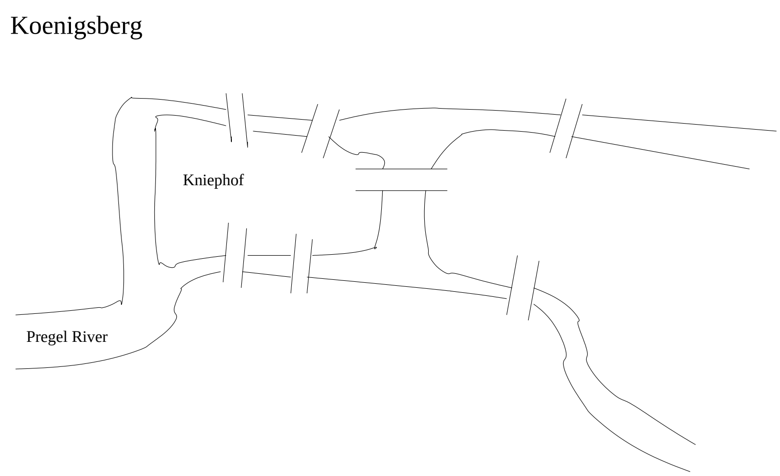
Graph Theory, Part 1

1 The seven bridges of Königsberg

1.1 The Problem

The city of Königsberg (formerly in Prussia, now a part of Russia and called Kaliningrad) is split by the River Pregel into various parts (including the island Kniephof), and back in the day there were seven bridges connecting the various parts, as you can see in the map below:

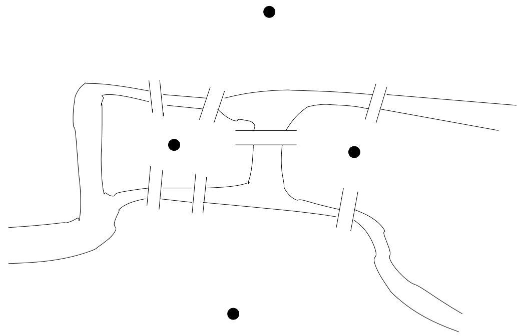


The city-folk wanted to know if it was possible to make a walking tour of the city, starting and ending at the same place, in such a way that the walkers crossed each bridge exactly once. This was considered a difficult problem at the time. After all, there are many different routes you might consider, and it would be tedious to check all of them. Of course, if you find something that works, you are done. But people couldn't seem to make anything work, and so the question arose whether there was not something inherent in the problem that made it insoluble.

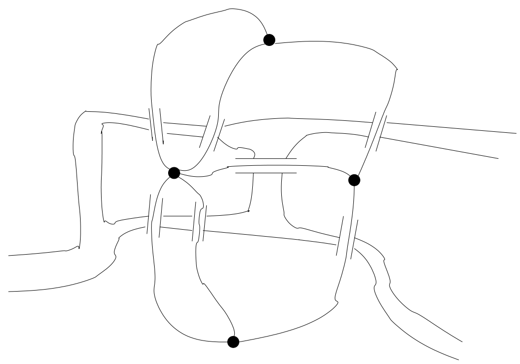
In a 1736 paper which is considered the birth of a branch of math known as *Graph Theory*, the great mathematician Leonhard Euler showed decisively that it is *impossible*. His idea is so stunningly clear that it sets the whole matter to rest.

1.2 Abstraction

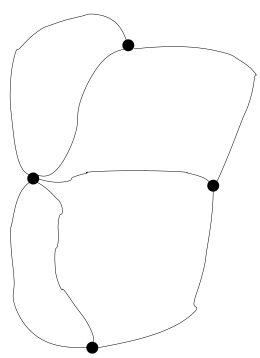
To help us deal with the problem more readily, let us *abstract* from the problem only the relevant information. Basically there are four pieces of land and various bridges connecting them. We don't care what shops you visit or things you go to see on the way when you are not crossing bridges—for us each of the four pieces of land is just a place with bridges to some of the other pieces of land. Let's draw a dot for each of the four pieces of land. Visiting the each piece of land is for us the same as visiting the dot in that land.



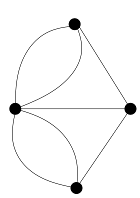
Now let's make connections between the dots, with one connection for each bridge:



We can now remove all the burdensome geographical details:



And draw the dots and connections more neatly. All we are concerned with is the connectivity of the lands by the bridges, not exactly how far or in what direction you need to walk:



Now the problem is to go from dot to dot using each bridge only once. Suppose we start on the rightmost dot. Then when we leave it, we use one of the three bridges, and so there are still two bridges attached to the rightmost dot which we eventually must use. The only way to use another is to return to that dot. When we do, we use another bridge, and are left with one bridge to use. Well then, we must leave that dot again, and use the last of the three bridges attached to it. But then there is no bridge with which to come back, and we are supposed to do a round trip.

OK. Maybe we shouldn't start the trip at the rightmost dot. We will start somewhere else. Nonetheless, we do need to visit the rightmost dot at some point in order to make use of bridges attached to it. On our first visit, we use one bridge coming in. Then we must use another, and so we leave. But there is still one more bridge attached to the right dot that has not been used. So we need to return again, but then we can't leave! This is bad, because now our trip is a round trip which didn't start at the rightmost dot.

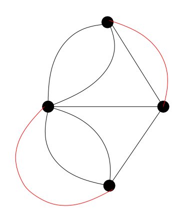
In fact, it is not hard to show that *every* dot (piece of land), whether it is the one you start and end at or not, *must* have an even number of bridges attached, otherwise you will have problems. You use an even number of bridges when you come to a place and then leave again. For dots other than your starting dot, you must always leave after coming, so you always use up a pair of bridges per visit. So there must be an even number of bridges. For you starting point/destination dot, you use a pair of bridges for each passing visit, but also one bridge to leave at the beginning and one bridge to return at the very end. So it should also have an even number of bridges. In Königsberg, *every* land has an odd number of bridges attached, so it is hopeless.

In honor of Euler, who showed that this cannot be done, a path that uses every bridge once is called a *Eulerian circuit.* Königsberg has no Eulerian circuit because it has regions with odd numbers of bridges.

1.3 A Modification

OK. So we are definitely ruined unless *every* dot has an even number of bridges attached. But might we still have problems even if we do have an even number of bridges for every land?

For instance, suppose we build two more two bridges in Königsberg so that each dot has an even number of bridges:

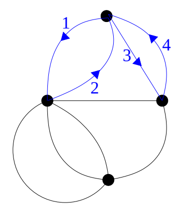


Now is it possible to make an Eulerian circuit? You can work out that it is indeed possible. Is this a general rule for all graphs? It turns out it is.

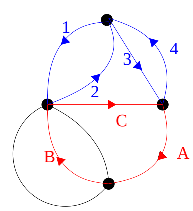
Theorem 1 (Euler, Informal Version). *It is possible to make an Eulerian circuit if and only if all vertices (dots) have an even number of edges (bridges) [and the whole system is connected—if one part is completely cut off from another, it is of course hopeless!].*

We have proved that you cannot make an Eulerian circuit if you do not satisfy the "all-even" condition. How can you prove than you *can* make an Eulerian circuit if you do satisfy the "all-even" condition? The idea is as follows: start off wherever you want and walk over bridges as you please with the only rule being that you never use the same bridge twice. Eventually you will get stuck since bridges keep getting used out. If you (by chance) managed to make an Eulerian circuit, then you are done!

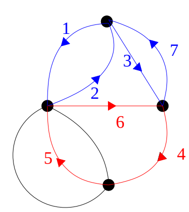
But maybe you didn't. For instance, in the last graph above, you might have just started at the top dot, walked to the left dot and then back again, and then walked to the right dot and back again. In this case, you would have used all four bridges from the top dot, namely:



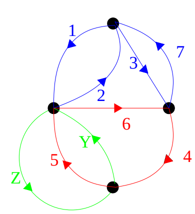
and you can no longer leave! But many bridges are left untraveled, so it is not an Eulerian circuit. What you do in this case is you go to your partly completed circuit, and pick a dot whose bridges are not all used—for instance the right-hand dot. Then start another walk there, where you only use bridges that haven't been used yet:



After traversing A-B-C, we are stuck again at the rightmost dot, because we have used all the bridges from that dot. We combine the two tours, by going on the first one (1-2-3-4) until we get to the departure point of the second tour (the right-hand dot). Then we make the second tour (A-B-C) a side trip, and after we finish the side trip, we complete the original tour. In this case, that means we start the original tour 1-2-3, then take the side trip A-B-C, and then use 4 to finish the original tour. So we now have a bigger tour 1-2-3-A-B-C-4, which we relabel 1-2-3-4-5-6-7.



There are still two bridges unused. between the left and bottom dot. So we make those two bridges into a little tour Y-Z, and then combine it with our existing tour 1-2-3-4-Y-Z-5-6-7.



Each of the mini-trips is a circuit, so it uses an even number of bridges for each dot visited. So what is left of the original graph at each stage after adding the side-trip is a smaller number of unused bridges, with an even number of unused bridges for each dot (because an even number of bridges minus an even number of bridges used is equal to an even number of bridges remaining). So we can always continue adding side-trips until all the bridges are used!

2 The Basics

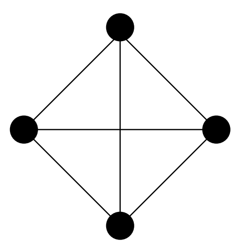
2.1 What is a Graph?

The diagrams which we used to represent the Königsberg problem economically are mathematical objects called *graphs*. Euler's paper on the Königsberg problem is considered the birth of the branch of mathematics now known as *Graph Theory*.

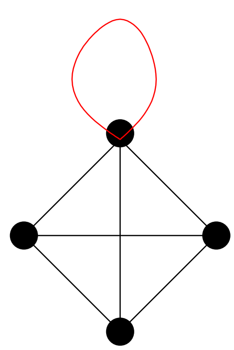
Let's make some definitions.

Definition 1 (Graph). A *graph* is an assembly of two kinds of things, *vertices* and *edges*. The only rule is that each edge starts at a vertex and ends a vertex.

We usually write vertices as dots and edges as lines between the two vertices they join. We do not need to draw the edges as straight lines, and in general we will not be able to avoid drawing pictures where the edges displayed seem to touch each other. In fact, such edges should not be considered to touch each other, but you can imagine that one goes over the other, much like a highway overpass. This is why we always draw the vertices as moderately-sized dots—so that we are not confused into thinking that the crossed edges in



meet at some vertex. Since there is no vertex-dot, they do not. Note that in our Königsberg problem, we had more than one edge connecting a pair of vertices, to represent multiple bridges that accomplish the same thing. It is even possible to have an edge connecting a vertex to itself; this is called a *loop*. Here we have added a loop to the last graph:



Sometimes it is important to assign directions to the edges on a graph, and so we define a graph with this additional information:

Definition 2 (Directed Graph). A *directed graph* is a graph where we have put arrows to assign directions to *all* the edges.

For example, the last graph which designates our Eulerian circuit in the modified Königsberg bridge problem is a directed graph.

2.2 Degree

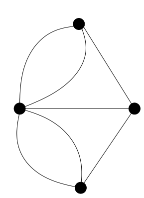
In the Königsberg bridge problem, we were searching for a Eulerian circuit, which can now be defined:

Definition 3 (Circuit, Eulerian Circuit). A *circuit* in a graph is a path which begins and ends at the same vertex. A *Eulerian circuit* is a circuit in a graph which traverses each edge precisely once.

We found that a Eulerian circuit exists if and only if each vertex (representing a piece of land) has an even number of edges (bridges) attached to it. We coin a new term to make it easier to speak of such conditions:

Definition 4 (Degree). The *degree* of a vertex is the number of edges attached to that vertex.

For example, if we return to the Königsberg bridges graph,



then we see that the degrees of the vertices (starting at the top an proceeding clockwise) are 3, 3, 3, and 5. Now we can state Euler's result succinctly:

Theorem 2 (Euler, Formal Version). *A graph has a Eulerian circuit if and only if all vertices have even degrees.*

Thus the Königsberg bridges graph has no Eulerian circuit.

2.3 Distance and Diameter

Definition 5 (Distance). The *distance* from vertex u to vertex v is the minimum number of edges one must traverse to get from u to v.

In the undirected Königsberg bridges graph vertex u is distance 1 from all other vertices, but vertex v is distance 2 from vertex x. We have a special name for vertices which are at distance 1 from each other.

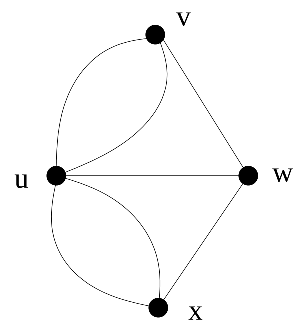
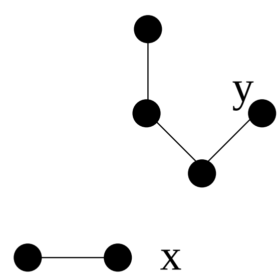


Figure 1: (Undirected) Königsberg Bridges Graph

Definition 6 (Adjacency). Two vertices in a graph are said to be *adjacent* if they are joined by an edge.

If one vertex cannot be reached from another by a path made of edges, then we say that their distance from each other is ∞. If this happens, it means that the graph is not connected. For instance the graph

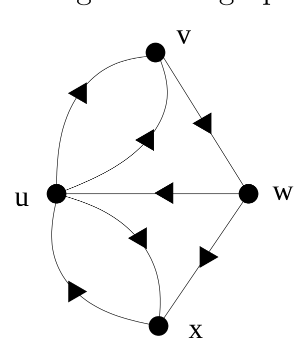


has the distance from vertex x to vertex y equal to ∞, and so the graph is not connected. On the other hand, the Königsberg bridges graph (see Figure 1) is connected.

NOTE: We shall ALWAYS assume our graphs are connected unless explicitly noted otherwise.

When we label the vertices of a graph, as we have done in the graph of Figure 1, we also obtain a compact notation for labeling edges. For example, the edge connecting u to v is called $\overline{uv}$. Note that $\overline{uv} = \overline{vu}$, since this is an undirected graph, so that the order of the vertices does not matter.

In a directed graph, we only count paths that follow the one-way signs attached to the edges. For example, if we assign directions to edges in the graph of Figure 1:



then the distance from w to x is 1, but the distance from x to w is 3. So distance is not symmetric in a directed graph, as anyone knows who has driven in a city with lots of one-way streets!

In a directed graph, we label edges in a way that indicates which way the arrow goes. For example, in the above graph, we have edge $\overrightarrow{wx}$, also known as $\overleftarrow{xw}$, but we DO NOT have the edge $\overrightarrow{xw} = \overleftarrow{wx}$.

Definition 7 (Diameter). The *diameter* of a graph is the maximum of the distances between pairs of vertices.

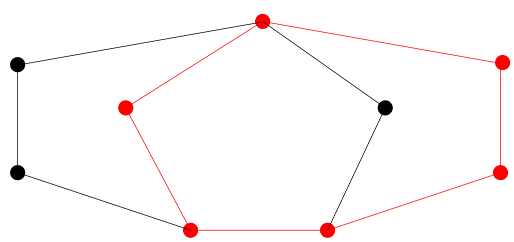
The undirected Königsberg bridges graph has diameter 2 (because every vertex is only distance 1 or 2 from the others), while an octagonal graph has diameter 4 (draw it and find the longest distance).

2.4 Polygons, Girth, and Trees

The octagonal graph we just mentioned is one member of an important family of graphs. Let us consider undirected graphs for the time being.

Definition 8 (Polygon). A *polygon* is a sequence of edges and vertices where no edge or vertex is visited more than once, with the exception that the first vertex and the last coincide. [Note that Eulerian circuits need not be polygons: although they never re-use edges, they do not always meet the restriction on multiple visits to vertices.] The *length* of a polygon is the number or edges in the polygon.

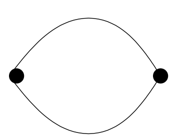
The following graph has a polygon of length 6 in red.



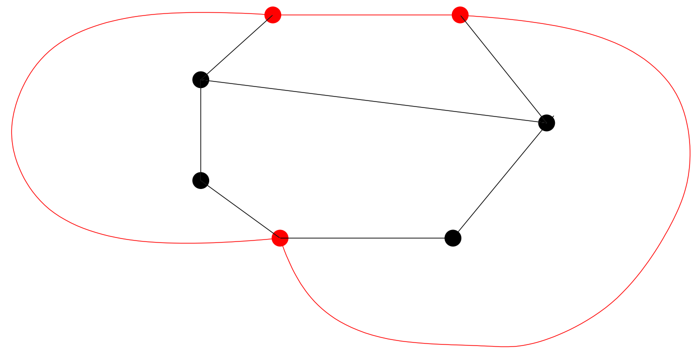
The graph also has polygons of lengths 5 and 7, but none of length less than 5, nor any of length greater than 7. This motivates the next definition:

Definition 9 (Girth). The *girth* of a graph is the length of the shortest polygon in the graph.

A graph of girth 1 must have a loop. A graph of girth 2 must have no loops, but must have two vertices connected thus:



We call this strange object a 2-*gon*. A graph of girth 3 must have no 2-gons, nor loops, but must have three vertices connected like a triangle (although they may be drawn with curvy edges in our drawing of the graph). For the purposes of graph theory, we *do* call such a structure a triangle, however it may be drawn. For example, the following graph has a triangle in red:

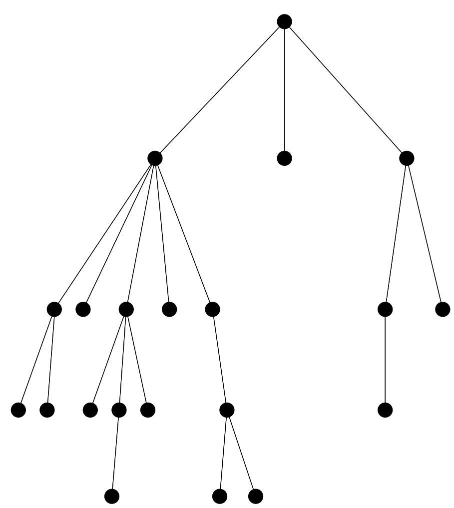


Once we allow this latitude in the use of the terms "triangle," "square," pentagon," and so on, we see that a graph of girth 4 is free of loops, 2-gons, and triangles, but does contain a square.

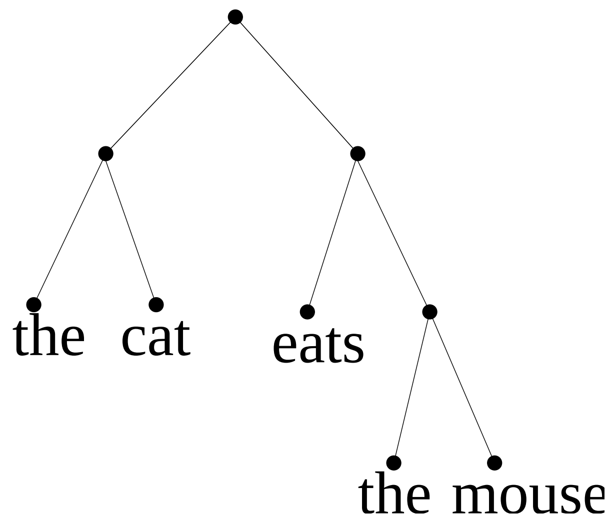
What if the graph has no polygons at all? Then we define the girth to be ∞. We have a special name for such graphs.

Definition 10 (Tree). A *tree* is a graph containing no polygons.

So a tree has no loops nor 2-gons, so that every pair of vertices is either not joined or joined by a single edge. Trees are widely used in such fields as biology, computer science, and linguistics in organizing data. It is common to draw trees to look like upside-down versions of the trees you see in the real world:



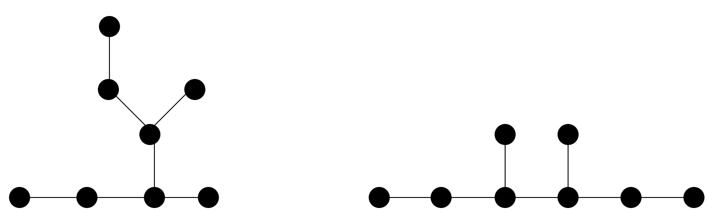
Here is a tree that represents the parsing of an English sentence:



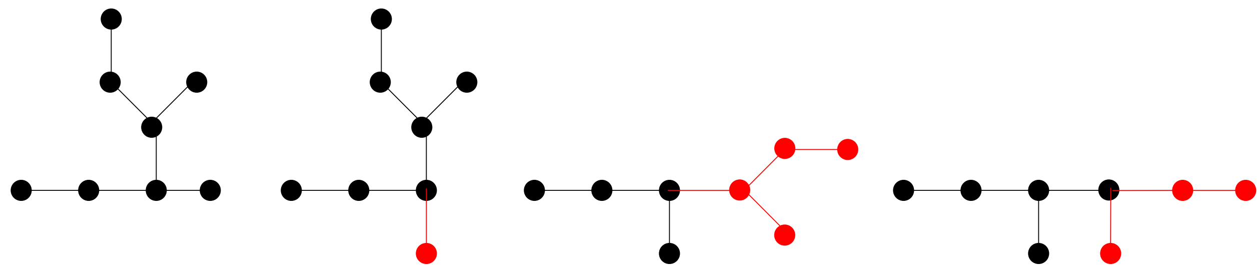
3 Isomorphism of Graphs

Two graphs G and H will be called *isomorphic* if they have the same pattern of connectivity. By this we mean that you can convert G into H by moving the vertices of G around without changing which vertices the edges are connected to. You might also need to bend the edges into new shapes, but without changing which vertices they join. Don't worry if you need edges to move over other edges and vertices when making these changes.

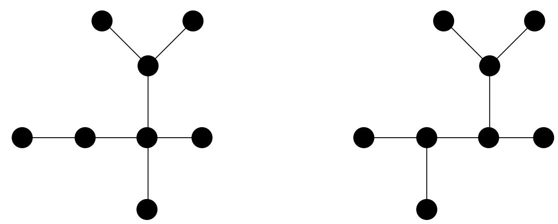
For an example of where isomorphism is important, we turn to chemistry. Hydrocarbons known as alkanes may be represented by trees in which each vertex represents a carbon atom. The edges represent bonds between carbon atoms. A carbon can be bonded to at most four other carbon atoms, so the maximum allowed degree of a vertex is 4. (The hydrogen atoms are not depicted in these graphs, but a chemist automatically knows where to put them if shown the graph of the carbon atoms.) The following two forms of octane (alkanes with 8 carbon atoms) are actually isomorphic:



We show the movements of vertices that make the first graph look like the second. We color red the elements that are moved in each step. We should rotate the final figure by 180 degrees to finish the process.



The following two graphs are not isomorphic, because one has a vertex of degree 4, while the other does not:



4 The Chinese Postman Problem

The Chinese Postman Problem is a generalization of the Königsberg bridges problem. A Eulerian circuit exists only in graphs where all degrees are even. If not, we might ask ourselves, "What is the next closest thing to a Eulerian circuit?" This could be answered in several ways: the one we shall propose is adumbrated in the exploration where we constructed extra bridges in Königsberg in order to make all degrees even, thus enabling a Eulerian circuit to exist.

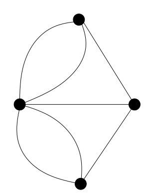
What we seek is a circuit that uses each edge once, and perhaps some edges more than once, but we would like to reuse edges as few times as possible. That is, we want a circuit employing all edges but of minimum total length (as measured in number of edges traversed). This is not a pointless quest: suppose we need to deliver mail, or sweep streets, or pick up garbage. Each segment of street between intersections could be represented as an edge, and the intersections as vertices. Then our insistence that each edge be traversed at least once is the insistence that all houses in town receive services; retracing is tolerated as needed to make a full circuit, but no more than necessary. Of course, in a real-world situation, we would note that different segments of streets are of different length, and we would try to minimize the total *length* (in meters) of street segments retraced, rather than the total *number* of segments of streets retraced. This is more complicated, so in these notes we shall stick to the simpler version, where we just try to minimize the number of edges in our circuit. In the homework, the more general problem will be discussed. This more complicated problem is called the *Chinese Postman Problem*, after the Chinese mathematician Meigu Guan who studied it, and our easier version is called the *Simplified Chinese Postman Problem*.

Using an edge $\overline{uv}$ of a graph G twice in a circuit is much like having two copies of the edge and using each copy once. In fact, let us make an extra copy of $\overline{uv}$ for each time we use it after the first traversal. We then use the duplicates instead of the originals, so that no edge is traversed multiple times. And let us do the same for all other edges, add a duplicate for every use beyond the first and traverse duplicates so as to avoid double-use of any edge. What we see is that a circuit in a graph G that traverses all edges at least once and which includes n

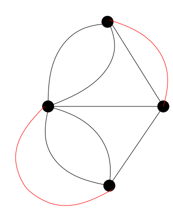
additional traversals of edges already used is equivalent to a Eulerian circuit in a graph obtained by adding n extra edges to G, where the added edges are copies of various edges existing in the original graph G. Thus a minimum-length circuit that uses all edges in G is a Eulerian circuit in graph H obtained from G by duplicating as few edges as possible. We know that a Eulerian circuit will exist in H if and only if all degrees of vertices in H are of even degree. So our goal is to add duplicate edges to G—as few as possible—to make a graph H with all degrees even. This prompts a definition.

Definition 11 (Eulerization, Minimal Eulerization). A *Eulerization* of a graph G is a graph H, all of whose vertices are of even degree, obtained from G by duplicating edges. A *minimal Eulerization* of G is a Eulerization of G with as few extra edges as possible.

So what we are looking for in the Simplified Chinese Postman Problem is a minimal Eulerization of a given graph. For example, the Königsberg bridges graph



has all four vertices of odd degree. With each edge we add, we can at best hope to change two vertices to even degree. So we must add at least two edges. Thus, our modified Königsberg bridges graph

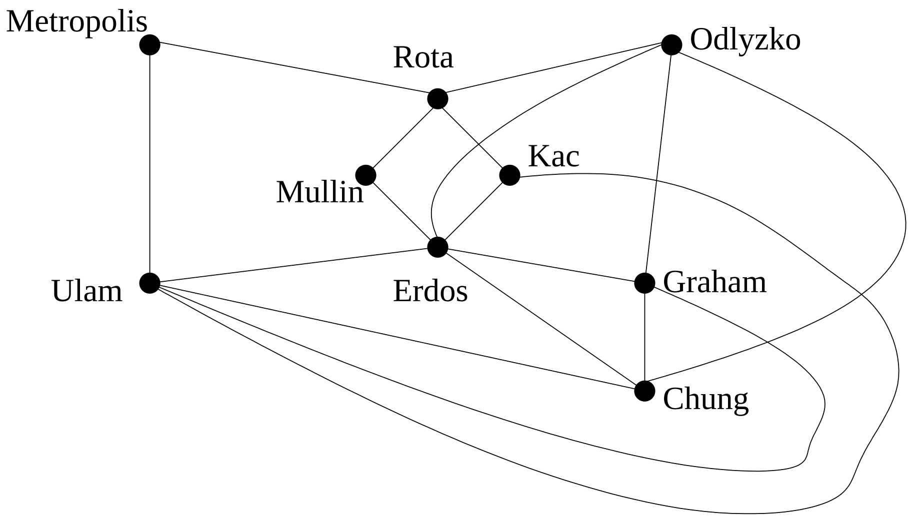


which duplicates two edges, is a minimal Eulerization of the original graph.

Is it always possible to Eulerize a graph? Yes! A Eulerization that is probably not very efficient is one that adds precisely one extra copy of each existing edge. This will definitely make all degrees even. This also shows that the postman certainly does not need to more than double the length of the streets serviced.

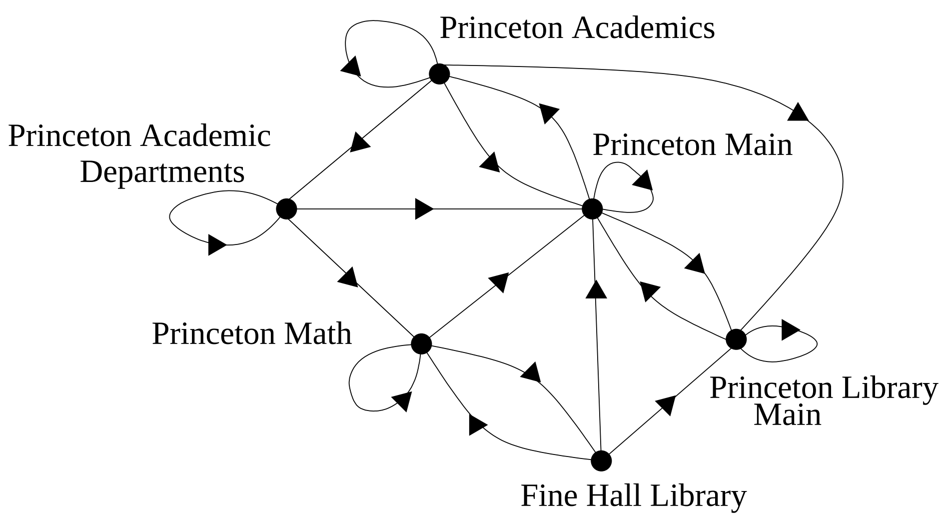
5 Graphs as Models of Adjacency

Graphs are used as a basic mathematical model for showing adjacency or any other relationship that occurs between pairs of individuals. For example, one graph has all the world's mathematicians as vertices. Two vertices (mathematicians) are joined by an edge if they are co-authors of a joint paper. One particular mathematician, Paul Erdös, is famous for having co-written with so many people, that most seasoned mathematicians are not far in distance from Erdös in the co-authorship graph. Here is a tiny portion of the authorship graph



The distance in the graph between a mathematical author and Erdös is given a name, "the Erdös number." Erdös has an Erdös number of zero, and people who coauthored with him are joined to him with an edge in the graph, so they have Erdös number one. People who wrote with someone who wrote with Erdös have Erdös number two. And so forth.

We could also graph the internet by making a site for each vertex. In our graph, an edge between sites indicates a link. But internet links have a direction—Site A can have a link sending you to Site B, but Site B may not have a link back to Site A. Thus, we should use a directed graph to represent the links on the internet.



5.1 Small World

When we start to look at large graphs like the coauthor graph or the internet graph, we often find what is called the *small world* phenomenon. The first issue is one of connectedness. Since people can make personal webpages that are unlinked to anything, we shall not always be able to get from one page to another by following links. However, in these large graphs it is often the case that there are many tiny isolated components, like little islands, along with one gigantic connected component, like a huge continent.

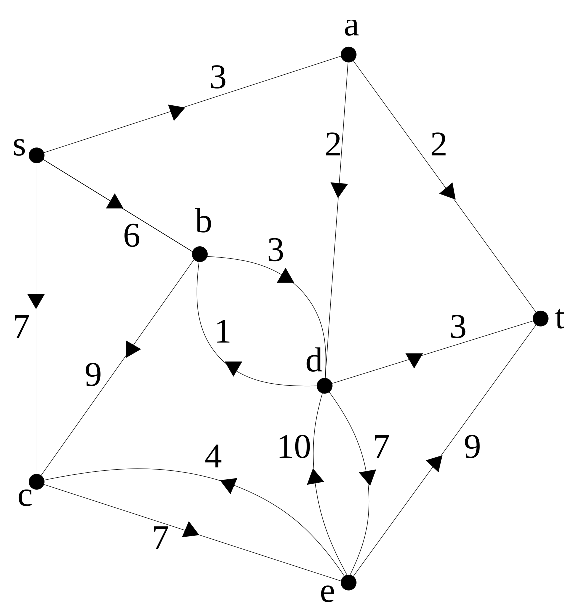
The second issue deals with how closely connected things are in this giant component. This is characterized by the diameter of the graph—the maximum distance you need to go to get from one vertex to another. Around the year 2000, people estimated that there were about 800 million pages in the web. However, researchers from Notre Dame performed an automated survey of a portion of the internet and concluded that the diameter is about 19, i.e., any two pages in the giant connected component are 19 links apart or less.

6 Flows in Networks

One type of problem that graph theory has been used to model and address is the problem of flows in networks. A network could be a road system, and the flow could be the movement of cars from one place to another on the roads. Or a network could be a computer network, with data flowing from one computer to the next. Or a sewer system, or an electrical distribution grid, and so forth.

6.1 The Model

We shall model our networks as directed graphs, and on each edge we shall write a number, which is the *capacity* of that edge. The capacity determines how much stuff (cars, data, sewage, energy) can flow along that edge in a fixed unit of time. Thus small roads or small pipes will have low capacities, and big pipes or big roads will have large capacity numbers. We shall always make our capacities integers. For now, we shall be interested in the somewhat restricted problem of moving all the stuff from one point (called the *source*) to another (called the *target*). We label the source s and the target t. For convenience, we shall always assume that no edges point into the source s or out of the target t. Here is an example of a network, with source, target, and capacities:

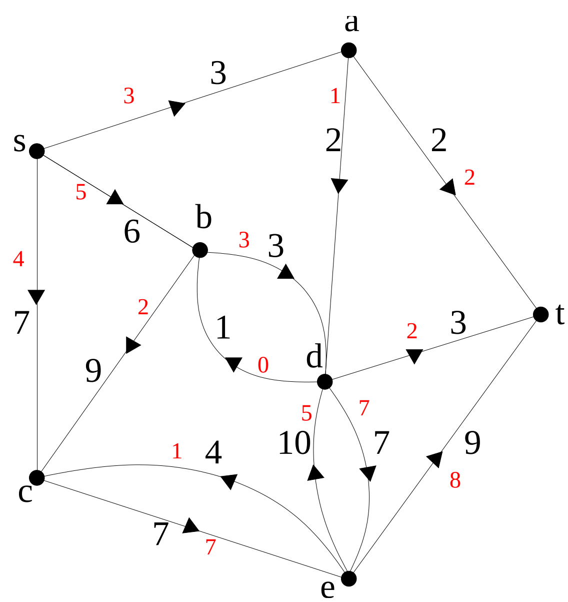


Suppose that this is a water distribution system, where the edges are pipes, the vertices are pumping stations, the direction of a given edge indicates which direction our pumps are set up to move water, and the capacities are the maximum number of cubic feet per second that can be moved through the various pipes. Our goal will be to move as much stuff from s to t as possible, with the following three constraints:

1. We may only move water through a pipe according to the direction it has.
2. We cannot move more water through a pipe than the capacity dictates.
3. The net flow into a pumping station must be exactly equal to the net flow out (otherwise we would accumulate water or have a deficit, but we don't have storage tanks in our system). The only stations exempt from this rule are the source s and the target t.

To respect these rules, we define a *feasible flow* to be an assignment of numbers to the edges such that each number is less than or equal to the capacity (to respect rule 2) and such that each vertex is *balanced*, meaning the sum of the numbers that we have assigned to edges pointing into the vertex is precisely the same as the sum of the numbers on the edges coming out of the vertex. This balance condition respects rules 1 and 3.

Here is an example of a feasible flow. We write our actual assignments in smaller type and in red—note that they are all less than or equal to the capacities, and check the balance of each vertex (except s and t):



For an example of checking balance, look at the lowest vertex, which is labeled e. Edges pointing into it have assigned flows 7 and 7, so the total inflow is 14. Edges pointing out of it have assigned flows 1, 5, and 8, so that the total outflow is also 14. Hence no water is accumulating here.

Notice that the total flow out of the source is $3 + 5 + 4 = 12$, and that the total flow into the target is $2+2+8 = 12$ also. No matter how tortuous the routing, it is inevitable that these two numbers match since we do not let water accumulate at any other station.

Definition 12. The *net flow from s to t* in a feasible flow is equal to the sum of the flows on edges pointing out of s. In a feasible flow, this must be the same as the sum of the flows on edges pointing into t.

6.2 Increasing Net Flow: Augmenting Paths

Our hope is to maximize the net flow from s to t. We note that the above assignment is not optimal. Here is one way we could improve it:

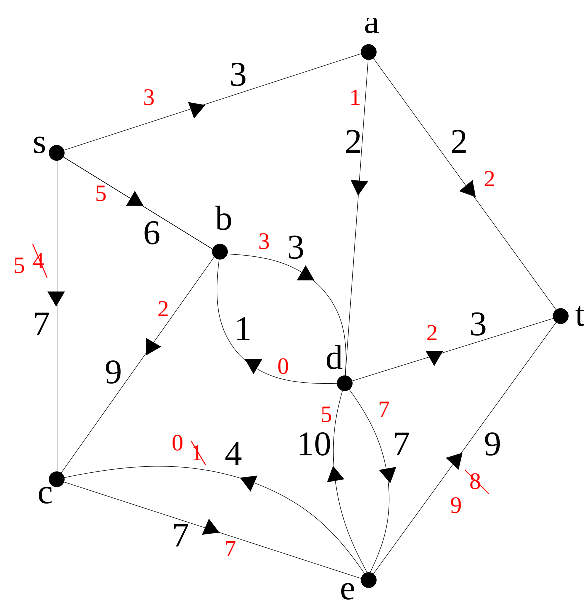
1. We need to get more water out of the source. So increase the $s \to c$ flow from 4 to 5.

2. Now we have an imbalance at c: an excess of one unit is incoming. We cannot increase the outflow from c any more, since the edge from c to e is operating at full capacity. So instead we *decrease* how much water is going from e to c by 1. This restore balance to c.

3. But now e is unbalanced: an excess of one unit is incoming. So we increase the $e \to t$ flow by 1, and this balances e.

4. Now one more unit of water is flowing into t (and one more unit flowing out of s). We have increased the net flow from t to s by one unit (from 12 to 13).

Here's the new flow:



This improvement we just made suggests the following definition:

Definition 13. Suppose we have a network with a feasible flow. An *augmenting path* is a sequence $E_1, E_2, \ldots, E_n$ of edges with the following properties:

1. If we ignore the directions of the edges, then $E_1, E_2, \ldots, E_n$ form a path from s to t.

2. As we traverse this path, all forward-pointing edges have a flow that is less than full capacity, and all backward pointing-edges have a flow that is greater than zero.

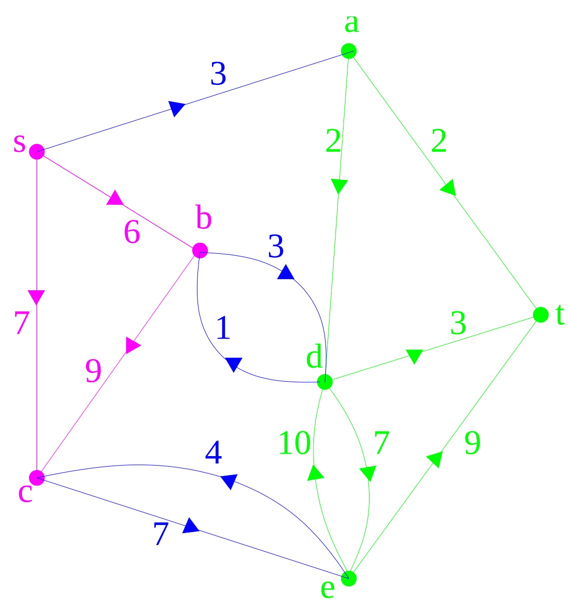
The "path" (ignoring edge directions) $\overrightarrow{sc}$, $\overleftarrow{ce}$, $\overrightarrow{et}$ that we modified above is an example of an augmenting path, since the forward-pointing edges ($\overrightarrow{sc}$ and $\overrightarrow{et}$) were at less than full capacity, and the backward-pointing edge ($\overleftarrow{ce}$) had nonzero capacity.

If we have an augmenting path, then increasing the total flow from s to t is straightforward. As we traverse the path, increase all flows on forward-pointing edges by 1 and decrease all flows on backward-pointing edges by 1. This is possible since the forward-pointing edges are not at full capacity, and the backward pointing edges have positive flow. This change increases the flow out of s by 1, increases the flow into t by 1, and we claim that it retains balance at all other vertices. To see this, look in our example above: since we increased the flow on $\overrightarrow{sc}$ by one and decreased the flow on $\overleftarrow{ce}$ by one, we have not changed the net flow into vertex c. Similarly, since we decreased the flow on $\overleftarrow{ce}$ by one and increased the net flow on $\overrightarrow{et}$ by one, we have not changed the net flow out of vertex e. There are other possible combinations of increase and decrease that we might see if we looked at other networks: generally speaking, we might increase the flow on both $\overrightarrow{uv}$ and $\overrightarrow{vw}$, or decrease the flow both on $\overleftarrow{uv}$ and $\overleftarrow{vw}$. Both of these preserve balance at v.

Let us return to our example, after we have increased the net flow from s to t to 13 by use of the augmenting path. Now we see that two of the pipes flowing into t are at maximum capacity and one is working at only one unit less than capacity. So perhaps we can improve the flow once again. This would clearly maximize the net flow from s to t in the system since all pipes into t would be at maximum capacity. We can try very hard to make incremental changes like before, but we seem to keep failing. But might there be a more clever way, perhaps involving a radical change in the way we use the network, that will get us to a net flow of 14?

6.3 Obstructions to Improvements: Cuts

The answer is NO. The reason is as follows: let us cut our graph into two components, one containing s, b, c, and the other containing a, d, e, t. The edges within the first and second components are pink and green, respectively, but the edges between components are in blue. We only notate the capacities of the edges, and do not put any assigned flow.



Now note that stuff moves from the blue component to the green component fastest if all pipes leading from blue to green are at maximum capacity and all pipes going the opposite way are unused. Thus the maximum possible flow from blue to green is the sum of the capacities of $\overrightarrow{sa}$, $\overrightarrow{bd}$, $\overrightarrow{ce}$, i.e., $3 + 3 + 7 = 13$. Since nothing can get from s to t without crossing from the blue component to the green component, we see that it would never be possible for the net flow from s to t to exceed 13. This PROVES that we cannot do better than a net flow of 13.

Now let us try to place the method we just used into a general context. We need to define what we did to the network above.

Definition 14. A *cut* of the network G is a partition of the vertices of G into two sets A and B, with s in set A, t in set B, and every other vertex is either in A or in B (but not both). We identify the cut with the ordered pair of sets (A, B). The *capacity* of (A, B) is the sum of the capacities of all the edges that start at vertices in A and end at vertices in B.

Thus, in the paragraph above, our cut was the partition of the vertices $\{s, t, a, b, c, d, e\}$ of G into two sets: $A = \{s, b, c\}$ and $B = \{a, d, e, t\}$. The capacity of the cut is found by adding the capacities of the edges that start in A and end in B, i.e., we add the capacities of edges $\overrightarrow{sa}$, $\overrightarrow{bd}$, $\overrightarrow{ce}$ to get $3 + 3 + 7 = 13$.

If we have a cut (A, B), then any flow from s to t must cross from A to B (since s is in A and t is in B). It is clear that you cannot move more from A to B than the capacity of (A, B), so you cannot move more from s to t than the capacity of (A, B). Thus we have the following principle:

The maximum net flow from s to t cannot be more than the minimum of the capacities of all cuts

That is, the lowest-capacity cut restricts the net flow from s to t to be less than or equal to its capacity. Actually, there may be multiple cuts that are equally low in capacity. These cuts which are tied for the lowest capacity are called *mincuts*.

So the net flow from s to t cannot exceed the capacity of a mincut. Can it get that high? Or could it be that even this is not achievable?

6.4 Maximum Flow exactly determined by Mincut

In fact, one can achieve the capacity of a mincut—a challenge problem on the homework asks you to prove that you can do this.

Theorem 3. *In a network G, the maximum achievable net flow from s to t is precisely the minimum of the capacities of all the cuts of G.*

Try to prove this in the homework!

From all that we have learned up to this point, we now have a general strategy for maximizing the net flow in a network:

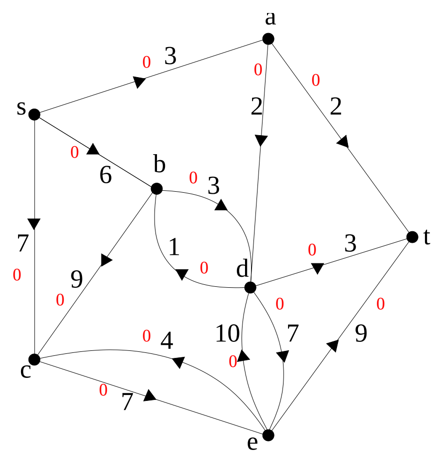
1. Start with a flow that assigns zero flow to all edges. This is certainly feasible and balanced.

2. Find an augmenting path P from s to t.

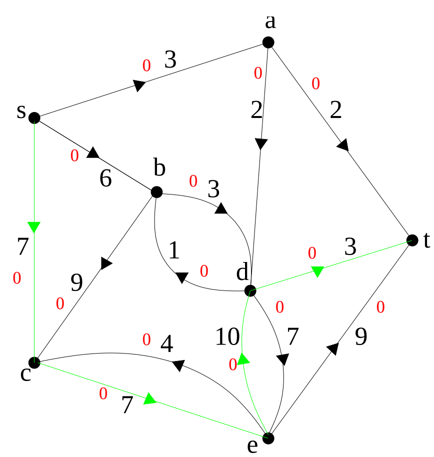
3. Increase by one the flow of all forward-pointing edges on P, and decrease by one the flow on all backward-pointing edges of P.

4. Repeat Step 3 until the path is no longer augmenting (i.e., some forward-pointing edge is working at full capacity or some backward-pointing edge has zero flow).
5. Repeat Steps 2 to 4 until you cannot find any more augmenting paths.
6. Compute the net flow from s to t.
7. To prove that net flow from s to t is maximized, find a cut (A, B) of your network such that the capacity of (A, B) is equal to the net flow you have from s to t.

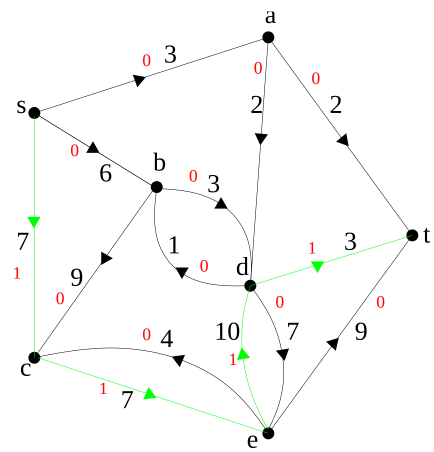
Let's try this procedure with our network above. We start (step 1) by assigning zero flow to all edges.



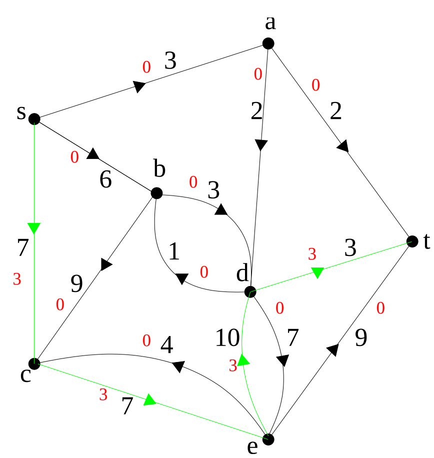
Then we find an augmenting path from s to t (step 2). We have a lot of freedom here. I feel like using high-capacity pipes, so starting at s, I follow $\overrightarrow{sc}$, then I am forced to use $\overrightarrow{ce}$, the I use $\overrightarrow{ed}$ because it is the biggest pipe out of e. It seems silly to go back to e again, so I use $\overrightarrow{dt}$. (Not that this was really a very intelligent overall choice of path, but bear with me.)



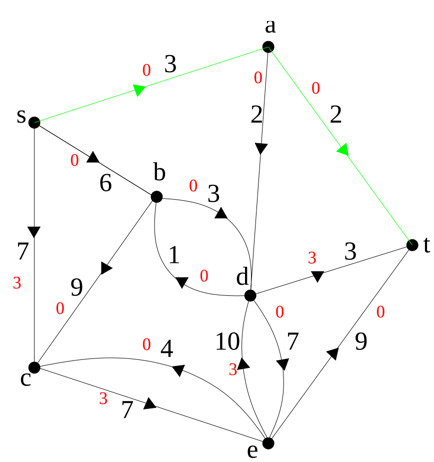
We increase the flow by one on each forward-pointing edge of the path (step 3). There are no backward-pointing edges.



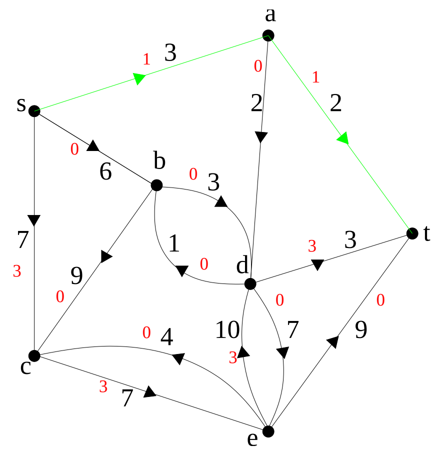
And then repeat this increasing two more times (step 4) until capacity limitations prevent further increase.



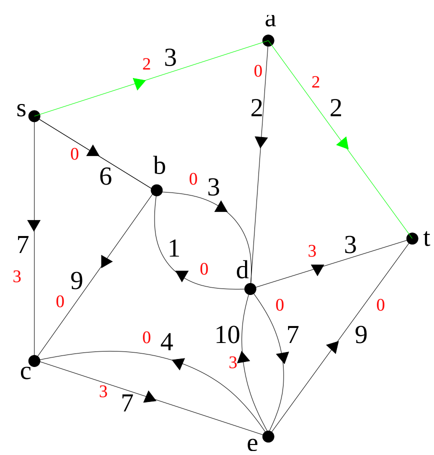
Now step 5 tells us to go back to step 2 and find another augmenting path: now I decide to go with $\overrightarrow{sa}$ and then $\overrightarrow{at}$.



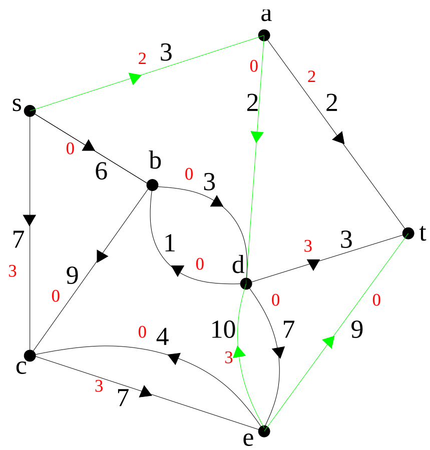
Both edges are forward-pointing; step 3 tells me to increase the flow on them by 1.



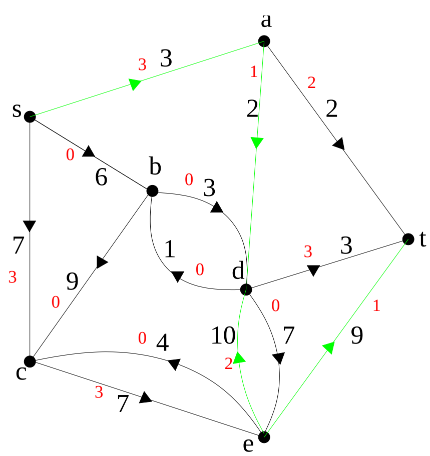
Step 4 tells us to repeat this procedure once more, at which point $\overrightarrow{at}$ cannot have its flow increased any more, so that we no longer have an augmenting path.



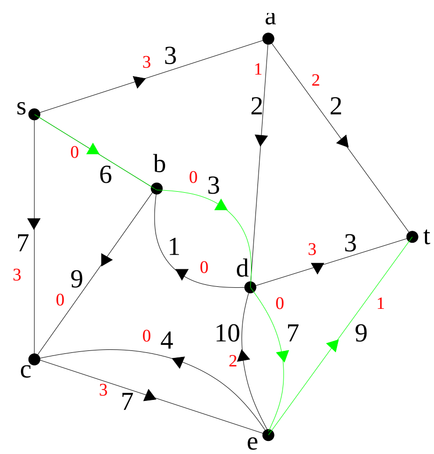
Step 5 tells us to go back to step 2 and look for an augmenting path. I think I can make further use of $\overrightarrow{sa}$ by proceeding to $\overrightarrow{ad}$. I cannot use $\overrightarrow{dt}$, since it is working at full capacity. But I can use $\overleftarrow{de}$ (which points toward me instead of away from me, but it is OK since it has nonzero flow, so it can be part of an augmenting path). I complete the path by using $\overrightarrow{et}$.



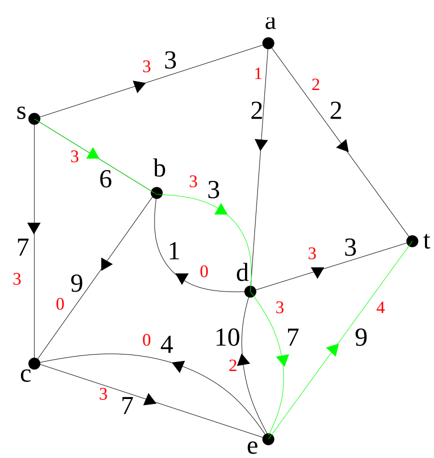
Now step 3 tells me to increase flows on the forward-pointing edges ($\overrightarrow{sa}$, $\overrightarrow{ad}$, and $\overrightarrow{et}$) by 1, while *decreasing* the flow on the backward-pointing edge $\overleftarrow{de}$ by 1:



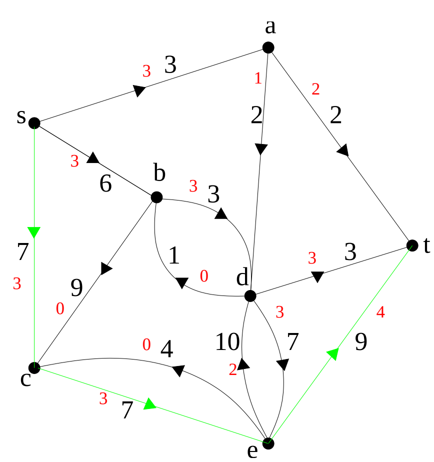
I can do no more with this path, since $\overrightarrow{sa}$ is at full capacity. So we look for yet another augmenting path, and find one consisting of $\overrightarrow{sb}$, $\overrightarrow{bd}$, $\overrightarrow{de}$, and $\overrightarrow{et}$.



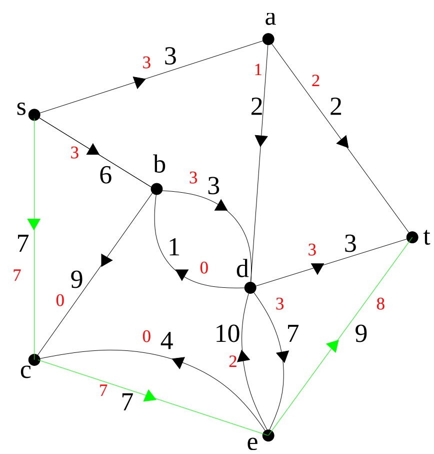
Steps 3 and 4 tell us to increase the flow on these edges until one of them ($\overrightarrow{bd}$) comes to full capacity:



Then we look for another augmenting path: this time we try $\overrightarrow{sc}$, $\overrightarrow{ce}$, $\overrightarrow{et}$.

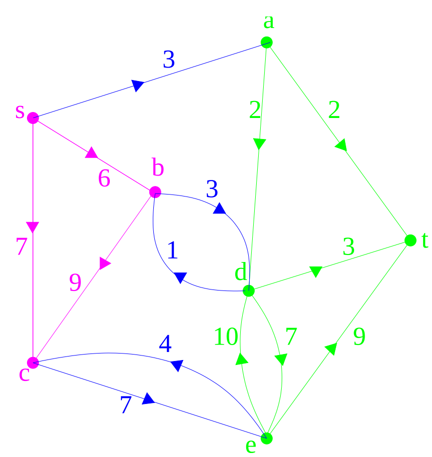


Steps 3 and 4 tell us to increase the flow on these edges until one of them ($\overrightarrow{sc}$, or, equally well, $\overrightarrow{ce}$) comes to full capacity:



If we try to find another augmenting path, we need to use $\overrightarrow{sb}$ (since all other pipes out of s are at full capacity). Then the only way to proceed is to use $\overrightarrow{bc}$ (since $\overrightarrow{bd}$ is at full capacity and $\overleftarrow{bd}$ has zero flow). Then the only way to proceed from there is to use $\overleftarrow{cs}$ (since $\overrightarrow{ce}$ is at full capacity and $\overleftarrow{ce}$ has zero flow). But now we are back at s again, so there is no way to get to t. This makes us think that we have maximized the net flow.

We compute the net flow (step 6) to be $3 + 7 + 7 = 13$. All we need to do is find a cut that has capacity 13 (step 7) and we will have proved that the net flow cannot be increased any further. But we know of such a cut: recall that the cut $(\{s, b, c\}, \{a, d, e, t\})$



has capacity 13.