Graphs

Graphs and trees come up everywhere.

- We can view the internet as a graph (in many ways)
  - who is connected to whom
- Web search views web pages as a graph
  - Who points to whom
- Niche graphs (Ecology):
  - The vertices are species
  - Two vertices are connected by an edge if they compete (use the same food resources, etc.)

  Niche graphs describe competitiveness.
- Influence Graphs
  - The vertices are people
  - There is an edge from a to b if a influences b

  Influence graphs give a visual representation of power structure.
- Social networks: who knows whom

There are lots of other examples in all fields . . .

By viewing a situation as a graph, we can apply general results of graph theory to gain a deeper understanding of what's going on.

We're going to prove some of those general results. But first we need some notation . . .

Directed and Undirected Graphs

A *graph* G is a pair (V, E), where V is a set of *vertices* or *nodes* and E is a set of *edges*

- We sometimes write $G(V, E)$ instead of G
- We sometimes write $V(G)$ and $E(G)$ to denote the vertices/edges of G.

In a *directed graph (digraph)*, the edges are ordered pairs $(u, v) \in V \times V$:

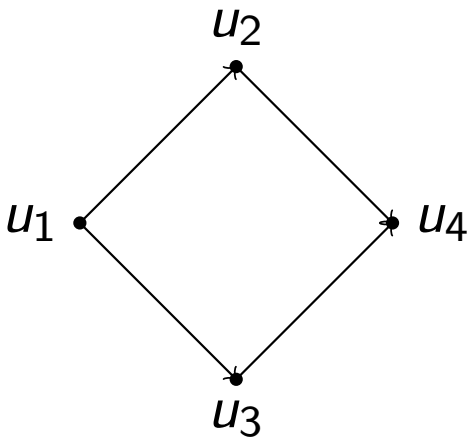
- the edge goes from node u to node v

In an *undirected simple graphs*, the edges are sets $\{u, v\}$ consisting of two nodes.

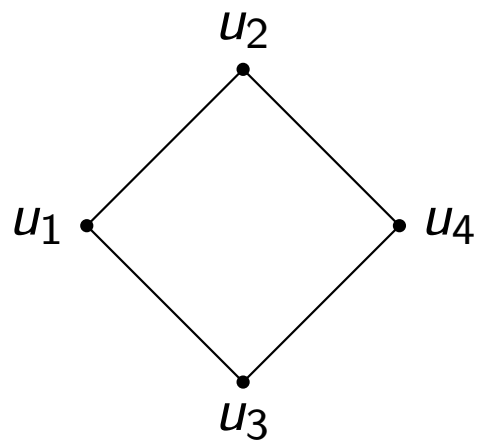
- there's no direction in the edge from u to v
- More general (non-simple) undirected graphs (sometimes called multigraphs) allow self-loops and multiple edges between two nodes.
  - There may be several nodes between two towns

Representing graphs

We usually represent a graph pictorially.



Directed graph

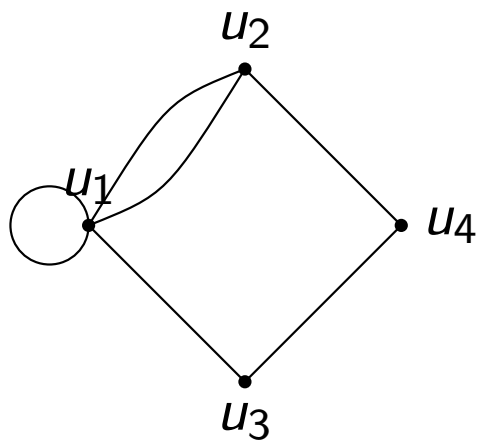


(Undirected) simple graph

- The directed graph on the left has edges $(u_1, u_2), (u_1, u_3), (u_2, u_4), (u_3, u_4)$
  - Note the arrows on the edges
  - The order matters!
- The undirected graph has edges $\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_4\}, \{u_3, u_4\}$
  - The order doesn't matter: $\{u_1, u_2\} = \{u_2, u_1\}$.

Non-simple undirected graphs

A non-simple undirected graph, with a self loop and multiple edges between nodes:



In this course, we'll focus on directed graphs and undirected simple graphs.

- Lots of the general results for simple graphs actually hold for general undirected graphs, if you define things right

Directed vs. Undirected Graphs

Is the following better represented as (a) a directed graph or (b) an undirected graph:

1. Social network (edge between u and v if u and v are friends)

Directed vs. Undirected Graphs

Is the following better represented as (a) a directed graph or (b) an undirected graph:

1. Social network (edge between u and v if u and v are friends)
2. Niche graph (edge between species u and v if they compete)

Directed vs. Undirected Graphs

Is the following better represented as (a) a directed graph or (b) an undirected graph:

1. Social network (edge between u and v if u and v are friends)
2. Niche graph (edge between species u and v if they compete)
3. influence graph (edge between u and v if u points to v)

Directed vs. Undirected Graphs

Is the following better represented as (a) a directed graph or (b) an undirected graph:

1. Social network (edge between u and v if u and v are friends)
2. Niche graph (edge between species u and v if they compete)
3. influence graph (edge between u and v if u points to v)
4. communication network (edge between u and v if u and v are connected by a communication link)

Degree

In a directed graph $G(V, E)$, the *indegree* of a vertex v is the number of edges coming into it

- $\text{indegree}(v) = |\{v' : (v', v) \in E\}|$

The *outdegree* of v is the number of edges going out of it:

- $\text{outdegree}(v) = |\{v' : (v, v') \in E\}|$

The *degree* of v, denoted $\deg(v)$, is the sum of the indegree and outdegree.

For an undirected simple graph, it doesn't make sense to talk about indegree and outdegree. The degree of a vertex is the sum of the edges incident to the vertex.

- edge e is *incident to* v if v is one of the endpoints of e

Theorem: Given a graph $G(V, E)$,

$$2|E| = \sum_{v \in V} \deg(v)$$

Proof: For a directed graph: each edge contributes once to the indegree of some vertex, and once to the outdegree of some vertex. Thus $|E|$ = sum of the indegrees = sum of the outdegrees.

Same argument for a simple undirected graph.

- This also works for a general undirected graph, if you double-count self-loops

Handshaking Theorem

Theorem: The number of people who shake hands with an odd number of people at a party must be even.

Proof: Construct a graph, whose vertices are people at the party, with an edge between two people if they shake hands. The number of people person p shakes hands with is $\deg(p)$. Split the set of all people at the party into two subsets:

- $A =$ those that shake hands with an even number of people
- $B =$ those that shake hands with an odd number of people

Handshaking Theorem

Theorem: The number of people who shake hands with an odd number of people at a party must be even.

Proof: Construct a graph, whose vertices are people at the party, with an edge between two people if they shake hands. The number of people person p shakes hands with is $\deg(p)$. Split the set of all people at the party into two subsets:

- $A =$ those that shake hands with an even number of people
- $B=$ those that shake hands with an odd number of people

$$\sum_{p} \deg(p) = \sum_{p \in A} \deg(p) + \sum_{p \in B} \deg(p)$$

- We know that $\sum_p \deg(p) = 2|E|$ is even.
- $\sum_{p \in A} \deg(p)$ is even, because for each $p \in A$, $\deg(p)$ is even.
- Therefore, $\sum_{p \in B} \deg(p)$ is even.
- Therefore $|B|$ is even (because for each $p \in B$, $\deg(p)$ is odd, and if $|B|$ were odd, then $\sum_{p \in B} \deg(p)$ would be odd).

Graph Isomorphism

When are two graphs that may look different when they're drawn, really the same?

Answer: $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are *isomorphic* if they have the same number of vertices ($|V_1| = |V_2|$) and we can relabel the vertices in G_2 so that the edge sets are identical.

- Formally, G_1 is isomorphic to G_2 if there is a bijection $f : V_1 \to V_2$ such that $\{v, v'\} \in E_1$ iff $(\{f(v), f(v')\} \in E_2$.
- Note this means that $|E_1| = |E_2|$

Checking for Graph Isomorphism

There are some obvious requirements for $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ to be isomorphic:

- $|V_1| = |V_2|$
- $|E_1| = |E_2|$
- for each d, #(vertices in V_1 with degree d) = #(vertices in V_1 with degree d)

Checking for isomorphism is in NP:

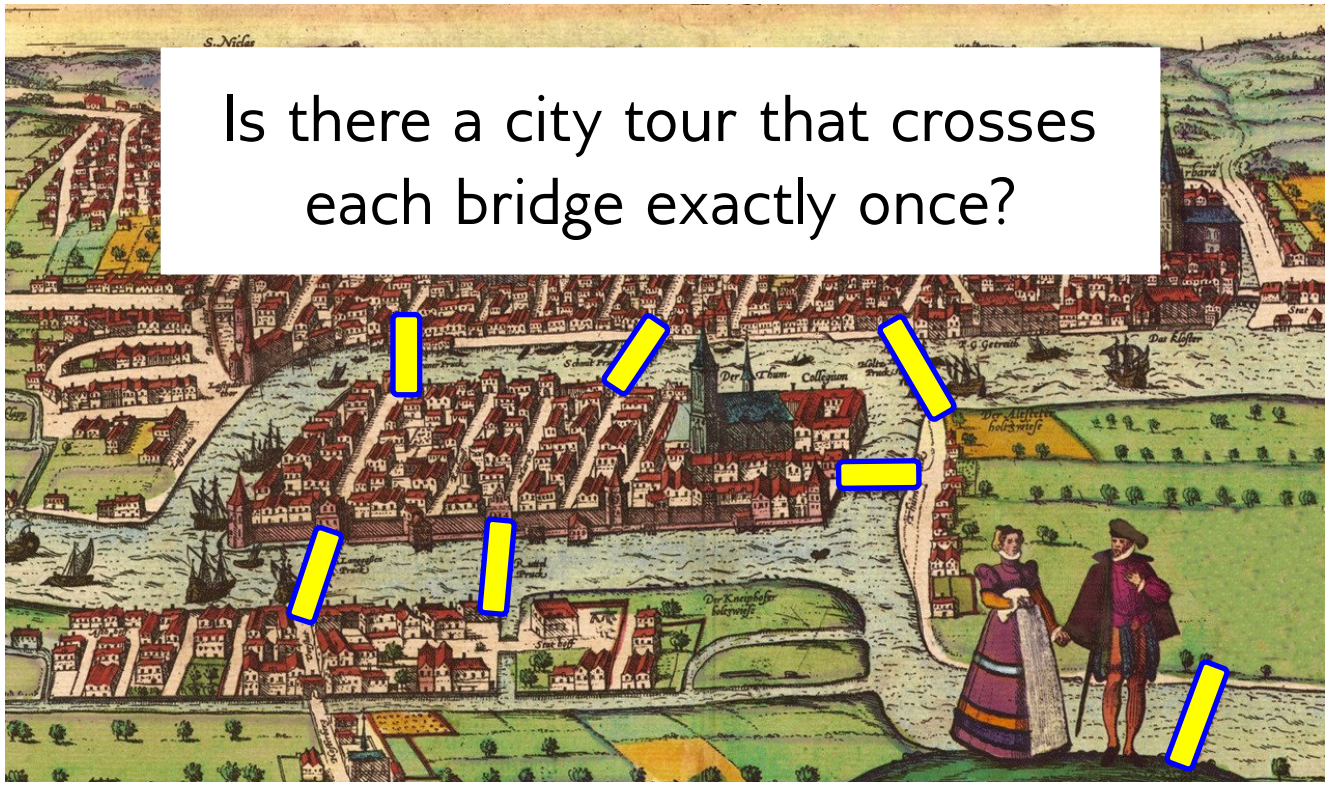
- Guess an isomorphism f and verify
- We believe it's not in polynomial time and not NP complete.

Paths

Given a graph $G(V, E)$.

- A *path* in G is a sequence of vertices $(v_0, \ldots, v_n)$ such that $\{v_i, v_{i+1}\} \in E$ ((v_i, v_{i+1}) in the directed case).
  - a simple path has no repeated vertices
    - MCS calls a path a *walk* and a simple path a *path*
  - vertex v is *reachable* from u if there is a path from u to v
- If $v_0 = v_n$, the path is a *cycle*
- An *Eulerian* path/cycle is a path/cycle that traverses every every edge in E exactly once
- A *Hamiltonian* path/cycle is a path/cycle that passes through each vertex in V exactly once.
- A graph with no cycles is said to be *acyclic*
- An undirected graph is *connected* if every vertex is reachable from every other vertex.

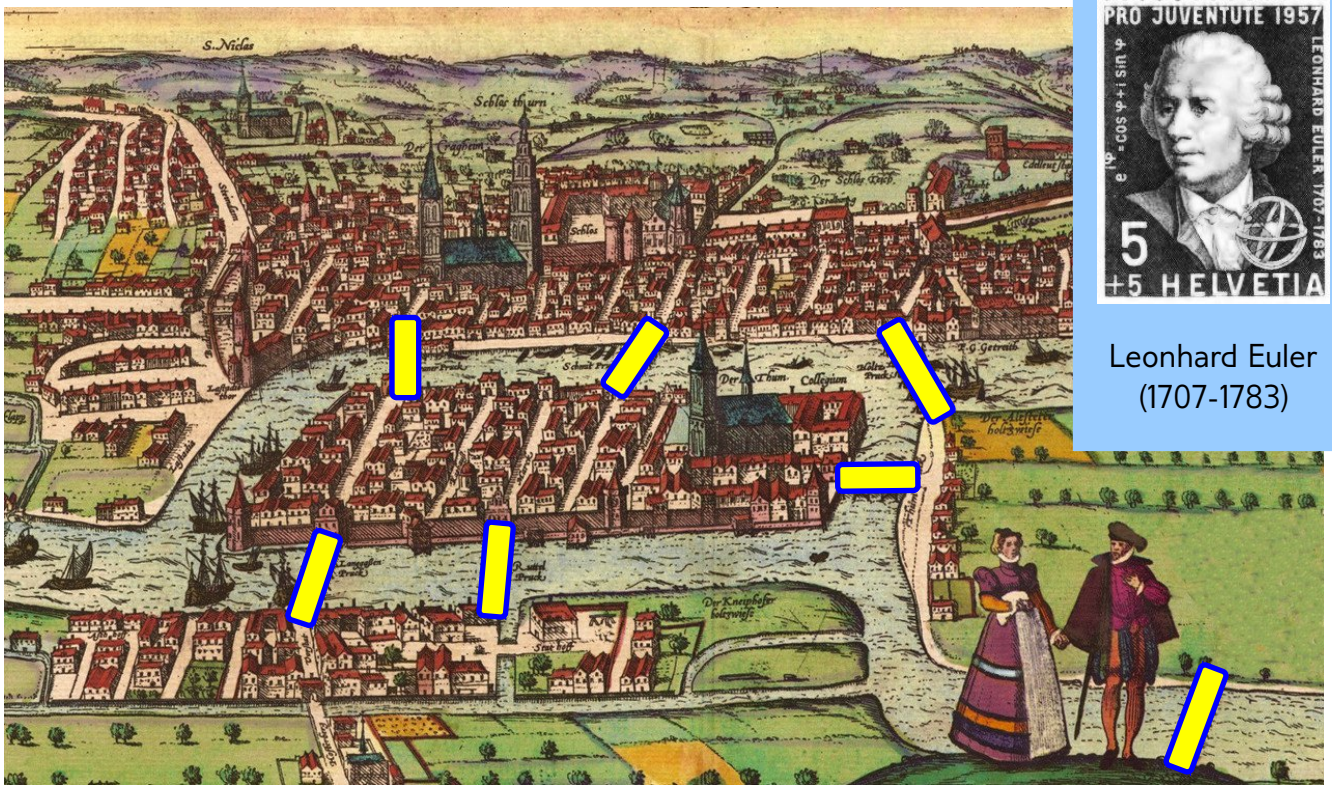
Bridges of Königsberg



Braun & Hogenberg, "Civitates Orbis Terrarum", Cologne 1585. Photoshopped to clean up right side and add 7th bridge.

Remember this from the first class?

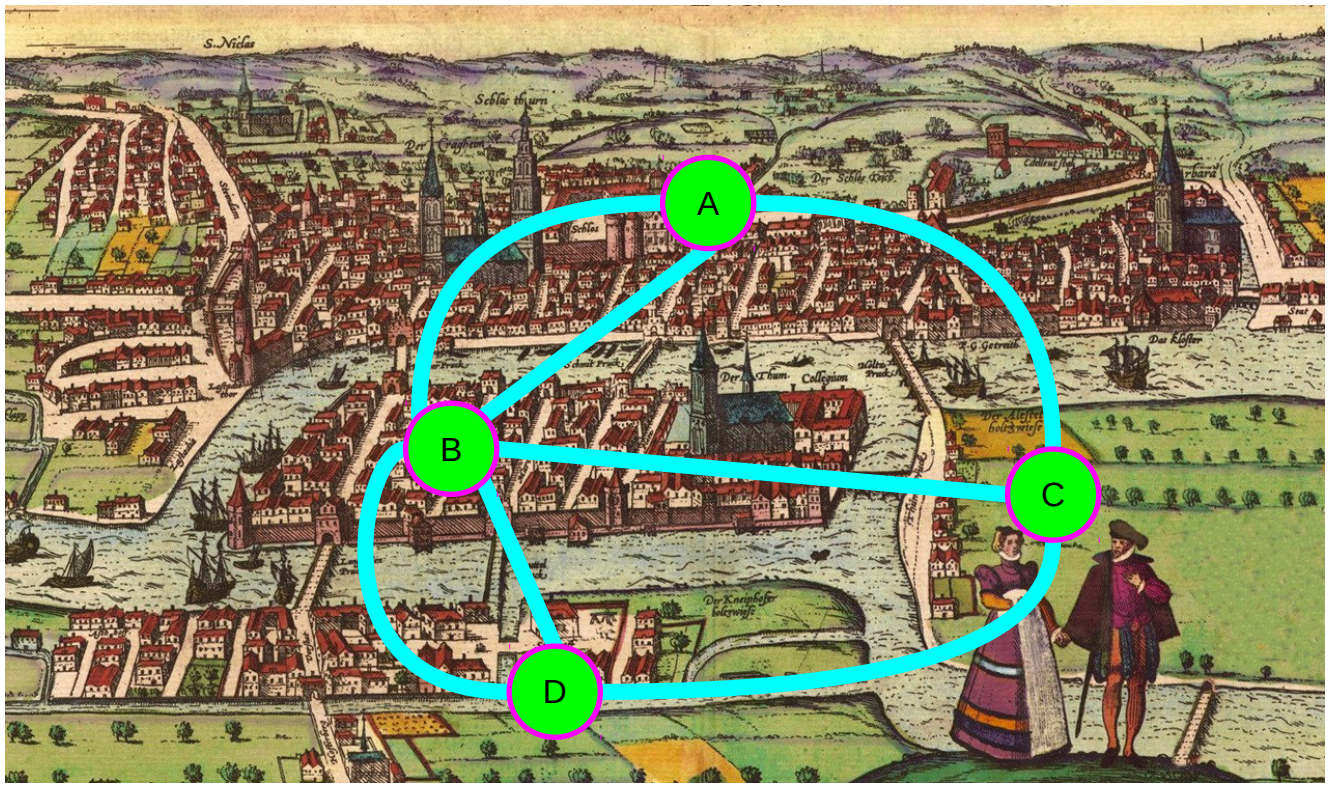
Bridges of Königsberg



Leonhard Euler
(1707-1783)

Braun & Hogenberg, "Civitates Orbis Terrarum", Cologne 1585. Photoshopped to clean up right side and add 7th bridge.

Bridges of Königsberg



Braun & Hogenberg, "Civitates Orbis Terrarum", Cologne 1585. Photoshopped to clean up right side and add 7th bridge.

Euler's key insight: represent the problem as a graph

Eulerian Paths

Recall that $G(V, E)$ has an Eulerian path if it has a path that goes through every edge exactly once. It has an Eulerian cycle (or Eulerian circuit) if it has an Eulerian path that starts and ends at the same vertex.

How can we tell if a graph has an Eulerian path/circuit?

What's a necessary condition for a graph to have an Eulerian circuit?
Count the edges going into and out of each vertex:

- Each vertex must have even degree!

This condition turns out to be sufficient too.

Theorem: A connected (multi)graph has an Eulerian cycle iff each vertex has even degree.

Proof: The necessity is clear: In the Eulerian cycle, there must be an even number of edges that start or end with any vertex.

To see the condition is sufficient, we provide an algorithm for finding an Eulerian circuit in $G(V, E)$.

Theorem: A connected (multi)graph has an Eulerian cycle iff each vertex has even degree.

Proof: The necessity is clear: In the Eulerian cycle, there must be an even number of edges that start or end with any vertex.

To see the condition is sufficient, we provide an algorithm for finding an Eulerian circuit in $G(V, E)$.

First step: Follow your nose to construct a cycle.

Second step: Remove the edges in the cycle from G. Let H be the subgraph that remains.

- every vertex in H has even degree
- H may not be connected; let $H_1, \ldots, H_k$ be its connected components.

Third step: Apply the algorithm recursively to $H_1, \ldots, H_k$, and then splice the pieces together.

Finding cycles

First, find an algorithm for finding a cycle:

Input: $G(V, E)$ [a list of vertices and edges]
procedure Pathgrow(V,E,v)
[v is first vertex in cycle]
$P \leftarrow ()$ [P is sequence of edges on cycle]
$w \leftarrow v$ [w is last vertex in P]
repeat until $I(w) - P = \emptyset$
[$I(w)$ is the set of edges incident on w]
Pick $e \in I(w) - P$
$w \leftarrow$ other end of e
$P \leftarrow P \cdot e$ [append e to P]
return P

Finding cycles

First, find an algorithm for finding a cycle:

Input: $G(V, E)$ [a list of vertices and edges]
procedure Pathgrow(V,E,v)
[v is first vertex in cycle]
$P \leftarrow ()$ [P is sequence of edges on cycle]
$w \leftarrow v$ [w is last vertex in P]
repeat until $I(w) - P = \emptyset$
[$I(w)$ is the set of edges incident on w]
Pick $e \in I(w) - P$
$w \leftarrow$ other end of e
$P \leftarrow P \cdot e$ [append e to P]
return P

Claim: If every vertex in V has even degree, then P will be a cycle

- Loop invariant: In the graph $G(V, E - P)$, if the first vertex (v) and last vertex (w) in P are different, they have odd degree; all the other vertices have even degree.

Finding Eulerian Paths

procedure Euler(V,E,v)
    //$G(V, E)$ is a connected undirected graph//
    //$v \in V$ is arbitrary//
    //output is an Eulerian cycle in G//
Pathgrow(V',E',v') [returns cycle P in G]
if P is not Eulerian
then delete the edges in P from E;
    let $G_1(V_1, E_1), \ldots, G_n(V_n, E_n)$ be
        the resulting connected components
    let v_i be a vertex in V_i also on P
    for $i = 1$ **to** n
        Euler(V_i, E_i, v_i) [returns Eulerian cycle C_i
        Attach C_i to P at v_i
    endfor
return P

Corollary: A connected multigraph has an Eulerian path (but not an Eulerian cycle) if it has exactly two vertices of odd degree.

Hamiltonian Paths

Recall that $G(V, E)$ has a Hamiltonian path if it has a path that goes through every vertex exactly once. It has a Hamiltonian cycle (or Hamiltonian circuit) if it has a Hamiltonian path that starts and ends at the same vertex.

There is no known easy characterization or algorithm to check if a graph has a Hamiltonian cycle/path.

Graphs and Scheduling

In a scheduling problem, there are tasks and constraints specifying that some tasks have to be completed before others.

- This can be represented graphically
- The nodes are the tasks
- There is an edge from u to v if u has to be completed before v is started
- Sometimes the edges have *weights* (how much time it takes to complete the task)

The result is a *directed acyclic graph (dag)*.

- If there's a cycle, the job can't be completed!

This way of representing task scheduling gives us lots of useful information:

- the length of the longest path in the graph gives us a lower bound on how long the task takes (even if we could throw an unbounded number of people at the tasks, and they can work in parallel).
- It can take longer if we have a bounded number of workers

Topological Sorts

If there's only one person doing the tasks, in what order should we do the tasks in a scheduling problem?

A *topological sort* of a dag $G = (V, E)$ is a list of all the vertices in V such that if v is reachable from u, then u precedes v on the list.

- A topological sort of a dag representing tasks gives a possible order to do the tasks
- There may be more than one topological sort of a dag

Theorem: Every dag $G = (V, E)$ has a topological sort.

How can we prove this formally?

Topological Sorts

If there's only one person doing the tasks, in what order should we do the tasks in a scheduling problem?

A *topological sort* of a dag $G = (V, E)$ is a list of all the vertices in V such that if v is reachable from u, then u precedes v on the list.

- A topological sort of a dag representing tasks gives a possible order to do the tasks
- There may be more than one topological sort of a dag

Theorem: Every dag $G = (V, E)$ has a topological sort.

How can we prove this formally?

- By induction on the number of vertices in V
  - Remove a node that has no edges going out of it.
  - How do you know there is one?

Graph Coloring

How many colors do you need to color the vertices of a graph so that no two adjacent vertices have the same color?

- Application: scheduling
  - Vertices of the graph are courses
  - Two courses taught by same prof are joined by edge
  - Colors are possible times class can be taught.

Lots of similar applications:

- E.g. assigning wavelengths to cell phone conversations to avoid interference.
  - Vertices are conversations
  - Edges between "nearby" conversations
  - Colors are wavelengths.
- Scheduling final exams
  - Vertices are courses
  - Edges between courses with overlapping enrollment
  - Colors are exam times.

Chromatic Number

The *chromatic number* of a graph G, written $\chi(G)$, is the smallest number of colors needed to color it so that no two adjacent vertices have the same color.

A graph G is *k-colorable* if $k \geq \chi(G)$.

Determining $\chi(G)$

Some observations:

- If G is a complete graph with n vertices, $\chi(G) = n$
  - A complete graph is one where there is an edge between every pair of nodes.
  - How many edges are there in a complete graph with n nodes?
- If G has a clique of size k, then $\chi(G) \geq k$.
  - A *clique* of size k in a graph G is a completely connected subgraph of G with k vertices.
  - Let $c(G)$ be the *clique number* of G: the size of the largest clique in G. Then
$$\chi(G) \geq c(G).$$
- If $\Delta(G)$ is the maximum degree of any vertex, then
$$\chi(G) \leq \Delta(G) + 1 :$$
  - Color G one vertex at a time; color each vertex with the "smallest" color not used for a colored vertex adjacent to it.

How hard is it to determine if $\chi(G) \leq k$?

- It's NP complete, just like
  - determining if $c(G) \geq k$
  - determining if G has a Hamiltonian path
  - determining if a propositional formula is satisfiable

Can guess and verify.

The Four-Color Theorem

Can a map be colored with four colors, so that no countries with common borders have the same color?

- This is an instance of graph coloring
  - The vertices are countries
  - Two vertices are joined by an edge if the countries they represent have a common border

A *planar graph* is one where all the edges can be drawn on a plane (piece of paper) without any edges crossing.

- The graph of a map is planar

Four-Color Theorem: Every map can be colored using at most four colors so that no two countries with a common boundary have the same color.

- Equivalently: every planar graph is four-colorable

Four-Color Theorem: History

- First conjectured by Galton (Darwin's cousin) in 1852
- False proofs given in 1879, 1880; disproved in 1891
- Computer proof given by Appel and Haken in 1976
  - They reduced it to 1936 cases, which they checked by computer
- Proof simplified in 1996 by Robertson, Sanders, Seymour, and Thomas
  - But even their proof requires computer checking
  - They also gave an $O(n^2)$ algorithm for four coloring a planar graph
- Proof checked by Coq theorem prover (Werner and Gonthier) in 2004
  - So you don't have to trust the proof, just the theorem prover

Note that the theorem doesn't apply to countries with non-contiguous regions (like U.S. and Alaska).

Bipartite Graphs

A graph $G(V, E)$ is *bipartite* if we can partition V into disjoint sets V_1 and V_2 such that all the edges in E joins a vertex in V_1 to one in V_2.

- A graph is bipartite iff it is 2-colorable
- Everything in V_1 gets one color, everything in V_2 gets the other color.

Example: Suppose we want to represent the "is or has been married to" relation on people. Can partition the set V of people into males (V_1) and females (V_2). Edges join two people who are or have been married.

Example: We can represent the "has taken a course from" relation by taking the nodes to be professors and students with an edge between s and t if student s has taken a course from professor t. Is this bipartite?

Characterizing Bipartite Graphs

Theorem: G is bipartite iff G has no odd-length cycles.

Proof: Suppose that G is bipartite, and it has edges only between V_1 and V_2. Suppose, to get a contradiction, that $(x_0, x_1, \ldots, x_{2k}, x_0)$ is an odd-length cycle. If $x_0 \in V_1$, then x_2 is in V_1. An easy induction argument shows that $x_{2i} \in V_1$ and $x_{2i+1} \in V_2$ for $0 = 1, \ldots, k$. But then the edge between x_{2k} and x_0 is an edge between two nodes in V_1; contradiction!

- Get a similar contradiction if $x_0 \in V_2$.

Conversely, suppose $G(V, E)$ has no odd-length cycles.

- Partition the vertices in V into two sets as follows:
  - Start at an arbitrary vertex x_0; put it in V_0.
  - Put all the vertices one step from x_0 into V_1
  - Put all the vertices two steps from x_0 into V_0;
  - ...

This construction works if all nodes are reachable from x.

- What if some node y isn't reachable from x?

Characterizing Bipartite Graphs

Theorem: G is bipartite iff G has no odd-length cycles.

Proof: Suppose that G is bipartite, and it has edges only between V_1 and V_2. Suppose, to get a contradiction, that $(x_0, x_1, \ldots, x_{2k}, x_0)$ is an odd-length cycle. If $x_0 \in V_1$, then x_2 is in V_1. An easy induction argument shows that $x_{2i} \in V_1$ and $x_{2i+1} \in V_2$ for $0 = 1, \ldots, k$. But then the edge between x_{2k} and x_0 is an edge between two nodes in V_1; contradiction!

- Get a similar contradiction if $x_0 \in V_2$.

Conversely, suppose $G(V, E)$ has no odd-length cycles.

- Partition the vertices in V into two sets as follows:
  - Start at an arbitrary vertex x_0; put it in V_0.
  - Put all the vertices one step from x_0 into V_1
  - Put all the vertices two steps from x_0 into V_0;
  - ...

This construction works if all nodes are reachable from x.

- What if some node y isn't reachable from x?
  - Repeat the process, starting at y

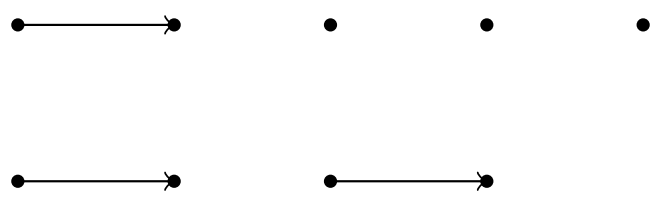
This gives a polynomial-time algorithm to check if G is bipartite.

Matchings

A *matching* in a graph $G = (V, E)$ is a subset M of E no vertex in v is on more than one edge in M

- $M = \emptyset$ is a (not so interesting) matching

Here's another matching:



M is a *perfect matching* if every vertex in V is on an edge in M.

- Marriage defines a matching
  - At least, if you don't allow polygamy or polyandry

  But it's not a perfect matching!

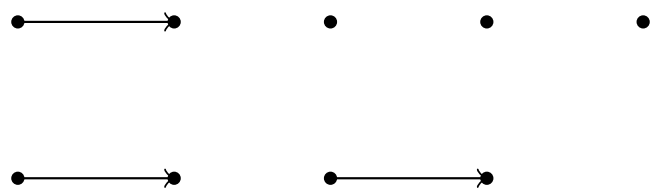
We are particularly interested in finding maximal matches in bipartite graphs (as many people as possible are married).

Matchings

A *matching* in a graph $G = (V, E)$ is a subset M of E no vertex in v is on more than one edge in M

- $M = \emptyset$ is a (not so interesting) matching

Here's another matching:



M is a *perfect matching* if every vertex in V is on an edge in M.

- Marriage defines a matching
  - At least, if you don't allow polygamy or polyandry

  But it's not a perfect matching!

We are particularly interested in finding maximal matches in bipartite graphs (as many people as possible are married).

Given a graph $G = (V, E)$ and $W \subseteq V$, define

$$N(W) = \{t : (s, t) \in E, s \in W\}.$$

$N(W) =$ the neighbors of nodes in W

Hall's Theorem

Theorem: (Hall's Theorem) If $G = (V, E)$ is a bipartite graph with edges from V_1 to V_2, there is a matching M that *covers* V_1 (i.e., every vertex in V_1 is incident to an edge in M) iff, for every subset $W \subseteq V_1$, $|W| \leq |N(W)|$. It's a perfect matching if $|V_1| = |V_2|$.

Hall's Theorem

Theorem: (Hall's Theorem) If $G = (V, E)$ is a bipartite graph with edges from V_1 to V_2, there is a matching M that *covers* V_1 (i.e., every vertex in V_1 is incident to an edge in M) iff, for every subset $W \subseteq V_1$, $|W| \leq |N(W)|$. It's a perfect matching if $|V_1| = |V_2|$.

Proof: If there is a matching V that covers all the vertices in V_1 then clearly $|V'| \leq |N(V')|$ for every subset $V' \subseteq V_1$.

For the converse, we proceed by strong induction on $|V_1|$. The base case ($|V_1| = 1$) is trivial.

For the inductive step, there are two cases:

- $|W| < |N(W)|$ for all $W \subseteq V_1$: Choose $v_1 \in V_1$ and $v_2 \in V_2$ such that $\{v_1, v_2\} \in E$. Add $\{v_1, v_2\}$ to the matching M and repeat the process for $G' = (V', E')$, where $V' = V - \{v_1, v_2\}$, $V_1' = V_1 - \{v_1\}$, $V_2' = V_2 - \{v_2\}$, and E' is the result of removing all edges involving v_1 or v_2 from E.

Hall's Theorem

Theorem: (Hall's Theorem) If $G = (V, E)$ is a bipartite graph with edges from V_1 to V_2, there is a matching M that *covers* V_1 (i.e., every vertex in V_1 is incident to an edge in M) iff, for every subset $W \subseteq V_1$, $|W| \leq |N(W)|$. It's a perfect matching if $|V_1| = |V_2|$.

Proof: If there is a matching V that covers all the vertices in V_1 then clearly $|V'| \leq |N(V')|$ for every subset $V' \subseteq V_1$.

For the converse, we proceed by strong induction on $|V_1|$. The base case ($|V_1| = 1$) is trivial.

For the inductive step, there are two cases:

- $|W| < |N(W)|$ for all $W \subseteq V_1$: Choose $v_1 \in V_1$ and $v_2 \in V_2$ such that $\{v_1, v_2\} \in E$. Add $\{v_1, v_2\}$ to the matching M and repeat the process for $G' = (V', E')$, where $V' = V - \{v_1, v_2\}$, $V_1' = V_1 - \{v_1\}$, $V_2' = V_2 - \{v_2\}$, and E' is the result of removing all edges involving v_1 or v_2 from E.
- If $|W| = |N(W)|$ some some $W \subseteq V_1$, this won't work (the induction hypothesis isn't maintained: may have $|W| > |N(W)|$). Instead, remove $W \cup N(W')$.

Representing Relations Graphically

Recall that *binary relation R on S* is a subset of $S \times S$ (MCS, Section 4.4):

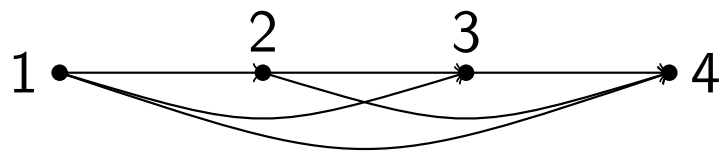
- a set of ordered pairs, where both components are in S.

Given a relation R on S, we can represent it by the directed graph $G(V, E)$, where

- $V = S$ and
- $E = \{(s, t) : (s, t) \in R\}$

A function $f : S \to S$ is a special case of a relation on S, where each node in S has outdegree 1.

Example: Representing the $<$ relation on $\{1, 2, 3, 4\}$ graphically.



Properties of Relations

- A relation R on S is *reflexive* if $(s, s) \in R$ for all $s \in S$;
- A relation R on S is *symmetric* if $(s, t) \in R$ whenever $(t, s) \in R$;
- A relation R on S is *transitive* if $(s, t) \in R$ and $(t, u) \in R$ implies that $(s, u) \in R$.

Examples:

- $<$ is transitive, but not reflexive or symmetric
- $\leq$ is reflexive and transitive, but not symmetric
- equivalence mod m is reflexive, symmetric, and transitive
- "sibling-of" is symmetric. Is it transitive or reflexive?
- "ancestor-of" is transitive (and reflexive, if you're your own ancestor)

How does the graphical representation show that a graph is

- reflexive?
- symmetric?
- transitive?

- If R and R' are relations on S, then so is $R \circ R'$:

$$R \circ R' = \{(s, u) : \exists t((s, t) \in R', (t, u) \in R)\}.$$

  - This agrees with the standard definition of function composition ($f \circ f'$) if R and R' are functions.
  - The reversal of order is not a typo!
- $R^2 = R \circ R$ consists of all pairs such that there is a path of length 2 in the graph representing R.
- What's R^k?

- If R and R' are relations on S, then so is $R \circ R'$:

$$R \circ R' = \{(s, u) : \exists t((s, t) \in R', (t, u) \in R)\}.$$

  - This agrees with the standard definition of function composition ($f \circ f'$) if R and R' are functions.
  - The reversal of order is not a typo!
- $R^2 = R \circ R$ consists of all pairs such that there is a path of length 2 in the graph representing R.
- What's R^k?
- $R^{-1} = \{(t, s) : (s, t) \in R\}$.

Transitive Closure

The *transitive closure* of a relation R is the smallest relation R^* such that

1. $R \subset R^*$
2. R^* is transitive (so that if $(u, v), (v, w) \in R^*$, then so is (u, w)).

Intuitively, R^* is what you get if you add the least number of edges to R to make it transitive

- I.e., add (a, b) if there is a path from a to b in (the graph representing) R, but (a, b) is not already in R.

Transitive Closure

The *transitive closure* of a relation R is the smallest relation R^* such that

1. $R \subset R^*$
2. R^* is transitive (so that if $(u, v), (v, w) \in R^*$, then so is (u, w)).

Intuitively, R^* is what you get if you add the least number of edges to R to make it transitive

- I.e., add (a, b) if there is a path from a to b in (the graph representing) R, but (a, b) is not already in R.

How do we know that there is such a smallest relation?

- Because if R_1 is transitive and contains R, and R_2 is transitive and contains R, then $R_1 \cap R_2$ is also transitive and contains R

Theorem: $R^* = R \cup R^2 \cup R^3 \cup \ldots$.

Proof: Let $R' = R \cup R^2 \cup R^3 \cup \ldots$. To show that $R' = R^*$, we need to show that (1) R' contains R, (2) R' is transitive, and (3) R' is the smallest transitive relation containing R.
(1) is obvious. For (2), suppose that $(a, b) \in R'$ and $(b, c) \in R'$. Then $(a, b) \in R^k$ for some k, so there is a path of length k from a to b, and $(b, c) \in R^m$ for some m, so there is a path of length m from b to c. It follows that there is a path of length $k + m$ from a to c. Thus, $(a, c) \in R^{k+m}$, so $(a, c) \in R'$. Thus, R' is transitive.
For (3), taking $R^1 = R$, we first prove by induction on k that R^k is in every transitive relation that contains R. So suppose that R'' is a transitive relation that contains R.

- Base case: obvious, since $R \subseteq R''$

- Inductive step: suppose that $R^k \subseteq R''$ and that $(a, b) \in R^{k+1}$. Thus, there is a path of length $k + 1$ from a to b in (the graph representing) R. Let c be the vertex just before b on this path. Then there is a path of length k from a to c, and an edge from c to b. Thus, $(a, c) \in R^k$ and $(c, b) \in R$. By the induction hyptothesis, $(a, c) \in R''$. Since $R \subseteq R''$, $(c, b) \in R''$. Since R'' is transitive by assumption, $(a, b) \in R''$. It follows that $R^{k+1} \subseteq R''$.

Since $R^k \subseteq R''$ for all k, $R' = R \cup R^2 \cup R^3 \cup \ldots \subseteq R''$.
Since every transitive relation that contains R contains R', it follows that R' is the *smallest* transitive relation that contains R; i.e., $R' = R^*$.

Example: Suppose $R = \{(1,2),(2,3),(1,4)\}$.

- $R^* = \{(1,2),(1,3),(2,3),(1,4)\}$
- we need to add $(1,3)$, because $(1,2),(2,3) \in R$

Note that we don't need to add (2,4).

- If (2,1), (1,4) were in R, then we'd need (2,4)
- (1,2), (1,4) doesn't force us to add anything (it doesn't fit the "pattern" of transitivity.

If R is already transitive, then $R^* = R$.

Example: Suppose $R = \{(1,2),(2,3),(1,4)\}$.

- $R^* = \{(1,2),(1,3),(2,3),(1,4)\}$
- we need to add $(1,3)$, because $(1,2),(2,3) \in R$

Note that we don't need to add (2,4).

- If (2,1), (1,4) were in R, then we'd need (2,4)
- (1,2), (1,4) doesn't force us to add anything (it doesn't fit the "pattern" of transitivity.

If R is already transitive, then $R^* = R$.

Lemma: R is transitive iff $R^2 \subseteq R$.

Equivalence Relations

- A relation R is an *equivalence relation* if it is reflexive, symmetric, and transitive
  - $=$ is an equivalence relation
  - *Parity* is an equivalence relation on N; $(x, y) \in \mathit{Parity}$ if $x - y$ is even

Strict partial orders

A relation R on S is *irreflexive* if $(a, a) \notin R$ for all $a \in S$.

- $>$ is an irreflexive relation on the natural numbers

A relation R is *antisymmetric* if $(a, b) \in R$ implies that $(b, a) \notin R$.

- $>$ is an irreflexive relation on the natural numbers

A *strict partial order* is an irreflexive, transitive relation.

- $>$ is an irreflexive relation on the natural numbers

Proposition: A strict partial order is antisymmetric

Proof: Suppose that R is a strict partial order, and $(a, b), (b, a) \in R$. Since R is transitive, $(a, a) \in R$. This is a contradiction, since R is irreflexive.

Weak partial orders

A relation R is a *weak partial order* if it is reflexive, transitive, and antisymmetric.

- $\geq$ is a weak partial order (and so is $\leq$)

More examples:

- The strict subset relation on sets $\subset$ is a strict partial order
- The subset relation $\subseteq$ is a weak partial order
- The divisibility relation $(a|b)$ is a weak partial order on the non-negative integers

R is a *strict (resp., weak) linear order* if R is a strict (resp., weak) partial order where all different elements are comparable:

- either $(a, b) \in R$ or $(b, a) \in R$ if $a \neq b$

Thus, $\leq$ is a weak linear order, but $\subseteq$ is not.

Random Graphs: Connecting Graph Theory, Logic, Probability, and Combinatorics

Suppose we have a random graph with n vertices. How likely is it to be connected?

- What is a *random* graph?
  - If it has n vertices, there are $C(n, 2)$ possible edges, and $2^{C(n,2)}$ possible graphs. What fraction of them is connected?
  - One way of thinking about this. Build a graph using a random process, that puts each edge in with probability $1/2$.

- Given three vertices a, b, and c, what's the probability that there is an edge between a and b and between b and c? $1/4$
- What is the probability that there is no path of length 2 between a and c? $(3/4)^{n-2}$
- What is the probability that there is a path of length 2 between a and c? $1-(3/4)^{n-2}$
- What is the probability that there is a path of length 2 between a and every other vertex? $> (1-(3/4)^{n-2})^{n-1}$

Now use the binomial theorem to compute $(1-(3/4)^{n-2})^{n-1}$

$$\begin{aligned} & (1-(3/4)^{n-2})^{n-1} \\ = \; & 1-(n-1)(3/4)^{n-2}+C(n-1,2)(3/4)^{2(n-2)}+\cdots \end{aligned}$$

For sufficiently large n, this will be (just about) 1.

Bottom line: If n is large, then it is almost certain that a random graph will be connected. In fact, with probability approaching 1, all nodes are connected by a path of length at most 2.

This is not a fluke!

Suppose we consider first-order logic with one binary predicate R.

- Interpretation: $R(x, y)$ is true in a graph if there is a directed edge from x to y.

What does this formula say:

$$\forall x \forall y (R(x, y) \vee \exists z (R(x, z) \wedge R(z, y)$$

This is not a fluke!

Suppose we consider first-order logic with one binary predicate R.

- Interpretation: $R(x, y)$ is true in a graph if there is a directed edge from x to y.

What does this formula say:

$$\forall x \forall y (R(x, y) \vee \exists z (R(x, z) \wedge R(z, y)$$

Theorem: [Fagin, 1976] If P is *any* property expressible in first-order logic using a single binary predicate R, it is either true in almost all graphs, or false in almost all graphs.

This is called a *0-1 law*.
This is an example of a deep connection between logic, probability, and graph theory.

- There are lots of others!

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.
- **Exemplification**: Find a sense in which you can try out a problem or solution on small examples.

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.
- **Exemplification**: Find a sense in which you can try out a problem or solution on small examples.
- **Abstraction**: Abstract away the inessential features of a problem.
  - represent it as a graph
  - describe it in first-order logic

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.
- **Exemplification**: Find a sense in which you can try out a problem or solution on small examples.
- **Abstraction**: Abstract away the inessential features of a problem.
  - represent it as a graph
  - describe it in first-order logic
- **Modularity**: Decompose a complex problem into simpler subproblems.

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.
- **Exemplification**: Find a sense in which you can try out a problem or solution on small examples.
- **Abstraction**: Abstract away the inessential features of a problem.
  - represent it as a graph
  - describe it in first-order logic
- **Modularity**: Decompose a complex problem into simpler subproblems.
- **Representation**: Understand the relationships between different representations of the same information or idea.
  - Graphs vs. matrices vs. relations

Eight Powerful Ideas

(With thanks to Steve Rudich.)

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.
- **Exemplification**: Find a sense in which you can try out a problem or solution on small examples.
- **Abstraction**: Abstract away the inessential features of a problem.
  - represent it as a graph
  - describe it in first-order logic
- **Modularity**: Decompose a complex problem into simpler subproblems.
- **Representation**: Understand the relationships between different representations of the same information or idea.
  - Graphs vs. matrices vs. relations
- **Probabilistic inference:** Drawing inferences from data
  - Bayes' rule
- **Probabilistic methods**: Flipping a coin can be surprisingly helpful!
  - probabilistic primality checking

(A Little Bit on) NP

(No details here; just a rough sketch of the ideas. Take CS 4810/4820 if you want more.)

NP = nondeterministic polynomial time

- a language (set of strings) L is in NP if, for each $x \in L$, you can guess a witness y showing that $x \in L$ and quickly (in polynomial time) verify that it's correct.
- Examples:
  - Does a graph have a Hamiltonian path?
    - guess a Hamiltonian path
  - Is a formula satisfiable?
    - guess a satisfying assignment
  - Is there a schedule that satisfies certain constraints?
  - ...

Formally, L is in NP if there exists a language L' such that

1. $x \in L$ iff there exists a y such that $(x, y) \in L'$, and
2. checking if $(x, y) \in L'$ can be done in polynomial time

NP-completeness

- A problem is NP-hard if every NP problem can be *reduced* to it.

A problem is NP-complete if it is in NP and NP-hard

- Intuitively, if it is one of the hardest problems in NP.

There are *lots* of problems known to be NP-complete

- If any NP complete problem is doable in polynomial time, then they all are.
  - Hamiltonian path
  - satisfiability
  - scheduling
  - ...
- If you can prove P = NP, you'll get a Turing award.