Versatile Mathematics

COMMON MATHEMATICAL APPLICATIONS

Josiah Hartley
Frederick Community College

Val Lochman
Frederick Community College

Erum Marfani
Frederick Community College

2nd Edition

2020

Attributions This book benefited tremendously from others who went before and freely shared their creative work. The following is a short list of those whom we have to thank for their work and their generosity in contributing to the free and open sharing of knowledge.

- David Lippman, author of *Math in Society.* This book uses sections derived from his chapters on Finance, Growth Models, and Statistics. He also administers MyOpenMath, the free online homework portal to which the problems in this text were added.
- The developers of onlinestatbook.com.
- OpenStax College (their book *Introductory Statistics* was used as a reference)
  OpenStax College, *Introductory Statistics.* OpenStax College. 19 September 2013. <http://cnx.org/content/col11562/latest/>
- The authors of OpenIntro Statistics, which was also used as a reference.
- The Saylor Foundation Statistics Textbook: http://www.saylor.org/site/textbooks/Introductory%20Statistics.pdf

Thanks The following is a short list of those whom we wish to thank for their help and support with this project.

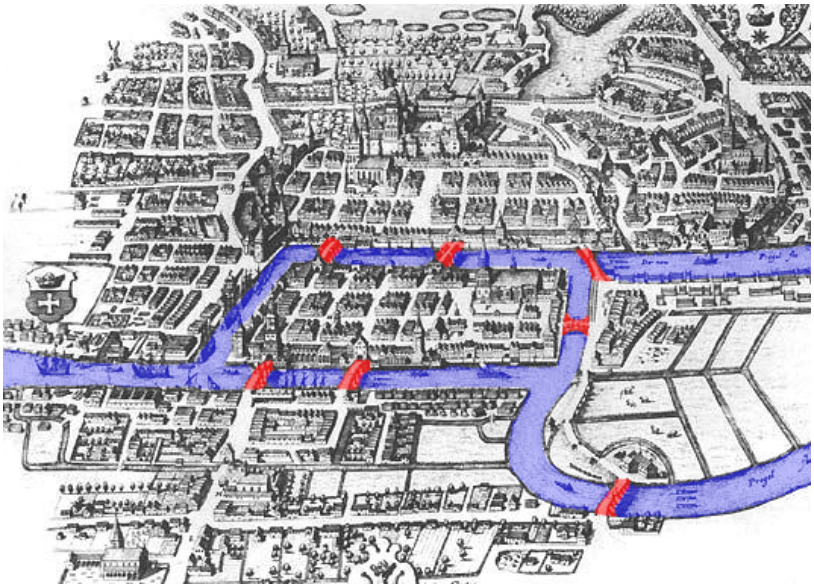
- The President's office at Frederick Community College, for providing a grant to write the first chapters.
- Gary Hull, who in his tenure as department chair gave us his full support and gave us the impetus to start the project, and generously shared his notes for MA 103.
- The entire FCC math department, who provided untold support and encouragement, as well as aid in reviewing and editing the text.

Contents

Chapter

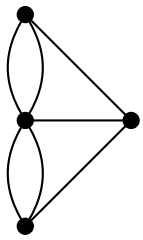
1 Graph Theory

SECTION 1.1 Introduction to Graphs



In the early 1700s, the city of Königsberg in Prussia (now Kaliningrad, Russia), which was split by the Pregel River, had 7 bridges connecting the north and south banks of the city to two islands in the center of the river, as shown in the drawing above. A popular puzzle of the day challenged travelers to plan a walk through this part of the city in such a way that they would cross each bridge exactly once (you may want to pause and see if you can find such a path).

Leonhard Euler, in 1736, turned his attention to this problem. Although the problem doesn't seem to be an important one, what is interesting is how Euler solved it. He noticed that the path a person travels on a given land mass is irrelevant; all that matters is which bridges they cross. Thus, he decided to simplify the picture by simply drawing a dot to represent each land mass, and a line connecting two dots to represent a bridge. Here's the result:



That's the first example of a **graph**; while we use the word *graph* to mean several different things, in this chapter, a graph will refer to a diagram like the one above.

Graphs

A **graph**, in graph theory, consists of **nodes** (or vertices) and **edges**; each edge connects one node to another.

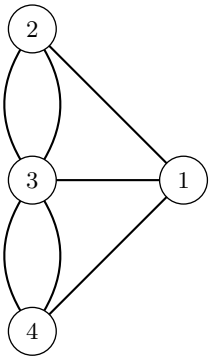
Now, notice that we've drawn the nodes in positions that roughly correspond to the orientation of the map; the one on the right represents the eastern island, for instance, while the two on the top and bottom of the left row represent the northern and southern banks.

Location is not important.

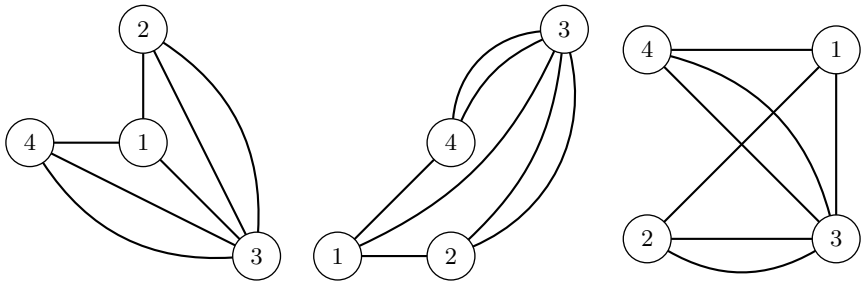
However, this is completely arbitrary; since we simplified the picture to land masses and connections, it turns out that the nodes can be rearranged at will, as long as the final result has the same pattern of connections, meaning that the same nodes are connected by edges.

Connections are what matters.

Before we illustrate this, let's label each node so that we can see them move:



Each of these graphs is equivalent, according to Euler's new theory, to the original one:



In fact, we could describe it in words, by saying something like

Of course, it's much more fun to draw pictures than to write something like this.

There are 4 nodes.
Node 1 is connected to nodes 2, 3, and 4.
Node 2 is connected to node 3 by two edges.
Node 3 is connected to node 4 by two edges.

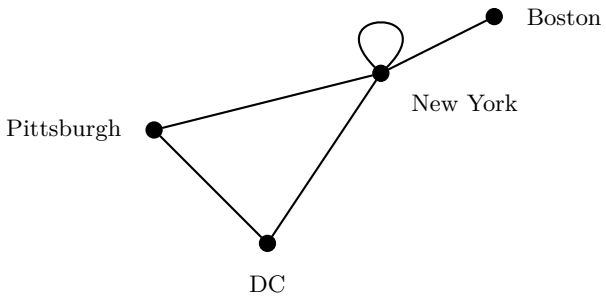
Any picture you could draw that fits this description would be equivalent to the first graph that we drew.

Examples and Definitions

Let's take a look at a few examples of graphs, and along the way, we'll encounter a few new terms that we can use to categorize and describe them.

Application 1

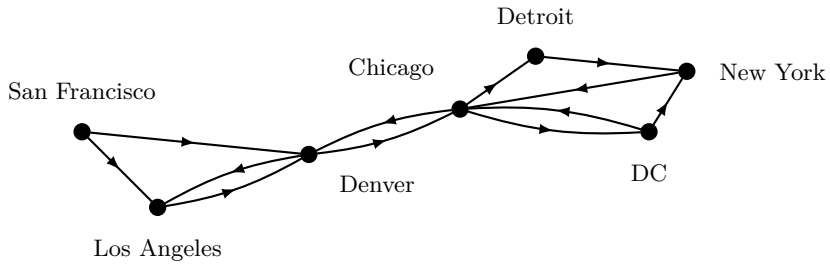
Transportation Network The graph below shows a simple train system; each edge indicates a line that runs between two cities.



Notice the **loop** at New York; this implies that there is a train line that returns to the same station from which it leaves. There could be, for instance, a sightseeing train that simply takes passengers on a short loop. We simply include it here to show the possibility of a loop, which is an edge that connects a node to itself.

Application 2

Communication Network This second example shows a fictional network, which could consist of connections between data centers.



Notice that the edges on this graph have arrows that indicate which direction the data can travel; for instance, data can flow from Detroit to New York, but to send data in the opposite direction, it would have to go from New York to Chicago, and then on to Detroit.

Also, this graph, unlike the one before it, has multiple edges between some pairs of nodes (we've seen this before, on the graph of Königsberg).

These two distinctions lead to our first set of definitions; the following terms can help to categorize graphs, and it is useful to think about new applications in terms of what category they fit into.

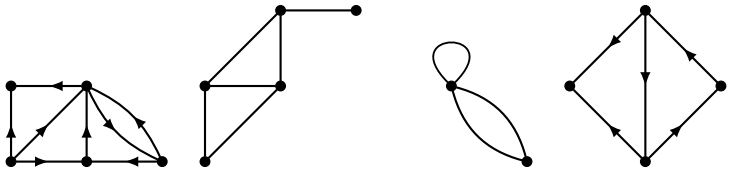
Graph Classifications

Simple Graphs and Multigraphs: a **simple graph** is one that has at most one edge between two nodes, and contains no loops; a **multigraph** can have multiple edges connecting the same pair of nodes, and may contain loops.

Undirected and Directed Graphs: an **undirected graph** has no notion of direction to the edges; in a **directed graph** (also called a **digraph**), each edge has an associated direction.

EXAMPLE 1 CLASSIFYING GRAPHS

For each of the following graphs, determine whether it is a simple graph or multigraph, and whether it is directed or undirected.



(a) (b) (c) (d)

Solution

(a) Since the edges have directions associated with them, and there is one pair of nodes that have multiple edges between them, this is a directed multigraph

(b) There are no multiple edges or loops, and no directions, so this is an undirected simple graph

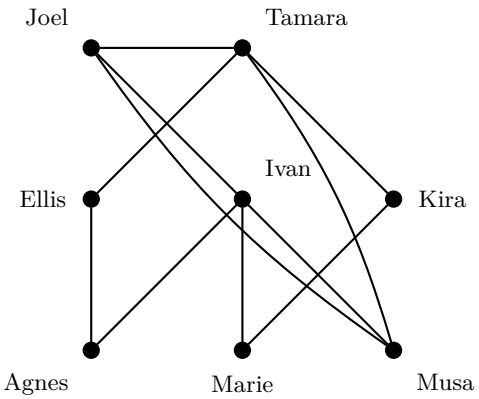
(c) There are no arrows, but because of the multiple edges and the loop, this is an undirected multigraph

(d) There are no multiple edges or loops, so this one is simple, but because of the arrows, it is a directed simple graph

Let's go back to examples of graph theory applications.

Application 3

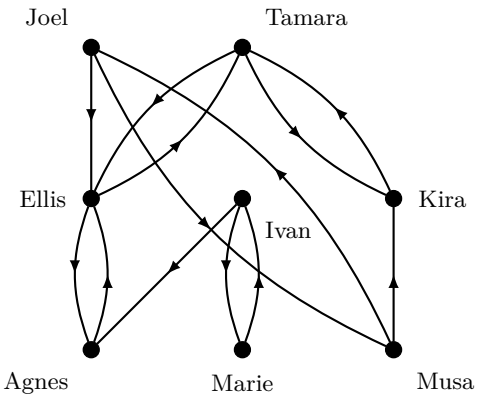
Social Network Suppose, for instance, that we selected a small group of people and checked whether they are connected on Facebook; if two people are friends, we draw an edge between them.



This is a simple graph, since it doesn't make sense to have more than one connection between two people; they are either friends or are not. Also, since Facebook friendship is a two-way relationship, this is an undirected graph.

If we wanted to make a directed multigraph for a social network, we could use Instagram, for instance, since one person can follow another without being followed back. The result might look like this:

Notice how complicated this is already, with only 8 people. Real social networks have incredibly complicated graphs; although we may not be able to draw them, computers can still analyze them to glean all sorts of information, like which users are the most influential.



Here, an edge indicates whether one person follows another, and the direction of the arrow goes from the follower to the one they follow.

Notice that one of the things we can glean from a graph like these is who in this social group is the most well-connected. The easiest way to do this is simply to count the number of connections that each person has to others, which corresponds to the number of edges that connect to that node. We have a special name for this count: we call it the **degree** of a node.

Degree

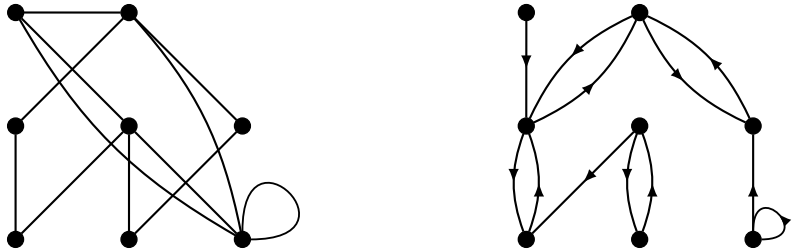
The **degree** of a node in an undirected graph is the number of edges that connect to it.

Note: a loop counts twice, since the loop connects to the node at both of its ends. This is mostly by convention, but it is important so that some concepts are consistent whether loops are included or not.

In a directed graph, each node has two degrees: the **in-degree** and **out-degree**. The in-degree counts how many edges come into that node; the out-degree counts how many edges leave from that node.

EXAMPLE 2 DEGREES OF NODES

Find the degree of each node in the social network graphs shown here.

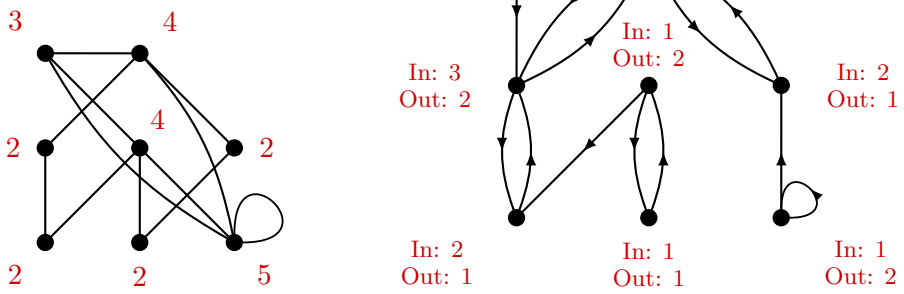


(a) (b)

Solution

We simply need to count the number of edges that connect at each node. The only complication in part (a) is the loop on the lower-right node; remember that the loop adds a total of 2 the degree of that node. For part (b), pay attention to the direction of each edge; an edge adds one to the out-degree of its starting node and one to the in-degree of its ending node (note that a loop here contributes one each to the in-degree and out-degree of its node).

Here's the final result; each node is labeled with its degree(s):



(a) (b)

There are a couple of interesting observations we can make about node degrees.

Degrees: Observation 1

The sum of the degrees of all the nodes in an undirected graph is always even.

Similarly, the sum of all in-degrees and out-degrees of all the nodes in a directed graph is always even.

The reason for this is very simple: each edge contributes two degrees to the total for the graph, one degree at each end, so no matter how many edges there are, there will be twice as many degrees, and two times anything is always even. In a directed graph, each edge contributes one out-degree and one in-degree; thus, the total in-degrees and out-degrees will be equal, and when these are added, the result will also be even.

Degrees: Observation 2

An undirected graph has an even number of nodes with odd degree.

To verify this, we need to remember that the total number of degrees is even, and we need to know a principle of even numbers: **if you start with an even number, add something to it, and end up with an even number, the number you added must have been even**. In other words, if you add an even number and an odd number, the result will be odd.

Therefore, if we split the graph into even nodes and odd nodes, we can split up the total of all degrees (which we know is even) like this:

$$\text{degrees of even nodes} + \text{degrees of odd nodes} = \text{even}$$

Since the first part is even (if you add up a bunch of even degrees, the result is even), and the result is even, we know that the total number of degrees for all the odd nodes must be even also (that's the bolded principle in the last paragraph).

Think about adding together a bunch of odd numbers: when we add two of them, we get an even result, but adding a third makes the result odd. Adding a fourth makes it even again, and so on, so clearly there must be an even number of these nodes to make the result even.

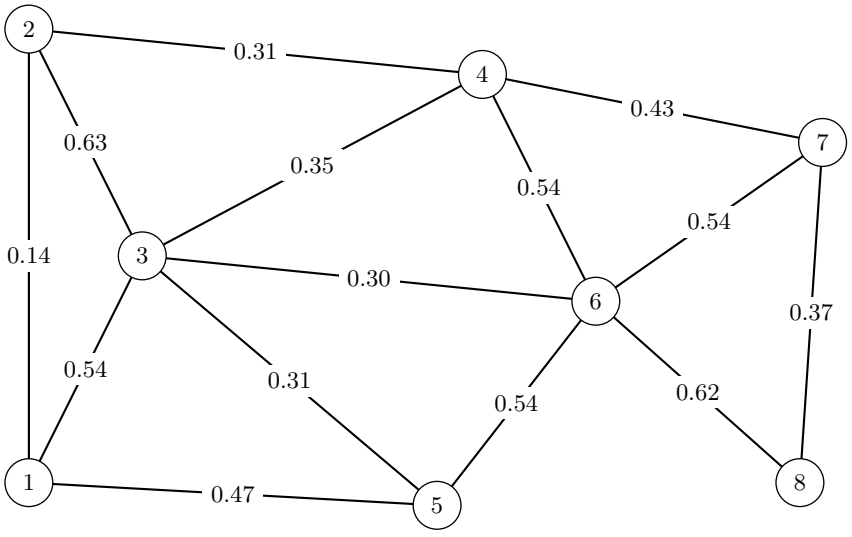
If that explanation wasn't clear to you, don't worry too much; this is simply an interesting principle that we can observe about graphs.

Back to examples!

Application 4

Street Map The graph below could represent a small segment of a street map, where each edge corresponds to a street, and each node corresponds to an intersection.

This is the way that Google Maps, for instance, stores mapping information. Notice the number written along each edge; these are called edge **weights**. These could represent the distance between each pair of nodes, or perhaps the time that it will take to cover that distance. Again, for services like Google Maps that track traffic information, these weights would be constantly updated to reflect current traffic data.



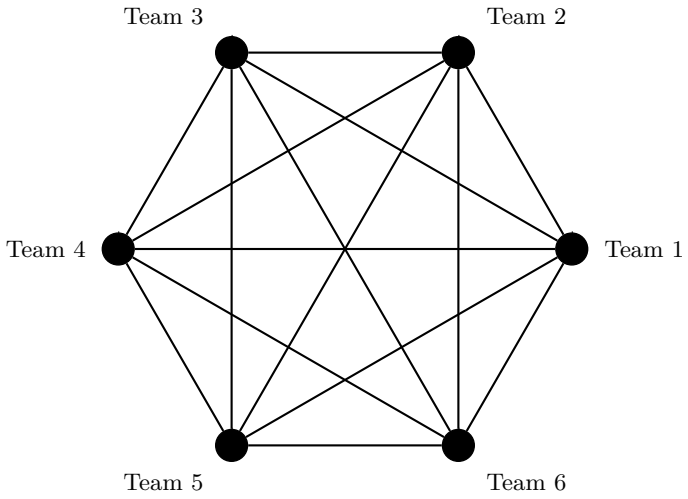
Weighted Graphs

A **weighted graph** has a weight associated with each edge; these weights can represent things like distance or cost.

Since the location of the nodes in a graph is not significant, weights can be used to encode distance information without having to worry about drawing a graph in such a way that it shows the distances between nodes.

Application 5

Round-Robin Tournament A tournament in which each team competes against every other team is called a *round-robin* tournament. If we draw a graph in which each node corresponds to a team and each edge represents a game played between two teams, it could look like the following.



This graph is interesting because every node is connected to every other node; we have a special name for graphs like this: we call them **complete graphs**.

Complete Graphs

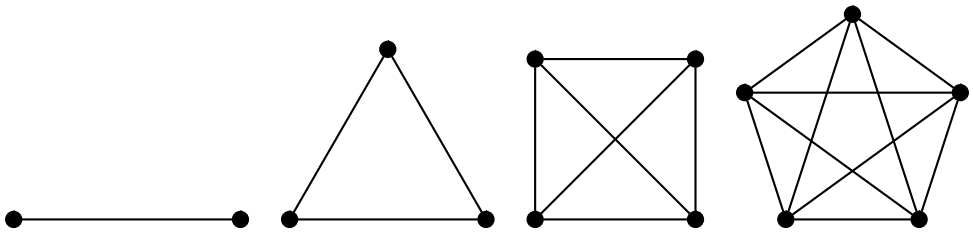
A **complete graph** is an undirected graph with an edge between every pair of nodes.

A complete graph with n nodes is often labeled K_n.

K_n has $\dfrac{n(n-1)}{2}$ edges.

Sidenote: there is an interesting probability question called the Birthday Problem, which deals with the probability that in a group of n people, at least two people will share the same birthday. It turns out that in a group of only 23 people, the probability is over 50%. This sounds surprising, because that seems like a very small group, but since the significant part is the *pairing* of people, we can think of this group as a complete graph with 23 nodes. By looking at the one above, you can imagine that K_{23} is much more complicated (with 253 edges). Now, recognizing that there are 253 pairings, the likelihood that one of these pairings will match birthdays is less surprising (there's more to the problem than this, but that's the core concept).

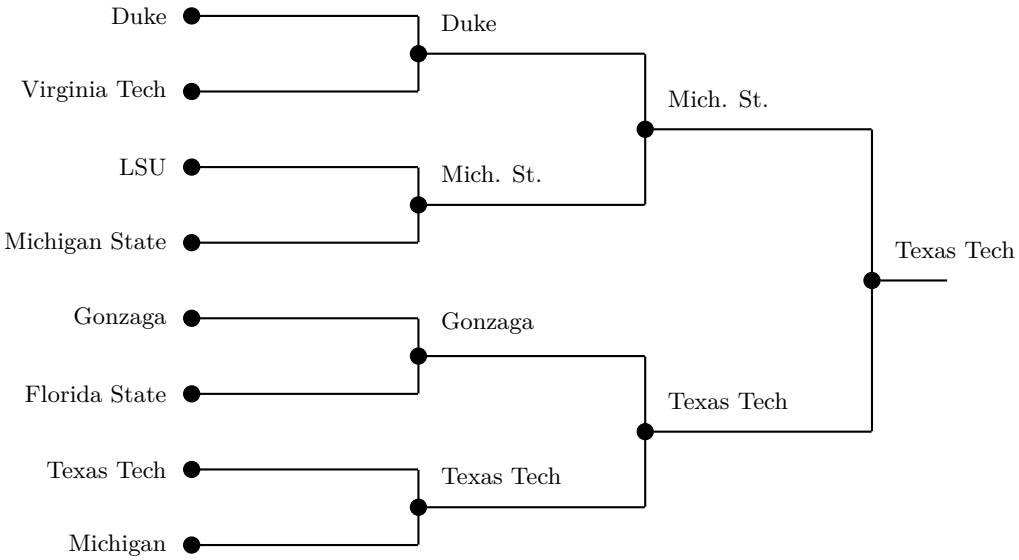
The first four complete graphs (K_2 through K_5) are shown below. For obvious reasons, K_1 is not interesting enough to show.



Application 6

Single-Elimination Tournament Another type of tournament, like most playoffs, is an **elimination** tournament, in which each team is paired up with another; the loser is knocked out, and the winner moves on to the next round to compete against the winner of another pairing. For simplicity, we can focus on single-elimination tournaments, where each competition consists of a single game (unlike the NBA playoffs, for instance, in which teams play a best-of-7-game series in each round).

The graph below shows a portion of the 2019 NCAA men's basketball tournament, from the Sweet Sixteen round onward in the West and East regions:



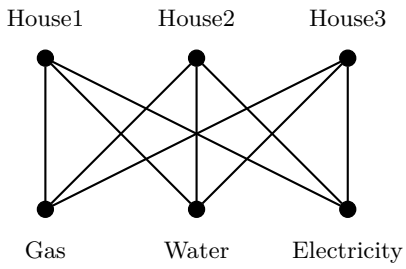
Notice that we could have drawn the full graph for the full bracket; instead, we drew only a *portion* of it. The graph above is a **subgraph** of the full tournament graph.

Subgraphs

If you start with a graph, and remove some nodes and/or edges, the result is a **subgraph** of the original one (the original one is called a *supergraph* of the smaller one).

Application 7

Utility Connections Suppose we need to connect three houses to three utilities; the graph could look like the following.



It turns out that this is an example of a *bipartite graph*, which is one that can be split into two sets of nodes, where there are only connections between the two sets (none within each set). That's not what we'll focus on here, though.

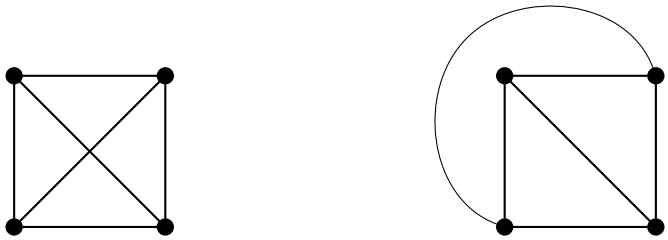
The question we'll think about is this one: is it possible to draw this graph in such a way that none of the edges cross? In practical terms, this would mean arranging the houses and utilities in such a way that burying the utility line would be simplified by not having to worry about the others while digging.

Planar Graphs

A **planar graph** is one that can be drawn without any edges crossing.

There are many applications of planar graphs; for instance, when designing a circuit board, engineers must lay out the connections in such a way that none of them cross, and highway engineers encounter a similar problem.

Let's take a look at a simpler example: we can show that K_4 is a planar graph by moving one of the edges to the outside. On the left below, we have the familiar representation of K_4, and on the right side, we have a planar representation for it:



Back to the example with the houses and utilities: it turns out that this example is *non-planar*. You can convince yourself of this by trying to draw its planar representation, but we won't show the full proof here for the sake of simplicity. In short, though, if you start with two houses and two utilities, that graph *is* planar, and you can add a third house without crossing any edges. Once you do, however, there's no region in which you can place the final utility in such a way that it can connect to all three houses without two edges intersecting.

We haven't exhausted the possibilities for applications, but hopefully the pattern is clear: whenever an application involves some sort of connection that can occur between two items, a graph can be used to model this situation. Once you start thinking about the world in this way, it becomes clear that the applications of graph theory are nearly unlimited.

The terms and definitions in this section are ones that you should be familiar with, since we'll use them throughout the rest of the chapter. As long as you can connect each concept with the relevant examples, you'll be well prepared for the rest of our study of graph theory.

SECTION 1.2 Euler and Hamilton Paths

CC BY-SA 2.0, Gage Skidmore

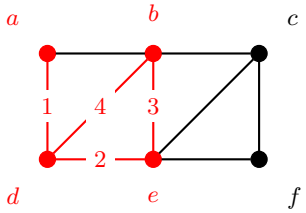
During a magazine interview in 1994, the actor Kevin Bacon made an offhand comment that he had "worked with everybody in Hollywood or someone who's worked with them." From this sprang the idea of the *Six Degrees of Kevin Bacon*, which sometimes appears as a game in which players must find a set of links between any given actor and Bacon. For instance, Emma Watson has a Bacon number of 2, meaning that it takes two steps to get from her to Bacon (Watson was in *The Circle* with Bill Paxton, who was in *Apollo 13* with Kevin Bacon).

It turns out that, as the name suggests, a surprising number and range of actors can be connected to Kevin Bacon (or really any prolific actor) with six steps or fewer. In fact, according to the website *oracleofbacon.org*, there are over 1.2 million actors with a Bacon number of 4 or less, and the average Bacon number is just over 3.

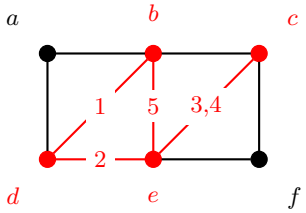
Paths and Circuits

What we've described above is the process of finding the **shortest path** between two nodes on a graph (a graph where the nodes represent actors, and two nodes are connected by an edge if those two actors have appeared together). We'll talk more about finding shortest paths in the next section, but for now, we can simply start with the definition of a **path**.

The definition of a path is quite intuitive: start at one node, then move along edges to another node, and you've taken a path. We use the term **circuit** to describe a path that ends at the same node where it started (so a *circuit* is a specific type of path; the term *path* is a general one that includes circuits).



Path: $a \to d \to e \to b \to d$



Circuit: $b \to d \to e \to c \to e \to b$

Note that in an undirected graph (like the two examples above), a path can travel in either direction along an edge, but in a directed graph, a path is only allowed to take an edge in its specified direction.

Paths and Circuits

A **path** in a graph is a sequence of edges; if the graph is simple (no multiple edges between nodes), the path can be described using the nodes that it passes through in order.

A **circuit** is a path that starts and ends at the same node.

Notice another distinction between the two graphs above: on the left, the path used four *distinct* edges to travel from a to d (although the path passed through d once before ending there), while the circuit on the right used the same edge (between c and e) twice. This brings us to another definition.

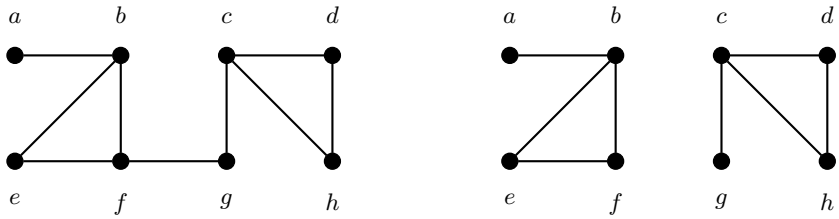
Simple Paths and Circuits

A **simple** path or circuit is one that does not contain any edge more than once.

In a moment, we'll return to the problem at the beginning of the last section: the Königsberg Bridge Problem. Remember that in that problem, we were hunting for a journey through the city that did not reuse any bridges; it turns out that we were actually looking for a *simple path*. Before we go back to that problem, though, we need to discuss one more concept: connectivity.

Connected Graphs

A connected graph is exactly what you would expect:



Connected

Disconnected (with two connected components)

Note: it's a bit more complicated for a directed graph; in that case, a graph is *strongly connected* if there is an allowable path between every pair of vertices, and *weakly connected* if we could find a path by ignoring directions.

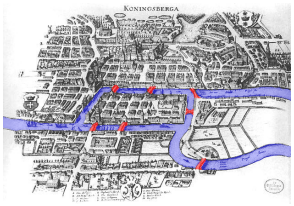
The definition is a simple one, and uses the idea of a path; a graph is connected if you can travel wherever you want from any starting place.

Connectivity

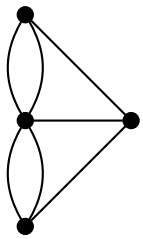
An undirected graph is **connected** if there is a path between every pair of nodes in the graph.

It turns out that we can even describe *how* connected a graph is, by seeing how hard it is to make it disconnected (in the example above, it was pretty easy to do, by simply removing one edge). But this simple description of connectivity is enough for us, because now we want to turn our attention back to the original problem: the bridges.

Euler Paths



Remember the setup: we want to find a way to travel through the city of Königsberg in such a way that we cross all of the seven bridges, but don't cross any more than once. In other words, we want to find a *simple path* through the following graph that uses every edge:



Just to clarify: a simple path means that we don't reuse edges, but we could leave some edges out. The additional requirement that we use *every* edge is what elevates this from a simple path to an *Euler path*.

Euler Paths

An **Euler path** in a graph is a path that goes through every edge of the graph exactly once.

Alternately, an Euler path is a simple path that contains every edge of the graph.

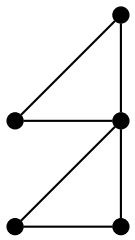
An *Euler circuit* is defined similarly: it is an Euler path that begins and ends at the same node.

Naturally, an Euler path is only possible in a connected graph.

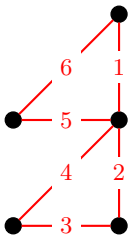
Now, the important question is, *how can we tell if an Euler path or circuit is possible?* If we know it's possible, of course, we'll also want to be able to find it.

It turns out that the answer to this question is surprisingly simple; there's something delightful about elegant answers like this one. Let's see what Euler discovered; we'll take a look at a few examples to see what we can learn.

Let's start with a simple one:



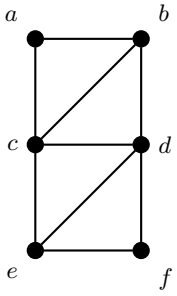
The goal is to trace over this graph, drawing every edge without lifting our virtual pen or retracing any lines. With a little trial and error, we can find an Euler circuit; for instance, if we start at the top node, we can trace every edge and end up back at the same place:



Notice that we can already draw one conclusion, in order to draw an Euler circuit, *we can't get stranded anywhere*. Now, what would it mean for us to get stranded? If we can find an edge leading to a node, but there aren't any unused edges that we can use to leave, we'll be stuck.

Observation 1:
to draw an Euler circuit, every time we arrive at a node, there must be an unused edge we can use to leave.

While you mull on that, here's another example:



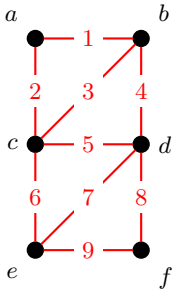
If we start at a, for instance, we'd have to also end there, because otherwise we'd have used up both edges that connect to it (one while leaving, and one while arriving) and have nowhere to go. In fact, this is true *whenever a node has an even degree*; leaving and returning uses up a pair of edges, so if you start at an even node, you have to end at the same node.

Observation 2:
if we start at an even node, we must end at the same node.

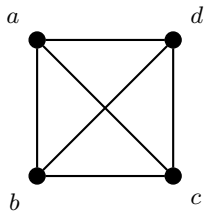
Now, notice that a, c, d, and f are all even nodes (only b and e are odd). If we started at any of these even nodes, we'd run into a problem, because we'd have to draw a circuit (see Observation 2). Look back at Observation 1: if we're drawing a circuit, we can't get stranded anywhere, which means we must be able to find an unused edge every time we arrive at a node. The first time we arrive at b, for instance, we'll have two unused edges to choose from, but when we return by the other one, we'd be stranded at b.

Okay, since we can't start at any of the even nodes, let's start at one of the odd ones (we'll pick b, but we could also use e). There are many options, but here's one possible route:

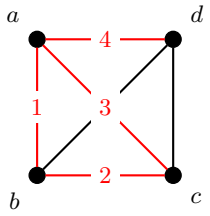
Observation 3:
if we start at an odd node, we must end at another odd node.



One final example (which happens to be K_4):



Since this one is symmetric, we can start at any node; there will be no difference if we pick another to start. Let's say we start at a:



Observation 4: if there are too many odd nodes, an Euler path is impossible.

After these first four edges, we arrive at d, and we're stuck; either way we go from here, we'll get stranded at either b or c. The problem is that there are too many odd nodes, so we end up running into one without anywhere to go.

Let's put together what we've observed from these three examples:

- The question hinges on whether nodes have even or odd degree.
- In the first example, all the nodes were even, and we could find an Euler circuit.
- In the second example, there were two odd nodes, and we could find an Euler path, but it had to start at one odd node and end at the other.
- In the third example, there were four odd nodes, and we couldn't find either an Euler path or an Euler circuit.

We can boil it down like this: for an Euler path/circuit, every time you arrive at a node, you need to be able to leave. The possible exceptions are the beginning and end, if you're only drawing a path, not a circuit. Arriving and leaving uses up two edges, so the nodes need to all be even for a circuit. For a path, there can be two odd nodes; one will be the starting point, and the other will be the ending point.

This sounds more complicated than it really is; we can write a simple rule to summarize everything we've observed.

Existence of Euler Paths and Circuits

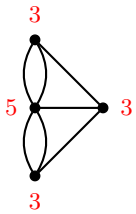
If all the nodes of a graph are even, the graph has an Euler circuit (it can start and end at any node).

If there are two odd nodes, there is no Euler circuit, but there is an Euler path (starting at one odd node and ending at the other).

If there are more than two odd nodes, there is no Euler path or circuit.

Note: if there is no Euler path, we could still try the Chinese Postman Problem (named in honor of the Chinese mathematician Kwan Mei-Ko, who first studied it. The goal of this problem is to find the shortest circuit that visits every edge; in other words, the circuit that reuses as few edges as possible.

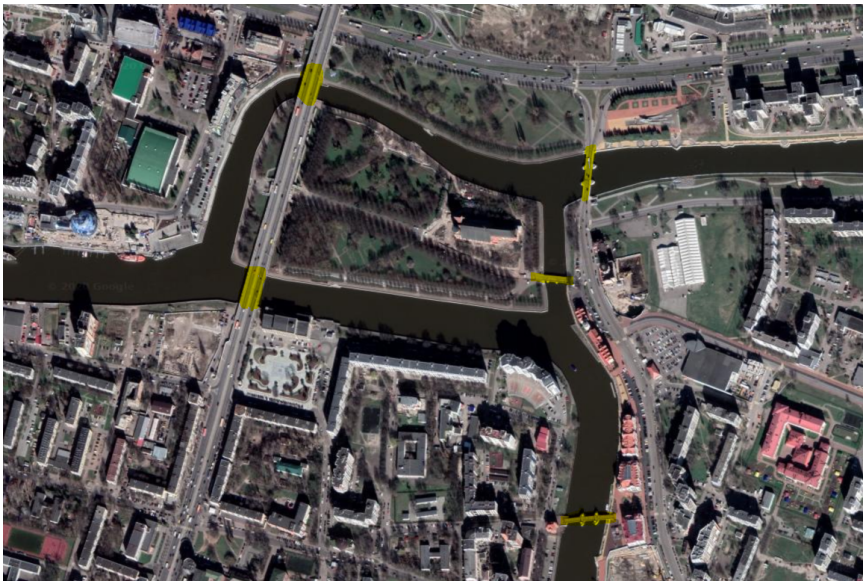
With that, let's go back to the bridges of Königsberg and see if there is an Euler path; let's label each node with its degree:



Since there are 4 odd nodes, there is no Euler path, so there's no way to travel through the city by crossing every bridge exactly once.

MODERN KÖNIGSBERG EXAMPLE 1

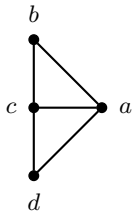
A modern image of part of Kaliningrad (which was once Königsberg) is shown below, with bridges highlighted.



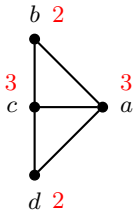
Is there an Euler path and/or circuit through this part of the city? If so, find one.

Solution

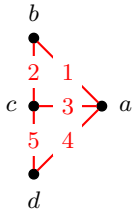
First, we need to abstract this map with a graph, keeping the nodes in the same positions as before, but redrawing the edges to match the current bridges:



Next, we can find the degree of each node:



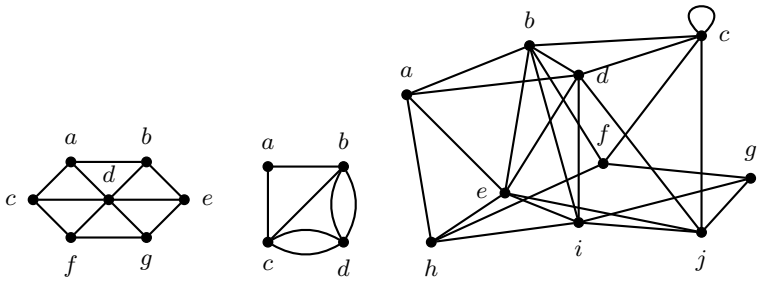
Since there are exactly two odd nodes, it is possible to find an Euler path through this graph, but not an Euler circuit. One Euler path is shown below:



This path can be written $\boxed{a \to b \to c \to a \to d \to c}$. Notice that it starts at one of the odd nodes and ends at the other.

EXAMPLE 2 EXISTENCE OF EULER PATHS

For each of the following graphs, determine if an Euler circuit exists. If not, determine whether there is an Euler path.



(a) (b) (c)

Solution

(a) Since all the nodes except d are odd, each with a degree of 3, there is no Euler path or circuit in this graph.

(b) All the nodes in this graph are even (a has degree 2, the rest have degree 4), there is an Euler circuit for this graph.

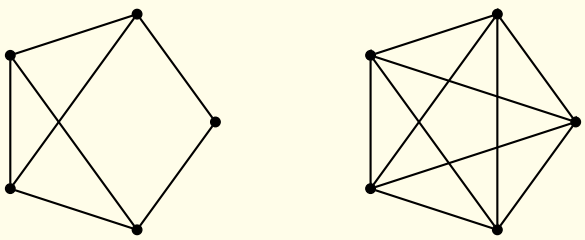
(c) Counting the degrees of each node for this graph is a bit more tedious, but the principle is the same. The results are below (remember that a loop contributes 2 to the degree of its node):

Node	Degree
a	4
b	6
c	6
d	6
e	6
f	4
g	3
h	4
i	6
j	5

Since there are exactly two nodes (g and j) with odd degree, there is an Euler path (but not an Euler circuit) for this graph.

TRY IT

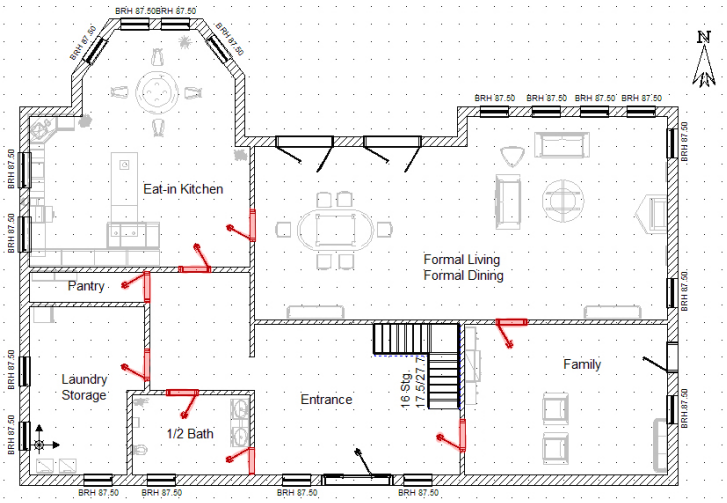
For each of the following graphs, determine if an Euler circuit exists. If not, determine whether there is an Euler path.



(a) (b)

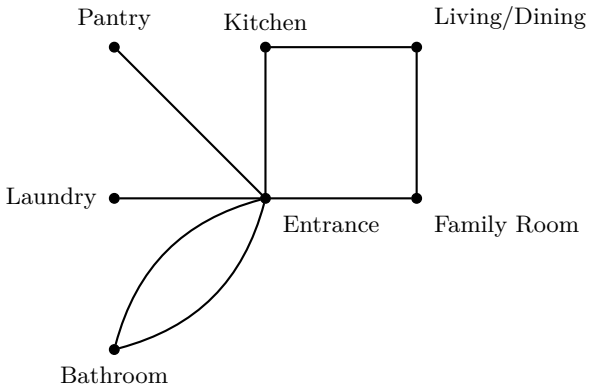
EULER PATH THROUGH HOUSE — EXAMPLE 3

The floor plan below shows the first floor of a single-family home. Is there an Euler circuit/path through the interior of this level, using the highlighted doors (in other words, ignoring external doors and stairs)?



Solution

First, let's draw a graph to represent this home, with each node representing a room and each edge representing a door. Since the entrance is the most central location (in terms of connections to the rest of the home, we'll place that in the center, with edges out to the other rooms:



Now, we can count the degree of each node:

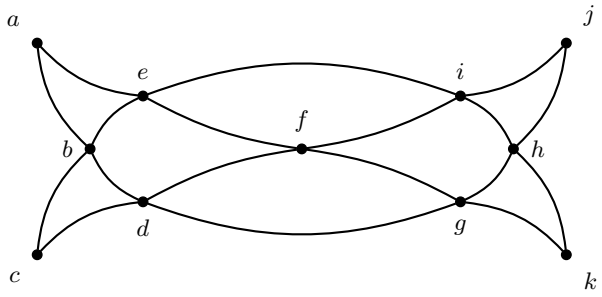
Room	Degree
Entrance	6
Family Room	2
Living/Dining Room	2
Kitchen	2
Pantry	1
Laundry	1
Bathroom	2

Since there are two rooms with an odd number of doors, there is an Euler path, but not an Euler circuit through this level. Any Euler path must start in either the pantry or laundry room and end in the other.

Finding an Euler Circuit

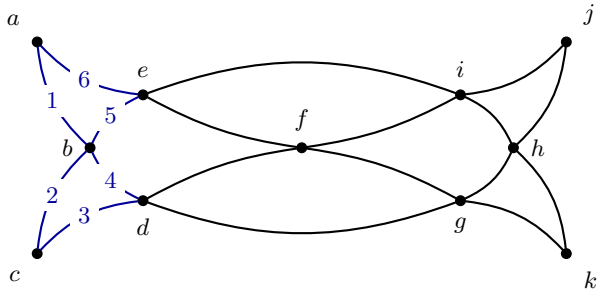
It's fairly simple to determine whether or not an Euler circuit exists, by simply identifying the degree of each node. Once we know that there *is* such a circuit, it is often simple enough to find what it is just by looking at the graph and tracing through it, as long as the graph is relatively small.

However, here we'll illustrate a more systematic process for finding an Euler circuit, in case it is useful. We'll use the graph below as an example.

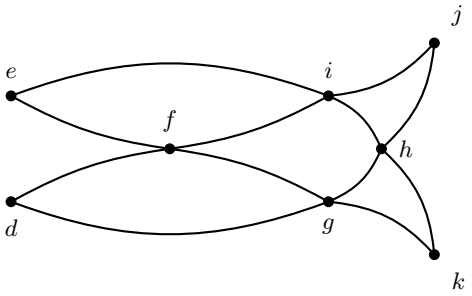


Notice that all the nodes are even, so it is possible to find an Euler circuit (or many, indeed). We can start at any point, and eventually come back and end at the same node.

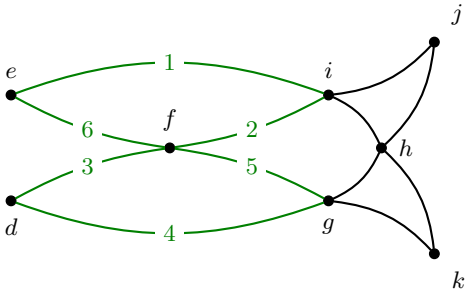
To begin, pick a starting point, and construct ***any*** circuit through the graph. Let's say we start with a; we could do something as simple as $a \to b \to e \to a$, but the longer our initial circuit is, the faster this process will go, so let's start with $a \to b \to c \to d \to c \to e \to a$:



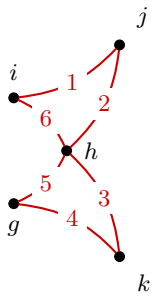
Now, delete those edges from the graph, and remove any points that are left isolated (so for instance, a will be deleted, but e will not, because there will be some remaining edges connected to e).



Then pick another point (we'll use e) and repeat this process. Say, for instance, we pick the next circuit to be $e \to i \to f \to d \to g \to f \to e$:



After removing those edges and isolated points, we can find one final circuit through the remaining points: $i \to j \to h \to k \to g \to h \to i$:



Having used all the edges now, we have three partial circuits:

$$a \to b \to c \to d \to c \to e \to a$$
$$e \to i \to f \to d \to g \to f \to e$$
$$i \to j \to h \to k \to g \to h \to i$$

To construct the final Euler circuit through the full graph, we simply stitch these three circuits together. Start from the end: notice that the last one begins and ends with i. Then, go to the one just before that: when that second circuit passes through i, we can pause and run through the third circuit before resuming. In practice, just replace i with the third circuit; instead of

$$e \to i \to f \to d \to g \to f \to e$$

we will now have

$$e \to i \to j \to h \to k \to g \to h \to i \to f \to d \to g \to f \to e$$

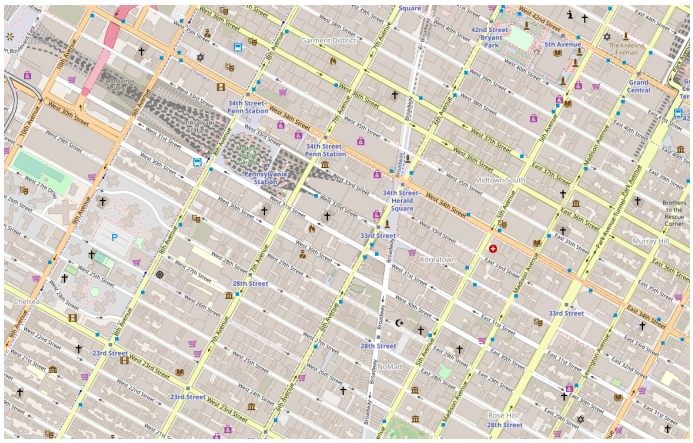
Finally, notice that this long string begins and ends at e, so we can substitute the whole thing in the place of e in the first circuit, and we'll have our full Euler circuit:

$$a \to b \to c \to d \to c \to e \to i \to j \to h \to k \to g \to h \to i \to f \to d \to g \to f \to e \to a$$

Again, in many cases there's no need to resort to a systematic process like this, but if you have trouble spotting an Euler circuit immediately, you can try it.

Hamilton Paths

Take a look at the map below; what do you see?



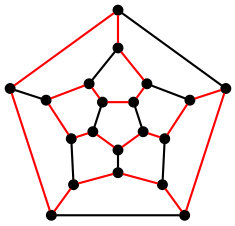
By now, you can probably see a graph; if we designate each intersection as a node, the edges would be one-block segments of streets.

Now, suppose you work for the city's public works department, and your job is to schedule workers throughout the city as efficiently as possible.

After a snowstorm, your job is plan routes for the snowplows to get the streets cleared with a minimal amount of wasted driving. This is, in fact, an Euler path problem (or more likely a Chinese Postman problem, if an Euler path doesn't exist). Having read this chapter, you draw up a hyper-efficient plan and save the city thousands in fuel.

Although William Rowan Hamilton was not the first one to study these paths, they are named for him because he invented a puzzle called the Icosian game, which starts with a 12-sided *dodecahedron*:

The goal is to find a Hamilton circuit, traveling along the edges and touching each corner exactly once. Hamilton sold this puzzle to a game dealer, and it was marketed throughout Europe. In the most popular form, the dodecahedron was modeled as a planar graph, like this one:



It was sold as a wooden board with pegs at the nodes of the graph, and players would wind a string along a path to solve it (one solution is shown in red).

A few months later, the city decides to switch out all the traffic lights for a new, more efficient, more reliable model. Since you did such a great job with the snowplow scheduling, the department tasks you with planning the routes of the work crews through the city for this new contract. What's different about this problem?

In the snowplow problem, the goal was to *travel along every* ***edge*** exactly once. Now, with the traffic light problem, the edges are not the important part: there's no need to drive along every road, but simply to reach every intersection so that the crew can do their work there. Thus, the goal of the traffic light problem is to *travel to every* ***node*** exactly once. This is an entirely new kind of problem, and what we're looking for now is called a **Hamilton path**.

Hamilton Paths

A **Hamilton path** (or Hamiltonian path) in a graph is a path that passes through every node of the graph exactly once (and a Hamilton circuit is a Hamilton path that happens to be a circuit).

Now, having seen how easy it is to determine whether a graph has an Euler path or circuit, you may be expecting a similar rule for Hamilton paths. However, it turns out that there is no known rule that can tell us for certain whether or not any graph has a Hamilton path or circuit.

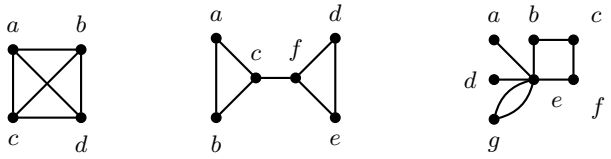
There are a few observations that we can make:

1. Every complete graph has a Hamilton path, and as long as there are at least 3 nodes, there is a Hamilton circuit. Since there are edges between every pair of nodes, we can visit all the nodes in any order we choose.
2. If a graph has a node with degree 1, it cannot have a Hamilton circuit, because there's no way to both arrive at and leave that node. It could, of course, have a Hamilton path.
3. When drawing a Hamilton path/circuit, once you have marked a node, you can eliminate all the other edges that connect to that node, which can simplify the picture.

There are other, more complicated theorems that give conditions under which Hamilton circuits exist (for instance, *Dirac's Theorem* says that a simple graph with n nodes, where $n \geq 3$, has a Hamilton circuit if the degree of every node is at least half of n), but for our purposes, we will simply check whether a graph has a Hamilton path or circuit by inspecting it and seeing if we can find one. Since we'll only deal with small graphs, this is good enough for us. For larger graphs, we could use a *brute force* approach, which means trying all the possible paths to see if any is Hamiltonian (as you can imagine, this is quite tedious).

EXAMPLE 4 HAMILTON PATHS

For each of the following graphs, determine whether a Hamilton circuit exists; if so, describe the circuit. If there is no Hamilton circuit, see if there is a Hamilton path.



Solution

(a) Since this is a complete graph, K_4, we know that a Hamilton circuit exists. We could trace, for instance, the path $a \to d \to c \to b$.

(b) There is no Hamilton circuit, because c and f form bottlenecks; once you pass through one of them to go to the other side of the graph, there's no way to return. However, we can find a Hamilton path; $a \to b \to c \to d \to e \to f$, for instance.

(c) We know that there is no Hamilton circuit, because we have nodes with degree 1. Say we start at a; as soon as we travel to e, we're trapped, because no matter where we go, we'll cut off several points without any way to get to them without going back through e. Therefore, there is no Hamilton circuit or path for this graph.