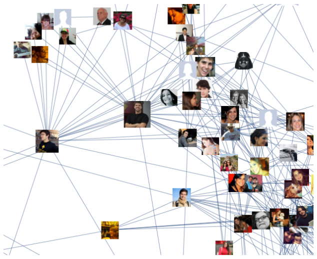
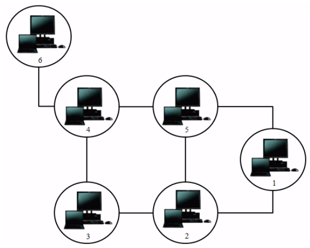
15-251
Great Ideas in Theoretical Computer Science

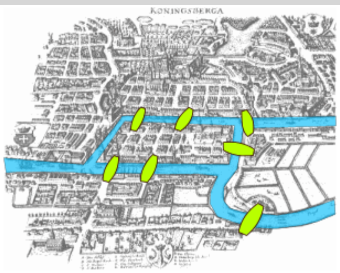
Lecture 12:
Graphs I: The Basics





February 22nd, 2018

Crossing bridges



Königsberg (Prussia)

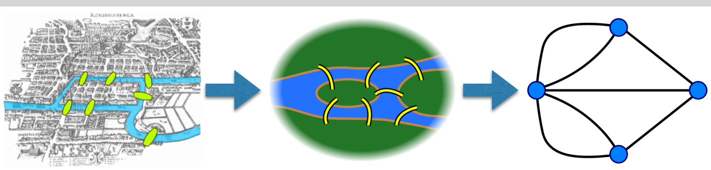
Now
Kaliningrad (Russia)

Is there a way to walk through the city that would cross each bridge **exactly** once?

Leonhard Euler
(1735)

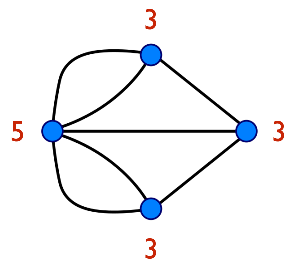
This is not possible!

Crossing bridges



Except for the start and end vertices:

Crossing bridges



Every vertex is incident to an odd number of edges.

So this graph does not have an "*Eulerian tour*".

Crossing bridges

What if it *is* the case that exactly 0 or 2 nodes are incident to an odd number of edges?

Does that imply the graph must have an Eulerian tour?

Why graphs?

Why now?

Some examples where graphs appear

Computer Science Life Lesson

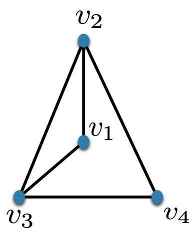
If your problem has a graph, 😃 👍 .

If not, try to make it have a graph.

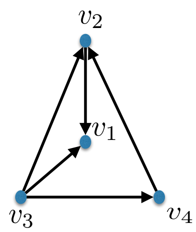
What is a graph?

(A hundred) definitions and basic properties

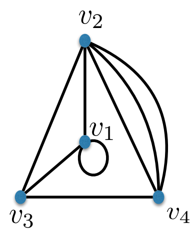
Types of Graphs



Simple
Undirected
Graph



Directed Graph



Multigraph

Formal Definition: (undirected) graph

A **graph** G is a tuple (V, E), where

Example:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_5, v_6\}\}$$

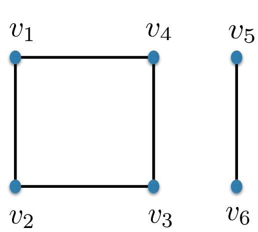
Formal Definition: (undirected) graph

Example:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_5, v_6\}\}$$

Graphs can be drawn:



Formal Definition: (undirected) graph

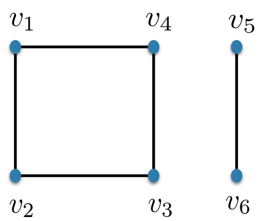
Example:

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_5, v_6\}\}$

Matrix representation
(adjacency matrix):

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



IMPORTANT Notation

Almost always:

n = number of vertices in the graph, $|V|$

m = number of edges, $|E|$

1st Challenge

Is it possible to have a party with 251 people in which everyone knows exactly 5 other people in the party?

Is it possible to have a graph with 251 vertices in which each vertex is adjacent to exactly 5 other vertices?

Terminology: Neighbor

Suppose $e = \{u, v\} \in E$ is an edge.

We say:

u and v are ____________ of e

u and v are ____________ .

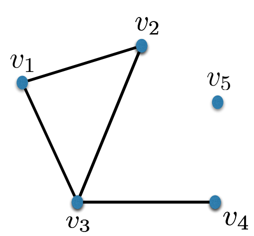
u and v are __________ on e

u is a __________ of v

v is a __________ of u

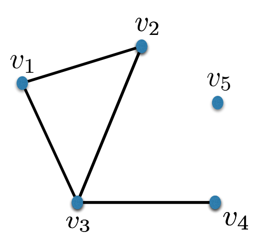
Terminology: Neighborhood

For $v \in V$, the **neighborhood** of v is defined as



Terminology: Degree

For $v \in V$, the **degree** of v is defined as



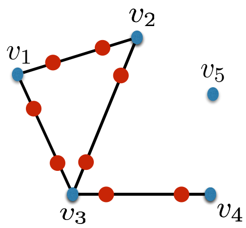
A graph is called **d-regular** if

1st Theorem

Theorem: Let $G = (V, E)$ be a graph. Then

$$\sum_{v \in V} \deg(v) = 2m.$$

Proof:



Poll

Is it possible to have a graph with 251 vertices in which each vertex is adjacent to exactly 5 other vertices?

Yes

No

Beats me

2nd Challenge

We have n computers that we want to connect.

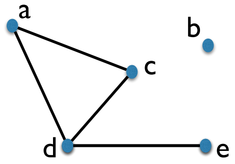
We can put a link between any two computers, but the links are expensive.

What is the least number of links we can use?

What is the least number of edges needed to connect n vertices?

Walks and Paths

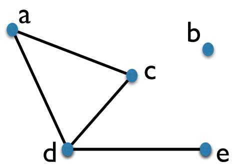
A **walk** in a graph G = (V, E) is

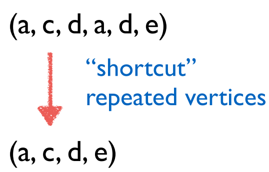


Walks and Paths

A **path** in a graph G = (V, E) is

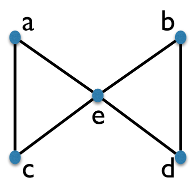
Fact: There is a path from u to v iff
there is a walk from u to v





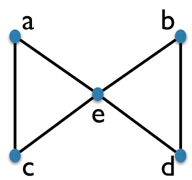
Circuits and Cycles

A **circuit** in a graph G = (V, E) is



Circuits and Cycles

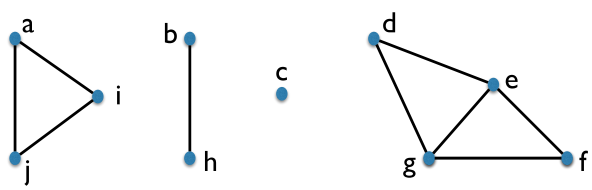
A **cycle** in a graph G = (V, E) is



A graph with no cycles is called __________.

Connected Graphs

A graph is **connected** if



This 10-vertex graph is **not** connected.

It has 4 **connected components**:

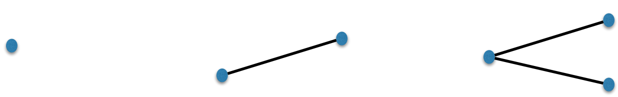
{a, i, j}, {b, h}, {c}, {d, e, f, g}

A graph is connected iff

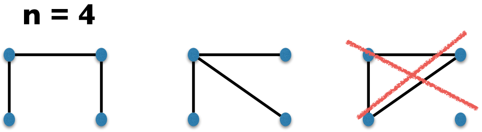
Back to the challenge

What is the least number of edges needed to connect n vertices?

n = 1 **n = 2** **n = 3**



n = 4



Back to the challenge

What is the least number of edges needed to connect n vertices?

Poll

Are n-1 edges always necassary to connect n vertices?

Yes

No

No opinion

2nd Theorem

Theorem: Let $G = (V, E)$ be a **connected** graph.

Then $m \geq n - 1$.

Furthermore,

$$m = n - 1 \iff G \text{ is acyclic.}$$

Proof:

Imagine the following process:

- remove all the edges of G.
- add them back one by one (in an arbitrary order).

n isolated vertices → G

n CCs → **1** CC

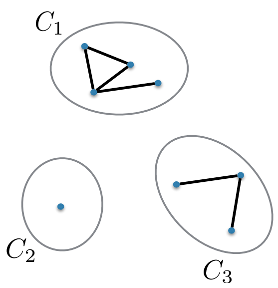
CC = connected component

2nd Theorem

Proof (continued):

Consider a step of adding an edge back.

2 possibilities:

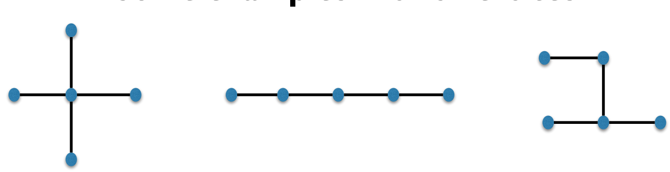


2nd Theorem

Proof (continued):

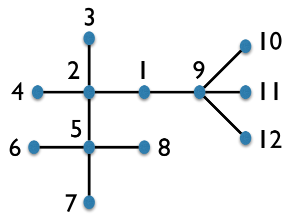
Trees

Some examples with 5 vertices



Definition:
An n-vertex **tree** is any graph with at least 2 of the following 3 properties:

Trees



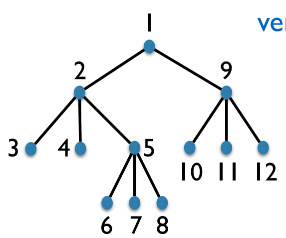
Leaf:

Internal node:

Rooted tree:

Trees

vertex 1 is the root



For rooted trees, we use "*family tree*" terminology:

- parent
- child
- sibling

- ancestor
- descendant

etc…

Binary tree:
- rooted tree
- each node has at most 2 children.

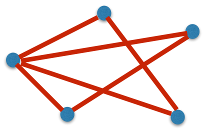
Back to Köningsberg's Bridges

Eulerian circuit

Eulerian Circuit Problem

Input: a graph G = (V, E)

Output: Yes if there is a circuit visiting each edge exactly once. No otherwise.



Eulerian circuit

Eulerian Circuit Problem

Input: a graph G = (V, E)

Output: Yes if there is a circuit visiting each edge exactly once. No otherwise.

Euler claimed (but did not provide a proof):

A connected graph has an Eulerian circuit **iff** deg(v) is even for all v.

proved by Hierholzer

Efficient algorithm:

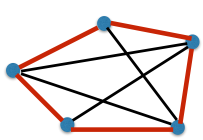
- Check that the graph is connected.
- Check that every vertex has even degree.

Hamiltonian cycle

Hamiltonian Cycle Problem

Input: a graph G = (V, E)

Output: Yes if there is a cycle visiting each **vertex** exactly once. No otherwise.



Hamiltonian cycle

Hamiltonian Cycle Problem

Input: a graph G = (V, E)

Output: Yes if there is a cycle visiting each **vertex** exactly once. No otherwise.

Brute-Force Algorithm:

- Try all cycles $O(n!)$

Dynamic Programming Algorithm: $O(2^n)$

Clever Algebraic Brute-Force: $O(1.657^n)$

Anything better?