“G connected and has $|V| - 1$ edges” $\equiv$
“G is connected and has no cycles.”

**Lemma:** If $v$ is degree 1 in connected graph $G$, $G - v$ is connected.

**Proof:**
For $x \neq v, y \neq v \in V$,
there is path between $x$ and $y$ in $G$ since connected.
and does not use $v$ (degree 1)
Equivalence of Definitions.

**Theorem:**
“$G$ connected and has $|V| - 1$ edges” $\equiv$
“$G$ is connected and has no cycles.”

**Lemma:** If $v$ is degree 1 in connected graph $G$, $G - v$ is connected.

**Proof:**
For $x \neq v, y \neq v \in V$,
there is path between $x$ and $y$ in $G$ since connected.
and does not use $v$ (degree 1)
$\implies G - v$ is connected.
Equivalence of Definitions.

**Theorem:**
“$G$ connected and has $|V| - 1$ edges” $\equiv$
“$G$ is connected and has no cycles.”

**Lemma:** If $v$ is degree 1 in connected graph $G$, $G - v$ is connected.

**Proof:**
For $x \neq v, y \neq v \in V$,
there is path between $x$ and $y$ in $G$ since connected.
and does not use $v$ (degree 1)
$\implies G - v$ is connected.
Equivalence of Definitions.

**Theorem:**
“G connected and has \(|V| - 1\) edges” $\equiv$
“G is connected and has no cycles.”

**Lemma:** If $v$ is degree 1 in connected graph $G$, $G - v$ is connected.

**Proof:**
For $x \neq v, y \neq v \in V$,
there is path between $x$ and $y$ in $G$ since connected.
and does not use $v$ (degree 1)
$\implies G - v$ is connected.
Proof of only if.

**Thm:**
"G connected and has \(|V| - 1\) edges" $\implies$
"G is connected and has no cycles."

**Proof of $\implies$:**
Proof of only if.

**Thm:**
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

**Proof of $\implies$:** By induction on $|V|$.

![Graph](image)
Proof of only if.

Thm:
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.
Proof of only if.

**Thm:**
“$G$ connected and has $|V| - 1$ edges” $\implies$ “$G$ is connected and has no cycles.”

**Proof of** $\implies$ : By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.
Proof of only if.

**Thm:**
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

**Proof of $\implies$:** By induction on $|V|$.

**Base Case:** $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

**Induction Step:**
Proof of only if.

Thm:
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof of only if.

Thm:
“G connected and has $|V| - 1$ edges” $\implies$ “G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. 0 = $|V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree $\geq 1$. 
Proof of only if.

Thm: 
“G connected and has $|V| - 1$ edges” $\implies$ 
“G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.

Proof: First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
Proof of only if.

**Thm:**
“$G$ connected and has $|V| - 1$ edges” $\implies$ “$G$ is connected and has no cycles.”

**Proof of $\implies$:** By induction on $|V|$.

Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:

**Claim:** There is a degree 1 node.

**Proof:** First, connected $\implies$ every vertex degree $\geq 1$.

- Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
- Average degree $2 - 2/|V|$
Proof of only if.

Thm: “G connected and has $|V| - 1$ edges” $\implies$ “G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
Average degree $2 - 2/|V|$
Not everyone is bigger than average!
Proof of only if.

Thm:
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
Average degree $2 - 2/|V|$
Not everyone is bigger than average!
Proof of only if.

**Thm:**
“G connected and has |V| − 1 edges” \(\iff\) “G is connected and has no cycles.”

**Proof of \(\iff\):** By induction on |V|.

**Base Case:** |V| = 1. 0 = |V| − 1 edges and has no cycles.

**Induction Step:**
**Claim:** There is a degree 1 node.

**Proof:** First, connected \(\iff\) every vertex degree \(\geq 1\).

- Sum of degrees is \(2|E| = 2(|V| - 1) = 2|V| - 2\)
- Average degree \(2 - 2/|V|\)

Not everyone is bigger than average!

By degree 1 removal lemma, \(G - v\) is connected.
Proof of only if.

Thm:
“$G$ connected and has $|V| - 1$ edges” $\implies$ 
“$G$ is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. 0 = $|V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.

Proof: First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
Average degree $2 - 2/|V|$
Not everyone is bigger than average!

By degree 1 removal lemma, $G - v$ is connected.
$G - v$ has $|V| - 1$ vertices and $|V| - 2$ edges so by induction
Proof of only if.

**Thm:**
“G connected and has $|V|−1$ edges” $\implies$
“G is connected and has no cycles.”

**Proof of $\implies$:** By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| − 1$ edges and has no cycles.

Induction Step:
**Claim:** There is a degree 1 node.

**Proof:** First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V|−1) = 2|V|−2$
Average degree $2 − 2/|V|$  
Not everyone is bigger than average!

By degree 1 removal lemma, $G − v$ is connected.
$G − v$ has $|V|−1$ vertices and $|V|−2$ edges so by induction $\implies$ no cycle in $G − v$.  

\[\square\]
Proof of only if.

Thm:
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
Average degree $2 - 2/|V|$
Not everyone is bigger than average!
By degree 1 removal lemma, $G - v$ is connected.
$G - v$ has $|V| - 1$ vertices and $|V| - 2$ edges so by induction
$\implies$ no cycle in $G - v$.
And no cycle in $G$ since degree 1 cannot participate in cycle.
Proof of only if.

Thm:
“G connected and has $|V| - 1$ edges” $\implies$
“G is connected and has no cycles.”

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree $\geq 1$.
Sum of degrees is $2|E| = 2(|V| - 1) = 2|V| - 2$
Average degree $2 - 2/|V|$
Not everyone is bigger than average!

By degree 1 removal lemma, $G - v$ is connected.
$G - v$ has $|V| - 1$ vertices and $|V| - 2$ edges so by induction
$\implies$ no cycle in $G - v$.
And no cycle in $G$ since degree 1 cannot participate in cycle.
Proof of if

**Thm:**
“$G$ is connected and has no cycles”

$\implies$ “$G$ connected and has $|V| - 1$ edges”

**Proof:**

Walk from a vertex using untraversed edges. Until get stuck.

Claim:
Degree 1 vertex.

Proof of Claim:
Can’t visit more than once since no cycle.

Entered. Didn’t leave.
Only one incident edge.
Removing node doesn’t create cycle.
New graph is connected.
Removing degree 1 node doesn’t disconnect from Degree 1 lemma.

By induction $G - v$ has $|V| - 2$ edges.

$G$ has one more or $|V| - 1$ edges.
Proof of if

**Thm:**
“G is connected and has no cycles”
\[ \implies \text{“G connected and has } |V| - 1 \text{ edges”} \]

**Proof:**
Walk from a vertex using untraversed edges.
Thm:
“G is connected and has no cycles”
⇒ “G connected and has $|V| - 1$ edges”

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Proof of if

**Thm:**
“G is connected and has no cycles”
\[ \Rightarrow \] “G connected and has \(|V| - 1\) edges”

**Proof:**
Walk from a vertex using untraversed edges.
Until get stuck.

**Claim:** Degree 1 vertex.
Proof of if

**Thm:**
“G is connected and has no cycles”
⇒ “G connected and has |V| − 1 edges”

**Proof:**
Walk from a vertex using untraversed edges.
Until get stuck.

**Claim:** Degree 1 vertex.

**Proof of Claim:**
Can’t visit more than once since no cycle.
Proof of if

Thm:
“G is connected and has no cycles”  
⇒  “G connected and has |V| – 1 edges”

Proof:
Walk from a vertex using untraversed edges.  
Until get stuck.

Claim: Degree 1 vertex.

Proof of Claim:
Can’t visit more than once since no cycle.  
Entered.
Proof of if

**Thm:**
“G is connected and has no cycles”
\[ \implies \text{“G connected and has } |V| - 1 \text{ edges”} \]

**Proof:**
Walk from a vertex using untraversed edges.
Until get stuck.

**Claim:** Degree 1 vertex.

**Proof of Claim:**
Can’t visit more than once since no cycle.
Entered. Didn’t leave.
**Thm:**
"G is connected and has no cycles"
\[\implies \text{“G connected and has } |V| - 1 \text{ edges”}\]

**Proof:**
Walk from a vertex using untraversed edges.
Until get stuck.

**Claim:** Degree 1 vertex.

**Proof of Claim:**
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.
Proof of if

**Thm:**
“$G$ is connected and has no cycles”
$$\implies \text{“$G$ connected and has } |V| - 1 \text{ edges”}$$

**Proof:**
Walk from a vertex using untraversed edges.
Until get stuck.

**Claim:** Degree 1 vertex.

**Proof of Claim:**
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.
Removing node doesn’t create cycle.
Proof of if

Thm:
“\(G\) is connected and has no cycles”
\[ \implies \text{“}\(G\) connected and has } |V| - 1 \text{ edges”} \]

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Degree 1 vertex.
Proof of Claim:
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.
Removing node doesn’t create cycle.
New graph is connected.
Proof of if

Thm:
“G is connected and has no cycles”
⇒ “G connected and has $|V| - 1$ edges”

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Degree 1 vertex.

Proof of Claim:
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.
Removing node doesn’t create cycle.
New graph is connected.
Removing degree 1 node doesn’t disconnect from Degree 1 lemma.
Proof of if

**Thm:**
“G is connected and has no cycles”

⇒ “G connected and has |V|−1 edges”

**Proof:**
Walk from a vertex using untraversed edges.
Until get stuck.

**Claim:** Degree 1 vertex.

**Proof of Claim:**
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.

Removing node doesn’t create cycle.
New graph is connected.
Removing degree 1 node doesn’t disconnect from Degree 1 lemma.
By induction $G - v$ has $|V| - 2$ edges.
Proof of if

Thm:
“G is connected and has no cycles”
⇒ “G connected and has \(|V| − 1\) edges”

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Degree 1 vertex.
Proof of Claim:
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.
Removing node doesn’t create cycle.
New graph is connected.
Removing degree 1 node doesn’t disconnect from Degree 1 lemma.
By induction \(G − v\) has \(|V| − 2\) edges.
\(G\) has one more or \(|V| − 1\) edges.
Proof of if

Thm:
“G is connected and has no cycles”
⇒ “G connected and has |V| − 1 edges”

Proof:
Walk from a vertex using untraversed edges. Until get stuck.

Claim: Degree 1 vertex.

Proof of Claim:
Can’t visit more than once since no cycle.
Entered. Didn’t leave. Only one incident edge.

Removing node doesn’t create cycle.
New graph is connected.
Removing degree 1 node doesn’t disconnect from Degree 1 lemma.
By induction \( G - v \) has \(|V| - 2\) edges.
\( G \) has one more or \(|V| - 1\) edges.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar?
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete?

(Complete $\equiv$ every edge present. $K_n$ is $n$-vertex complete graph.)

Two to three nodes, bipartite? Yes.
Three to three nodes, complete/bipartite or $K_3, 3$. No.
Why? Later.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete \equiv \text{every edge present. } K_n \text{ is } n\text{-vertex complete graph.})
Five node complete or \(K_5\)?
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete $\equiv$ every edge present. $K_n$ is $n$-vertex complete graph.)
Five node complete or $K_5$? No!
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.

(complete $\equiv$ every edge present. $K_n$ is $n$-vertex complete graph.)
Five node complete or $K_5$? No! Why?
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.

Four node complete? Yes.
(complete ≡ every edge present. $K_n$ is $n$-vertex complete graph. )

Five node complete or $K_5$? No! Why? Later.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.

(complete \equiv every\ edge\ present. K_n\ is\ n\ -vertex\ complete\ graph. )
Five node complete or $K_5$? No! Why? Later.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete \( \equiv \) every edge present. \( K_n \) is \( n \)-vertex complete graph. )
Five node complete or \( K_5 \) ? No! Why? Later.

Two to three nodes, bipartite?
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete $\equiv$ every edge present. $K_n$ is $n$-vertex complete graph.)
Five node complete or $K_5$? No! Why? Later.

Two to three nodes, bipartite? Yes.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete \equiv every edge present. \(K_n\) is \(n\)-vertex complete graph.)
Five node complete or \(K_5\) ? No! Why? Later.

Two to three nodes, bipartite? Yes.
Three to three nodes, complete/bipartite or \(K_{3,3}\).
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete \(\equiv\) every edge present. \(K_n\) is \(n\)-vertex complete graph.)
Five node complete or \(K_5\) ? No! Why? Later.

Two to three nodes, bipartite? Yes.
Three to three nodes, complete/bipartite or \(K_{3,3}\). No.
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete $\equiv$ every edge present. $K_n$ is $n$-vertex complete graph.)
Five node complete or $K_5$? No! Why? Later.

Two to three nodes, bipartite? Yes.
Three to three nodes, complete/bipartite or $K_{3,3}$. No. Why?
Planar graphs.

A graph that can be drawn in the plane without edge crossings.

Planar? Yes for Triangle.
Four node complete? Yes.
(complete $\equiv$ every edge present. $K_n$ is $n$-vertex complete graph.)
Five node complete or $K_5$? No! Why? Later.

Two to three nodes, bipartite? Yes.
Three to three nodes, complete/bipartite or $K_{3,3}$. No. Why? Later.