Graphs and Trees

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 give a visual representation of 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.

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

Terminology and Notation

A *graph* G is a pair (V, E), where V is a set of *vertices* or *nodes* and E is a set of *edges* or *branches*; an edge is a set $\{v, v'\}$ of two not necessarily distinct vertices (i.e., $v, v' \in V$).

- We sometimes write $G(V, E)$ instead of G
- If $V = \emptyset$, then $E = \emptyset$, and G is called the *null graph*.

We usually represent a graph pictorially.

- A vertex with no edges incident to it is said to be *isolated*
- If $\{v\} \in E$, then there is a *loop* at v
- $G'(V', E')$ is a *subgraph* of $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

Directed Graphs

Note that $\{v, u\}$ and $\{u, v\}$ represent the same edge.

In a *directed graph* (*digraph*), the order matters. We denote an edge as (v, v') rather than $\{v, v'\}$. We can identify an undirected graph with the directed graph that has edges (v, v') and (v', v) for every edge $\{v, v'\}$ in the undirected graph.

Directed Graphs

Note that $\{v, u\}$ and $\{u, v\}$ represent the same edge.

In a *directed graph* (*digraph*), the order matters. We denote an edge as (v, v') rather than $\{v, v'\}$. We can identify an undirected graph with the directed graph that has edges (v, v') and (v', v) for every edge $\{v, v'\}$ in the undirected graph.

Two vertices v and v' are *adjacent* if there is an edge between them, i.e., $\{v, v'\} \in E$ in the undirected case, $(v, v') \in E$ or $(v', v) \in E$ in the directed case.

Representing Relations Graphically

A *relation R on S* is a subset of $S \times S$

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

Example: Represent 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?

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 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, except that we double-count all self-loops.

- Why? Because things work out better that way

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 an undirected graph without loops. We need to double-count the loops to make this right in general.

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).
- 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*

Connectivity

- An undirected graph is *connected* if there is for all vertices u, v, $(u \neq v)$ there is a path from u to v.
- A digraph is *strongly connected* if for all vertices u, v $(u \neq v)$ there is a path from u to v and from v to u.
- If a digraph is *weakly connected* if, for every pair u, v, there is a path from u to v in the underlying undirected graph.
  - This is the definition in Rosen; other books use different definitions.
- A *connected component* of an (undirected) graph G is a connected subgraph G' which is not the subgraph of any other connected subgraph of G.

Connectivity

- An undirected graph is *connected* if there is for all vertices u, v, $(u \neq v)$ there is a path from u to v.
- A digraph is *strongly connected* if for all vertices u, v $(u \neq v)$ there is a path from u to v and from v to u.
- If a digraph is *weakly connected* if, for every pair u, v, there is a path from u to v in the underlying undirected graph.
  - This is the definition in Rosen; other books use different definitions.
- A *connected component* of an (undirected) graph G is a connected subgraph G' which is not the subgraph of any other connected subgraph of G.

Example: We want the graph describing the interconnection network in a parallel computer:

- the vertices are processors
- there is an edge between two nodes if there is a direct link between them.
  - if links are one-way links, then the graph is directed

We typically want this graph to be connected.

Trees

A *tree* is a digraph such that

(a) with edge directions removed, it is connected and acyclic
- You can remove either the assumption that it is acyclic
- If it is finite, you can alternatively remove the assumption that it is connected

(b) every vertex but one, the *root*, has indegree 1

(c) the root has indegree 0
- If there are only finitely many nodes, you can remove either the assumption that the root has indegree 0 or the assumption that the nodes other than the root have degree 1

Trees come up everywhere:

- when analyzing games
- representing family relationships

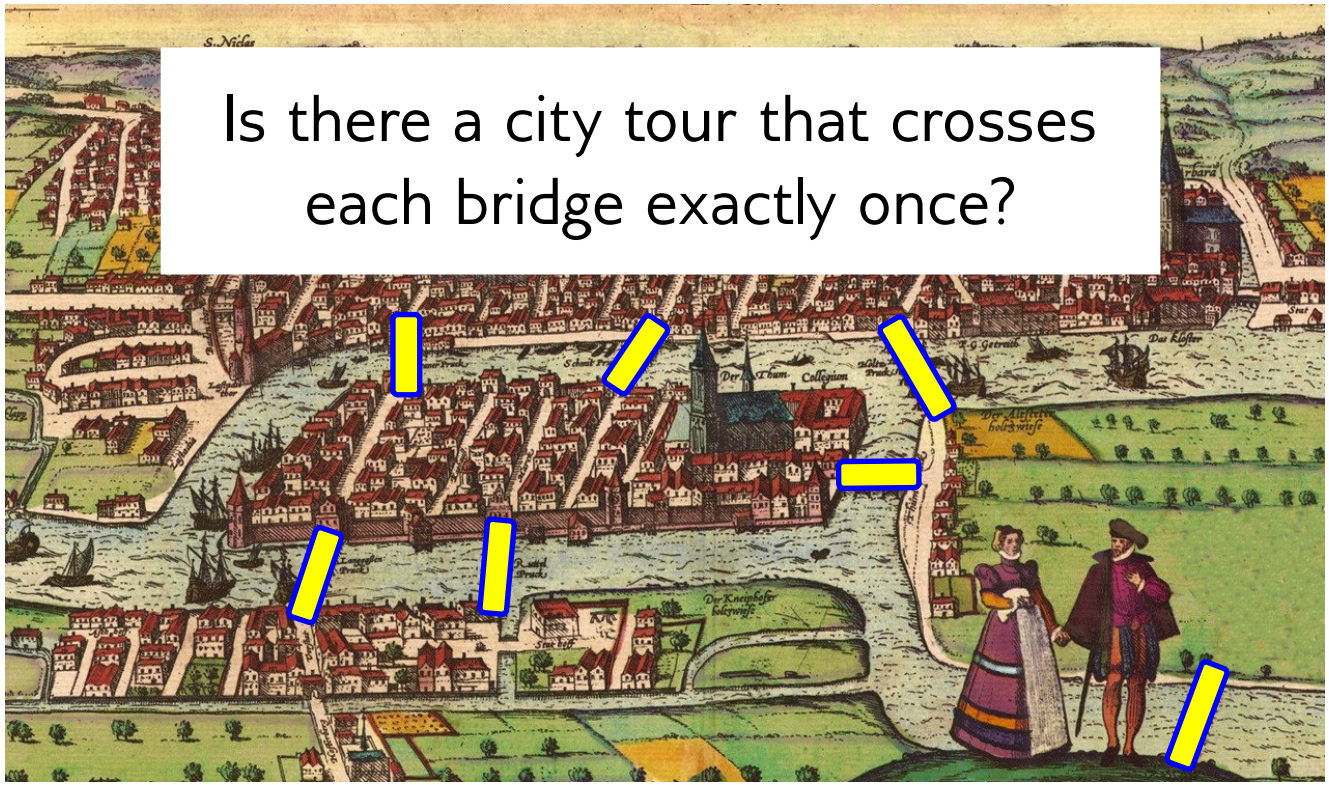
Complete Graphs and Cliques

- An undirected graph $G(V, E)$ is *complete* if it has no loops and for all vertices u v $(u \neq v)$, $\{u, v\} \in E$.
  - How many edges are there in a complete graph with n vertices?

A complete subgraph of a graph is called a *clique*

- The *clique number* of G is the size of the largest clique in G.

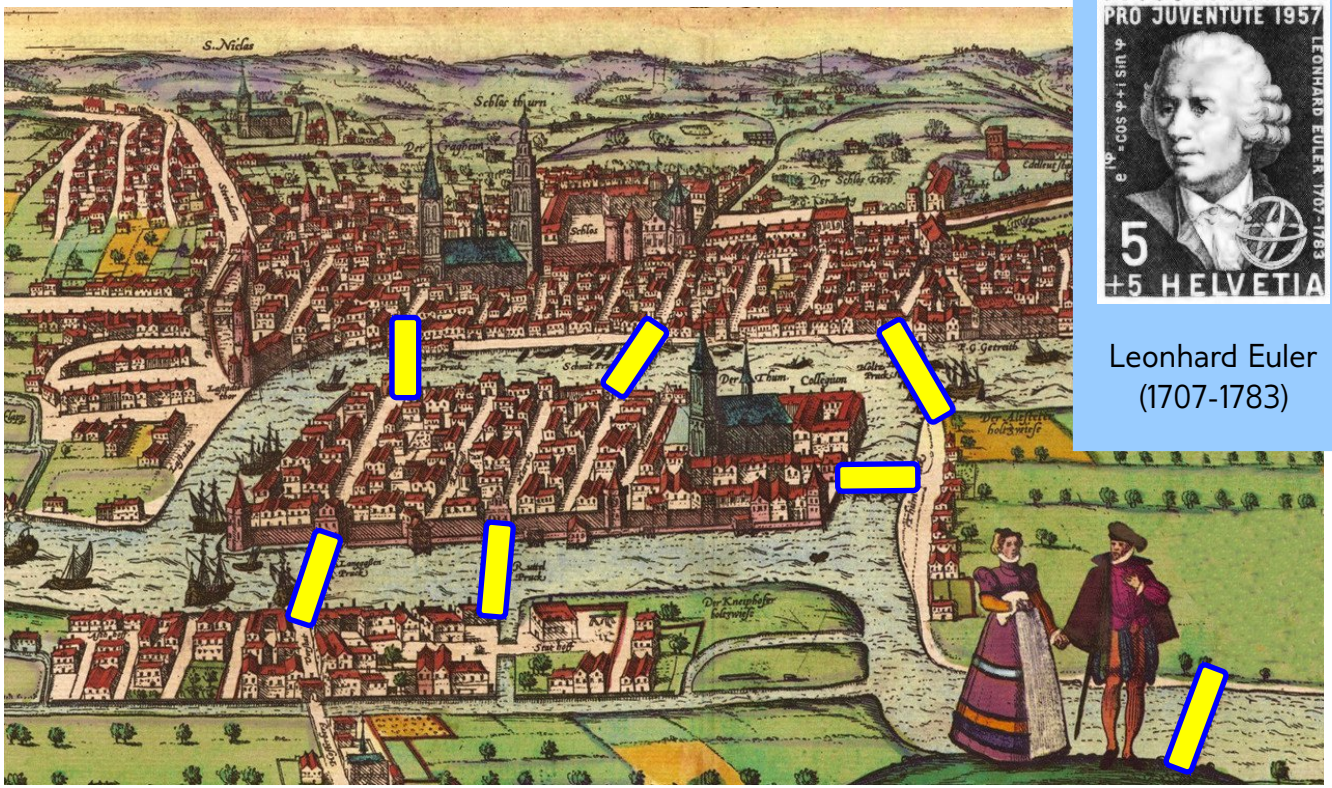
Bridges of Königsberg



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

Remember this from the first class?

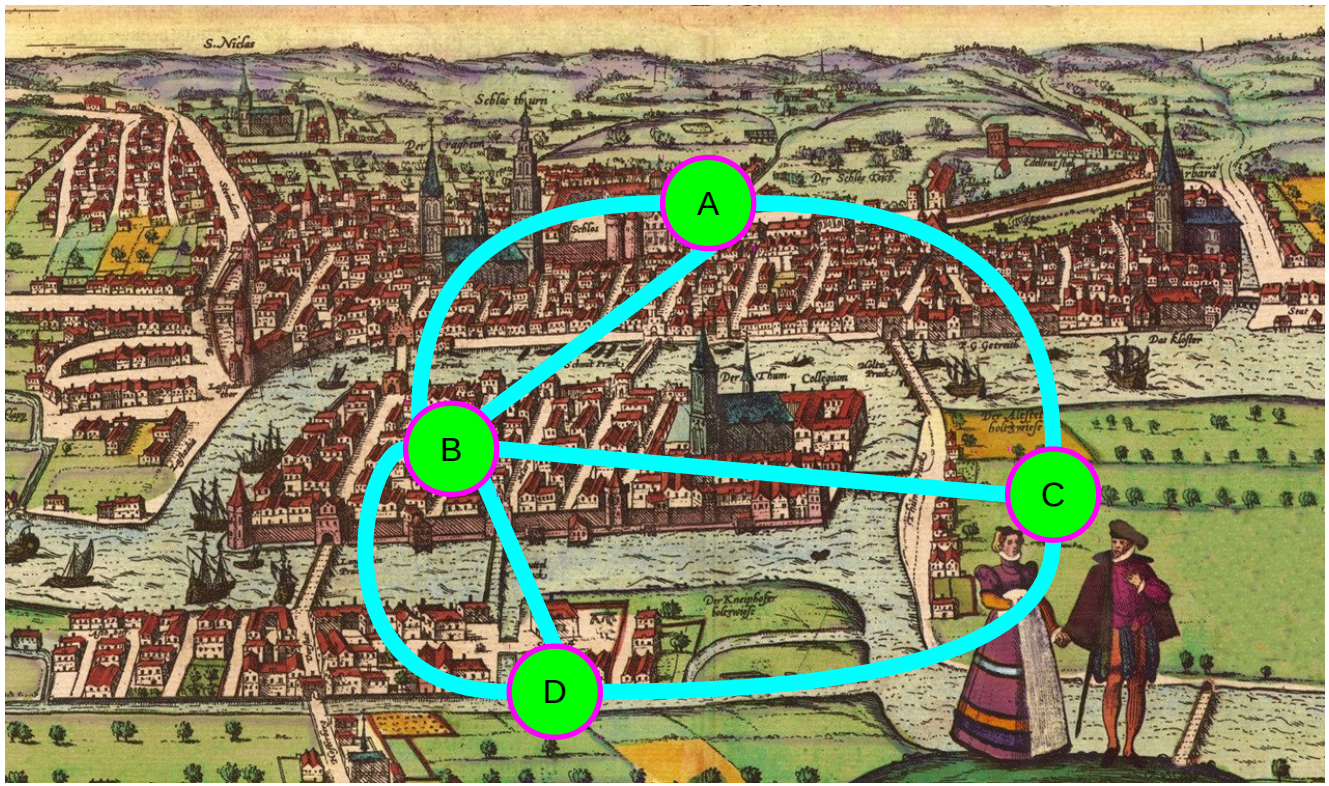
Bridges of Königsberg



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 for checking if a graph has a Hamiltonian cycle/path.

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, $\chi(G) = n$
- If G has a clique of size k, then $\chi(G) \geq k$.
  - 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 means that there is an edge between two nodes in V_1; this is a 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 G is connected and has no odd-length cycles.

- What if G isn't connected?

Searching Graphs

Suppose we want to process data associated with the vertices of a graph. This means we need a systematic way of searching the graph, so that we don't miss any vertices.

There are two standard methods.

- Breadth-first search
- Depth-first search

It's best to think of these on a tree:

Breadth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Breadth-First Search

visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on Q [Q is a queue]
repeat while $Q \neq \emptyset$
  $u \leftarrow head(Q)$ [$head(Q)$ is the first item on Q]
  for $w \in A(u)$ [$A(u) = \{w | \{u, w\} \in E\}$]
    if $w \notin V'$
    then visit w
      Put w on Q
      $V' \leftarrow V' \cup \{w\}$
  endfor
  Delete u from Q

The BFS algorithm basically finds a tree embedded in the graph.

BFS and Shortest Length Paths

If all edges have equal length, we can extend this algorithm to find the shortest path length from v to any other vertex:

- Store the path length with each node when you add it.
- Length$(v) = 0$.
- Length$(w) =$ Length$(u) + 1$

With a little more work, can actually output the shortest path from u to v.

- This is an example of how BFS and DFS arise unexpectedly in a number of applications.
  - We'll see a few more

Depth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Depth-First Search

visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on S [S is a stack]
$u \leftarrow v$
repeat while $S \neq \emptyset$
if $A(u) - V' \neq \emptyset$
then Choose $w \in A(u) - V'$
visit w
$V' = V' \cup \{w\}$
Put w on stack
$u \leftarrow w$
else $u \leftarrow top(S)$ [Pop the stack]
endif

- DFS uses *backtracking*
  - Go as far as you can until you get stuck
  - Then go back to the first point you had an untried choice

Spanning Trees

A *spanning tree* of a connected graph $G(V, E)$ is a connected acyclic subgraph of G, which includes all the vertices in V and only (some) edges from E.

Think of a spanning tree as a "backbone"; a minimal set of edges that will let you get everywhere in a graph.

- Technically, a spanning tree isn't a tree, because it isn't directed.

The BFS search tree and the DFS search tree are both spanning trees.

- Rosen gives algorithms to produce minimum weight spanning trees
- That's done in CS 4820, so we won't do it here.

More on Relations

Recall that a relation on S is a subset of $S \times S$, and that a relation can be represented using a graph

- The nodes are elements of S; there is an edge from s to t iff $(s, t) \in R$.

More generally, a relation R on $S \times T$ is a subset of $S \times T$.

- We can represent this graphically using a graph where the nodes are labeled with elements of $S \cup T$
- The graph is *bipartite*: edges only go from nodes in S to nodes in T.

More on Relations

Recall that a relation on S is a subset of $S \times S$, and that a relation can be represented using a graph

- The nodes are elements of S; there is an edge from s to t iff $(s, t) \in R$.

More generally, a relation R on $S \times T$ is a subset of $S \times T$.

- We can represent this graphically using a graph where the nodes are labeled with elements of $S \cup T$
- The graph is *bipartite*: edges only go from nodes in S to nodes in T.

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

- for every elemenet in S, there's a unique element in T that's $f(s)$

Put another way, relations just generalize functions.

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

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

- $R^2 = R \circ R$ consists of all pairs such thath there is a path of length i2 in the graph representing R.
- What's R^k?

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)).

Can prove that $R^* = R \cup R^2 \cup R^3 \cup \ldots$.

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)).

Can prove that $R^* = R \cup R^2 \cup R^3 \cup \ldots$.

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 $R^* = R$.

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)).

Can prove that $R^* = R \cup R^2 \cup R^3 \cup \ldots$.

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