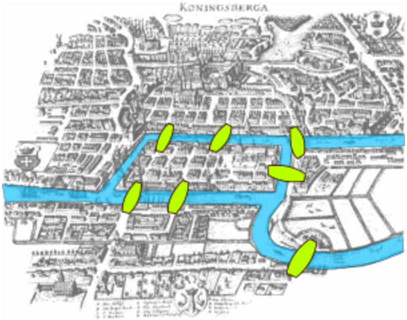
Lecture 4: Graph Theory 1

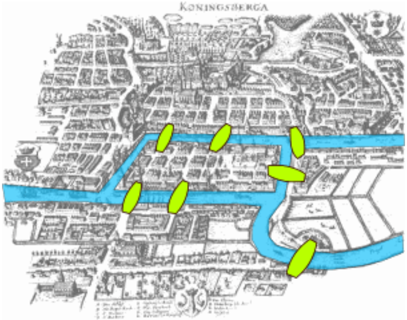
In Which We Draw a Bunch Of Pretty Pictures

The Seven Bridges of Königsberg



Source: Wikipedia

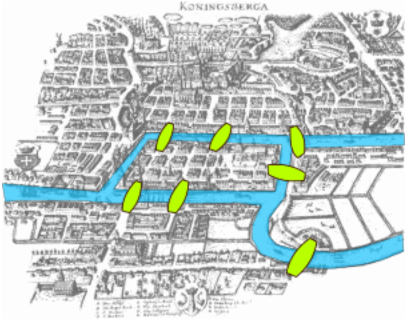
The Seven Bridges of Königsberg



Source: Wikipedia

Cross each bridge once and end where you started?

The Seven Bridges of Königsberg



Source: Wikipedia

Cross each bridge once and end where you started?

Turns out, impossible! Proved by Euler in 1736, inventing graph theory to do so.

What Is a Graph?

A *graph* $G = (V, E)$ is:

- A set of *vertices*[1] V
- A set of *edges* $E \subseteq V \times V$

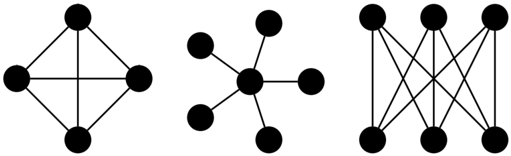
[1] Sometimes also known as *nodes*

What Is a Graph?

A *graph* $G = (V, E)$ is:

- A set of *vertices*[1] V
- A set of *edges* $E \subseteq V \times V$

Visualizations:



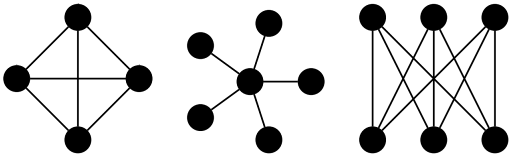
[1]Sometimes also known as *nodes*

What Is a Graph?

A *graph* $G = (V, E)$ is:

- A set of *vertices*[1] V
- A set of *edges* $E \subseteq V \times V$

Visualizations:



Not the graph of a function!

[1] Sometimes also known as *nodes*

Can You Give Me Directions?

Edges model relationships between vertices

Can You Give Me Directions?

Edges model relationships between vertices

Relations could be mutual (friends on Facebook)

- Treat edges as unordered sets $\{u, v\}$
- Called *undirected* graphs

Can You Give Me Directions?

Edges model relationships between vertices

Relations could be mutual (friends on Facebook)

- Treat edges as unordered sets $\{u, v\}$
- Called *undirected* graphs

Or only one direction (follow on Twitter)

- Treat edges as ordered pairs (u, v)
- Called *directed* graphs
- Use arrows to show direction

Can You Give Me Directions?

Edges model relationships between vertices

Relations could be mutual (friends on Facebook)

- Treat edges as unordered sets $\{u, v\}$
- Called *undirected* graphs

Or only one direction (follow on Twitter)

- Treat edges as ordered pairs (u, v)
- Called *directed* graphs
- Use arrows to show direction

Focus (mainly) on undirected graphs in 70

Terminology

For an edge $e = \{u, v\}$:

- u and v are the *endpoints* of e
- e is *incident* on u and v

Terminology

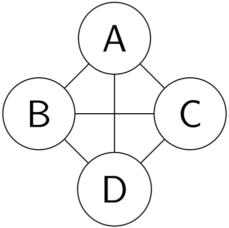
For an edge $e = \{u, v\}$:

- u and v are the *endpoints* of e
- e is *incident* on u and v

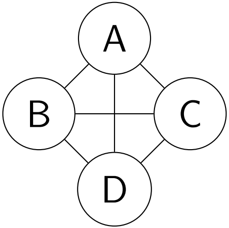
For a vertex v:

- number of edges incident is the *degree*
- u is a *neighbor* (or is *adjacent*) if $\{u, v\} \in E$
- if no neighbors, v is *isolated*

Paths and Cycles and Tours, Oh My!

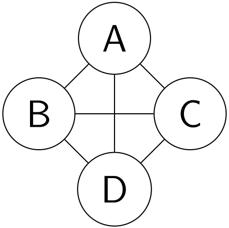


Paths and Cycles and Tours, Oh My!



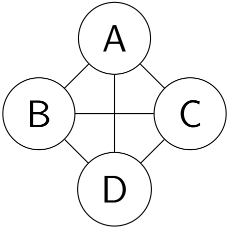
A *walk* is a sequence of (connected) edges

Paths and Cycles and Tours, Oh My!



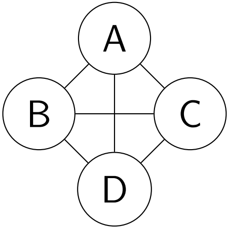
A *walk* is a sequence of (connected) edges
A *tour* is a “closed” walk

Paths and Cycles and Tours, Oh My!



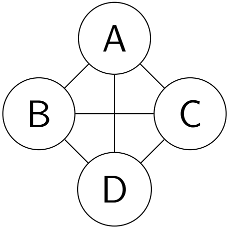
A *walk* is a sequence of (connected) edges
A *tour* is a "closed" walk
A *simple path* is a walk with no repeated vertices

Paths and Cycles and Tours, Oh My!



A *walk* is a sequence of (connected) edges
A *tour* is a “closed” walk
A *simple path* is a walk with no repeated vertices
A *cycle* is a “closed” path

Paths and Cycles and Tours, Oh My!



A *walk* is a sequence of (connected) edges
A *tour* is a "closed" walk
A *simple path* is a walk with no repeated vertices
A *cycle* is a "closed" path

Cycle $\subseteq$ tour $\subseteq$ walk; path $\subseteq$ walk.

Eulerian Tours

An *Eulerian Tour* is a tour using each edge once

Eulerian Tours

An *Eulerian Tour* is a tour using each edge once

Undirected graph is *connected* if $\exists$ a path between any two vertices

Eulerian Tours

An *Eulerian Tour* is a tour using each edge once

Undirected graph is *connected* if $\exists$ a path between any two vertices

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Only If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Proof (only if):

- Suppose G has an Eulerian Tour

Only If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Proof (only if):

- Suppose G has an Eulerian Tour
- Uses two edges when passing through a vertex
- So number of edges incident to v used is even

Only If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Proof (only if):

- Suppose G has an Eulerian Tour
- Uses two edges when passing through a vertex
- So number of edges incident to v used is even
- But all incident edges used!
- Thus, degree of v is even

If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Proof (if):

- Suppose all degrees are even

If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Proof (if):

- Suppose all degrees are even
- Follow arbitrary edges until stuck
- All degrees even means stuck at start vertex

If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

Proof (if):

- Suppose all degrees are even
- Follow arbitrary edges until stuck
- All degrees even means stuck at start vertex
- Remove this tour, recurse on *connected components*

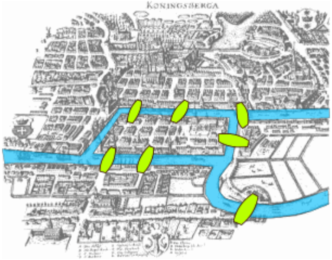
If Direction

Theorem: Let G be a connected graph. G has an Eulerian Tour iff every vertex has even degree.

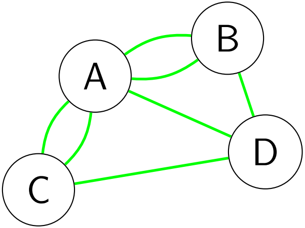
Proof (if):

- Suppose all degrees are even
- Follow arbitrary edges until stuck
- All degrees even means stuck at start vertex
- Remove this tour, recurse on *connected components*
- "Splice" the recursive tours into the main one
- Result is Eulerian Tour of G!

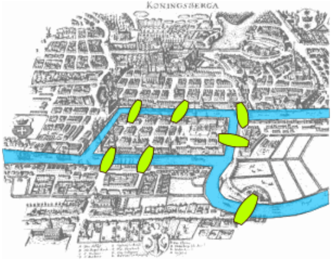
Back To Königsberg



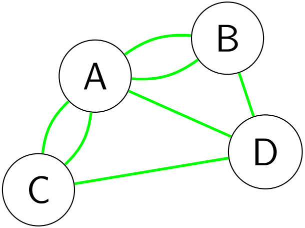
$\equiv$



Back To Königsberg

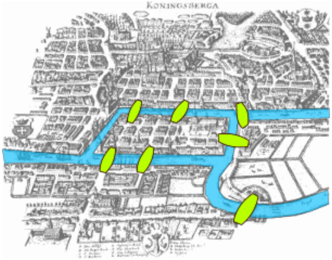


$\equiv$

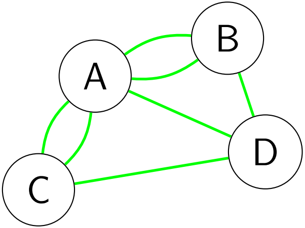


Model Königsberg as a graph
"Cross each bridge once" $\equiv$ "Eulerian Tour"

Back To Königsberg



$\equiv$



Model Königsberg as a graph
"Cross each bridge once" $\equiv$ "Eulerian Tour"

Note: This is a *multigraph*
Everywhere else, assume *simple graphs*

- No repeated edges
- No self-loops

if(tired) { break; }

Whew, that was a long proof. Time for a break.

if(tired) { break; }

Whew, that was a long proof. Time for a break.

Today's Discussion Question:
What building on campus would you get rid of?

Special Types of Graphs

Complete graphs have every possible edge

- Denote complete graph on n vertices as K_n
- K_5 has an important role next lecture!

Special Types of Graphs

Complete graphs have every possible edge

- Denote complete graph on n vertices as K_n
- K_5 has an important role next lecture!

Bipartite graphs have two halves, often denoted L, R

- Edges can only go between L and R

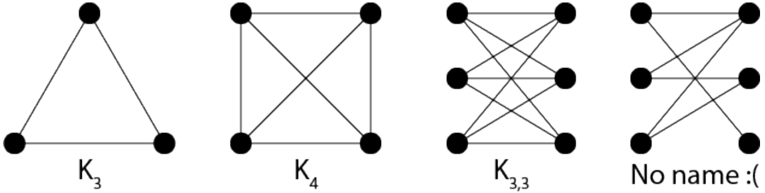
Special Types of Graphs

Complete graphs have every possible edge

- Denote complete graph on n vertices as K_n
- K_5 has an important role next lecture!

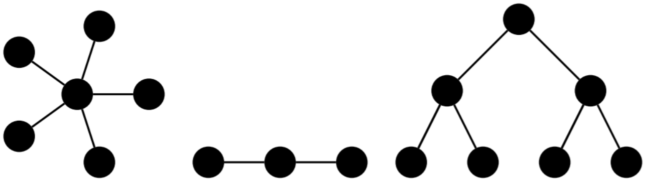
Bipartite graphs have two halves, often denoted L, R

- Edges can only go between L and R



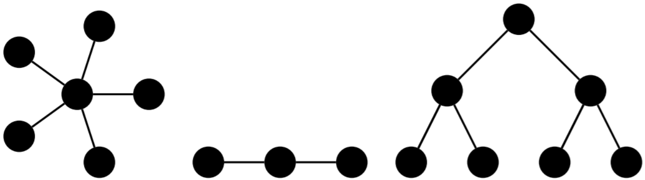
Can't See the Forest For All the Trees

A *tree* is a connected, acyclic graph



Can't See the Forest For All the Trees

A *tree* is a connected, acyclic graph



Equivalent definitions:

- Connected and $|V| - 1$ edges
- Connected and any edge removal disconnects
- Acyclic and any edge addition creates cycle

Leaf Lemmas

A *leaf* is a vertex of degree 1

Lemma: Every tree has at least one leaf.

Leaf Lemmas

A *leaf* is a vertex of degree 1

Lemma: Every tree has at least one leaf.

Proof:

- Consider longest (simple) path in tree
- v is vertex at beginning

Leaf Lemmas

A *leaf* is a vertex of degree 1

Lemma: Every tree has at least one leaf.

Proof:

- Consider longest (simple) path in tree
- v is vertex at beginning
- v only connected to next vertex in path

Leaf Lemmas

A *leaf* is a vertex of degree 1

Lemma: Every tree has at least one leaf.

Proof:

- Consider longest (simple) path in tree
- v is vertex at beginning
- v only connected to next vertex in path

Lemma: A tree minus a leaf is still a tree.

Leaf Lemmas

A *leaf* is a vertex of degree 1

Lemma: Every tree has at least one leaf.

Proof:

- Consider longest (simple) path in tree
- v is vertex at beginning
- v only connected to next vertex in path

Lemma: A tree minus a leaf is still a tree.

Proof:

- Can't create a cycle by removing
- No path through leaf, so can't disconnect

Leaf Lemmas

A *leaf* is a vertex of degree 1

Lemma: Every tree has at least one leaf.

Proof:

- Consider longest (simple) path in tree
- v is vertex at beginning
- v only connected to next vertex in path

Lemma: A tree minus a leaf is still a tree.

Proof:

- Can't create a cycle by removing
- No path through leaf, so can't disconnect

Allows us to do induction on trees!

Definitions, Definitions

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Definitions, Definitions

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\implies$):

- Induct on $|V|$. Base case easy.

Definitions, Definitions

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\implies$):

- Induct on $|V|$. Base case easy.
- Remove a leaf and its incident edge

Definitions, Definitions

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\implies$):

- Induct on $|V|$. Base case easy.
- Remove a leaf and its incident edge
- By IH, result has $|V| - 2$ edges
- So original graph had $|V| - 1$ edges

Definitions, Definitions 2

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\Longleftarrow$):

- Induct on $|V|$. Base case easy.

Definitions, Definitions 2

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\Longleftarrow$):

- Induct on $|V|$. Base case easy.
- Total degree is $2|E| = 2|V| - 2$
- Some vertex v must have degree 1

Definitions, Definitions 2

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\Longleftarrow$):

- Induct on $|V|$. Base case easy.
- Total degree is $2|E| = 2|V| - 2$
- Some vertex v must have degree 1
- Remove v and its edge

Definitions, Definitions 2

Theorem: T connected and acyclic $\iff$ T connected and has $|V| - 1$ edges

Proof ($\Longleftarrow$):

- Induct on $|V|$. Base case easy.
- Total degree is $2|E| = 2|V| - 2$
- Some vertex v must have degree 1
- Remove v and its edge
- By IH, result is acyclic
- Adding v back can't create a cycle or disconnect!

A Bad Proof

Claim: Every graph with a leaf is a tree.

A Bad Proof

Claim: Every graph with a leaf is a tree.

“Proof”:

- Induct on $|V|$. Base case easy.

A Bad Proof

Claim: Every graph with a leaf is a tree.

"Proof":

- Induct on $|V|$. Base case easy.
- Suppose true for graphs with k vertices
- Create graph on $k + 1$ vertices by adding a leaf

A Bad Proof

Claim: Every graph with a leaf is a tree.

“Proof”:

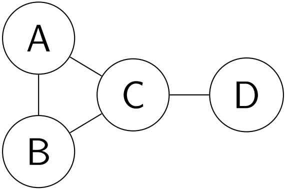
- Induct on $|V|$. Base case easy.
- Suppose true for graphs with k vertices
- Create graph on $k+1$ vertices by adding a leaf
- Doesn't create a cycle or disconnect
- So size $k+1$ graph also a tree!

A Bad Proof

Claim: Every graph with a leaf is a tree.

"Proof":

- Induct on $|V|$. Base case easy.
- Suppose true for graphs with k vertices
- Create graph on $k+1$ vertices by adding a leaf
- Doesn't create a cycle or disconnect
- So size $k+1$ graph also a tree!



Build Up Error

Last slide was an example of *build up error*

Build Up Error

Last slide was an example of *build up error*

Assumed we could *build up* bigger graph from smaller graph in some specific way, but can't

Build Up Error

Last slide was an example of *build up error*

Assumed we could *build up* bigger graph from smaller graph in some specific way, but can't

Avoid using "shrink down, grow back" method

- Start with big graph, shrink to smaller
- Apply IH to small graph
- Add back what was removed

Build Up Error

Last slide was an example of *build up error*

Assumed we could *build up* bigger graph from smaller graph in some specific way, but can't

Avoid using "shrink down, grow back" method

- Start with big graph, shrink to smaller
- Apply IH to small graph
- Add back what was removed

See what happens if we use this in our "proof"

Shrink Down, Grow Back Example

Claim: Every graph is a tree.

"Proof":

- Induct on $|V|$. Base case easy.

Shrink Down, Grow Back Example

Claim: Every graph is a tree.

"Proof":

- Induct on $|V|$. Base case easy.
- Start with graph on $k + 1$ vertices
- Remove a leaf to get a k vertex graph

Shrink Down, Grow Back Example

Claim: Every graph is a tree.

“Proof”:

- Induct on $|V|$. Base case easy.
- Start with graph on $k+1$ vertices
- Remove a leaf to get a k vertex graph
- ...wait

Shrink Down, Grow Back Example

Claim: Every graph is a tree.

"Proof":

- Induct on $|V|$. Base case easy.
- Start with graph on $k+1$ vertices
- Remove a leaf to get a k vertex graph
- ...wait
- How do we know there's still a leaf?

Shrink Down, Grow Back Example

Claim: Every graph is a tree.

"Proof":

- Induct on $|V|$. Base case easy.
- Start with graph on $k+1$ vertices
- Remove a leaf to get a k vertex graph
- ...wait
- How do we know there's still a leaf?

We're stuck!
And should be–the theorem is false

Fin

Next time: moar graphs!