

Graph Theory and Network Flows

In the modern world, planning efficient routes is essential for business and industry, with applications as varied as product distribution, laying new fiber optic lines for broadband internet, and suggesting new friends within social network websites like Facebook.

This field of mathematics started nearly 300 years ago as a look into a mathematical puzzle (we'll look at it in a bit). The field has exploded in importance in the last century, both because of the growing complexity of business in a global economy and because of the computational power that computers have provided us.

Graphs

Drawing Graphs

Here is a portion of a housing development from Missoula, Montana¹. As part of her job, the development's lawn inspector has to walk down every street in the development making sure homeowners' landscaping conforms to the community requirements.

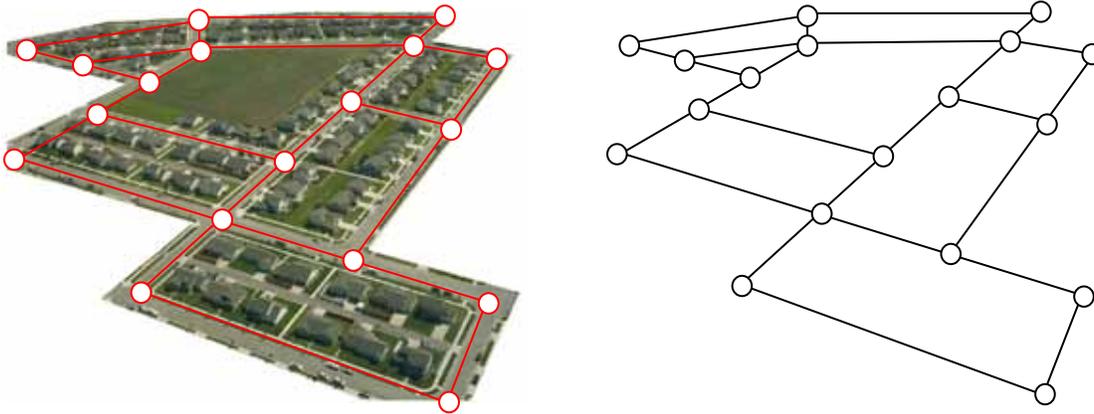


Naturally, she wants to minimize the amount of walking she has to do. Is it possible for her to walk down every street in this development without having to do any backtracking?

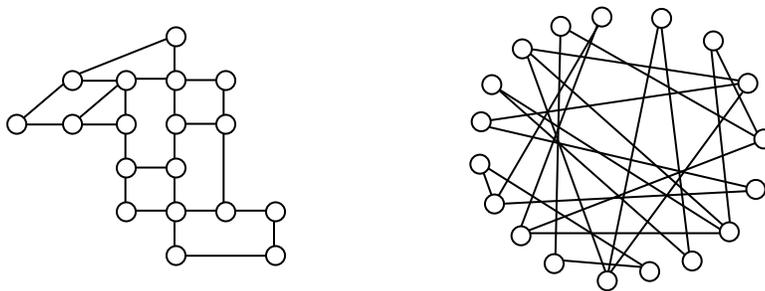
¹ Sam Beebe. <http://www.flickr.com/photos/sbeebe/2850476641/>

While you might be able to answer that question just by looking at the picture for a while, it would be ideal to be able to answer the question for any picture regardless of its complexity.

To do that, we first need to simplify the picture into a form that is easier to work with. We can do that by drawing a simple line for each street. Where streets intersect, we will place a dot.



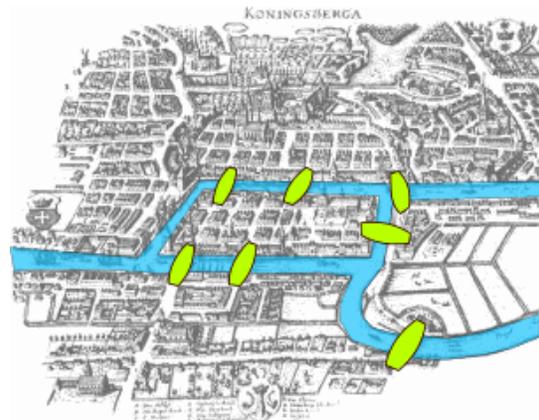
This type of simplified picture is called a **graph**. A graph consists of a set of dots, called **vertices**, and a set of **edges** connecting pairs of vertices. While we drew our original graph to correspond with the picture we had, there is nothing particularly important about the layout when we analyze a graph. Both of these graphs are equivalent to the one drawn above.



Example: Back in the 18th century in the Prussian city of Königsberg, a river ran through the city and seven bridges crossed the forks of the river. The river and the bridges are highlighted in the picture to the right².

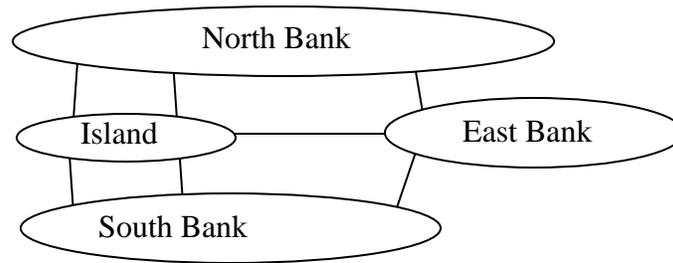
As a weekend amusement, townsfolk would see if they could find a route that would take them across every bridge once and return them to where they started.

Leonard Euler (pronounced OY-lur), one of the most prolific mathematicians ever, looked at this problem in 1735, laying the foundations for graph theory as a field in mathematics.

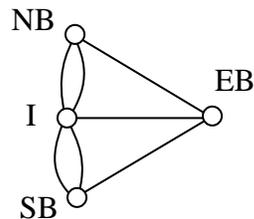


² Bogdan Giușcă. http://en.wikipedia.org/wiki/File:Königsberg_bridges.png

To analyze this problem, Euler introduced edges representing the bridges:



Since it is not relevant to the question of bridge crossing how large each land mass is, they can be shrunk down to single vertices representing each location:



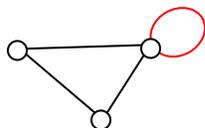
Notice that in this graph there are *two* edges connecting the north bank and island, corresponding to the two bridges in the original drawing. Depending upon the interpretation of edges and vertices appropriate to a scenario, it is entirely possible and reasonable to have more than one edge connecting two vertices.

Definitions

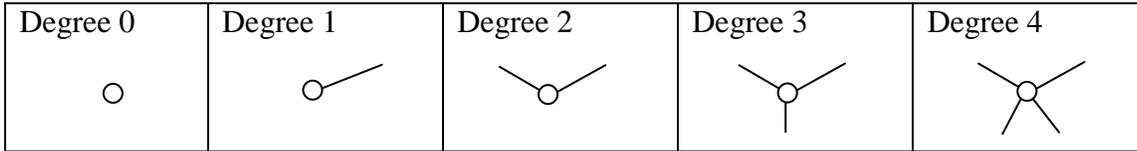
Vertex. A vertex is a dot in the graph where edges meet. A vertex could represent an intersection of streets, a land mass, or a general location, like “work” or “school”. Note that vertices only occur when a dot is explicitly placed, not whenever two edges cross. Imagine a freeway overpass – the freeway and side street cross, but it is not possible to change from the side street to the freeway at that point, so there is no intersection and no vertex would be placed.

Edges. Edges connect pairs of vertices. An edge can represent a physical connection between locations, like a street, or simply that a route connecting the two locations exists, like an airline flight.

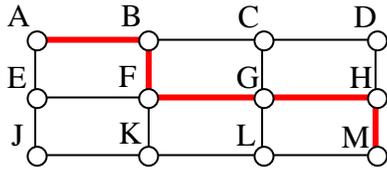
Loop. A loop is a special type of edge that connects a vertex to itself. Loops are not used much in street network graphs.



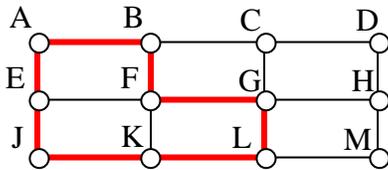
Degree of a vertex. The degree of a vertex is the number of edges meeting at that vertex. It is possible for a vertex to have a degree of zero or larger.



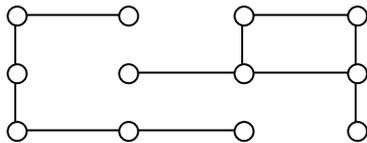
Path. A path is a sequence of vertices using the edges. Usually we are interested in a path between two vertices. For example, a path from vertex A to vertex M is shown below. It is one of many possible paths in this graph.



Circuit. A circuit is a path that begins and ends at the same vertex. A circuit starting and ending at vertex A is shown below.



Connected. A graph is connected if there is a path from any vertex to any other vertex. Every graph drawn so far has been connected. The graph below is **disconnected**; there is no way to get from the vertices on the left to the vertices on the right.



Weights. Depending upon the problem being solved, sometimes weights are assigned to the edges. The weights could represent the distance between two locations, the travel time, or the travel cost. It is important to note that the distance between vertices in a graph does not necessarily correspond to the weight of an edge.

Shortest Path

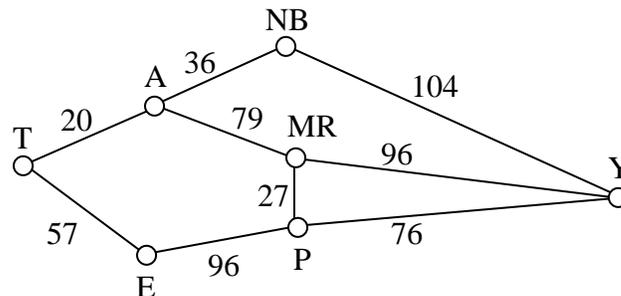
When you visit a website like Google Maps or MapQuest and ask for directions from home to your Aunt's house in Pasadena, you are usually looking for a shortest path between the two locations. These computer applications use representations of the street maps as graphs, with estimated driving times as edge weights.

While often it is possible to find a shortest path on a small graph by guess-and-check, our goal in this chapter is to develop methods to solve complex problems in a systematic way by following **algorithms**. An algorithm is a step-by-step procedure for solving a problem. Dijkstra's (pronounced dike-strā) algorithm will find the shortest path between two vertices.

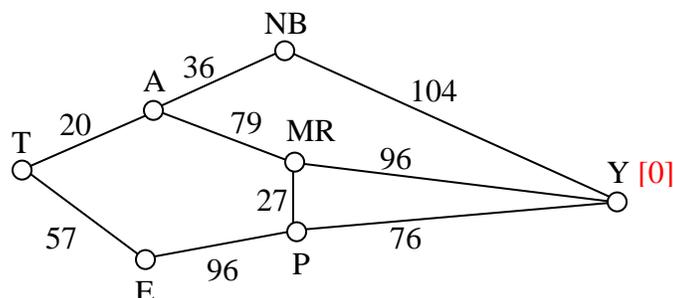
Dijkstra's Algorithm

- 1) Mark the ending vertex with a distance of zero. Designate this vertex as current.
- 2) Find all vertices leading to the current vertex. Calculate their distances to the end. Since we already know the distance the current vertex is from the end, this will just require adding the most recent edge. Don't record this distance if it is longer than a previously recorded distance.
- 3) Mark the current vertex as visited. We will never look at this vertex again.
- 4) Mark the vertex with the smallest distance as current, and repeat from step 2.

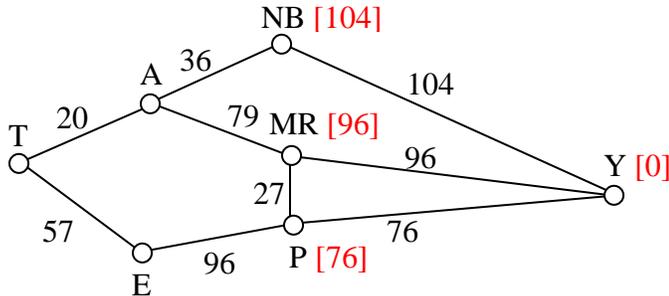
Example: Suppose you need to travel from Tacoma, WA (vertex T) to Yakima, WA (vertex Y). Looking at a map, it looks like driving through Auburn (A) then Mount Rainier (MR) might be shortest, but it's not totally clear since that road is probably slower than taking the major highway through North Bend (NB). A graph with travel times in minutes is shown below. An alternate route through Eatonville (E) and Packwood (P) is also shown.



Step 1: Mark the ending vertex with a distance of zero. The distances will be recorded in **[brackets]** after the vertex name.



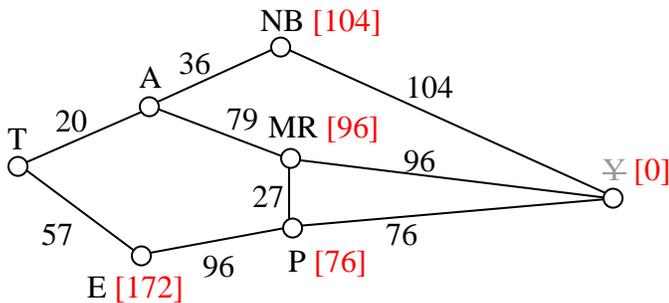
Step 2: For each vertex leading to Y, we calculate the distance to the end. For example, NB is a distance of 104 from the end, and MR is 96 from the end. Remember that distances in this case refer to the travel time in minutes.



Step 3 & 4: We mark Y as visited, and mark the vertex with the smallest recorded distance as current. At this point, P will be designated current. Back to step 2.

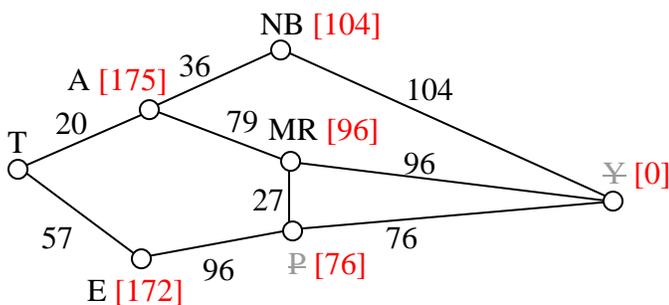
Step 2 (#2): For each vertex leading to P (and not leading to a visited vertex) we find the distance from the end. Since E is 96 minutes from P, and we've already calculated P is 76 minutes from Y, we can compute that E is $96+76 = 172$ minutes from Y.

If we make the same computation for MR, we'd calculate $76+27 = 103$. Since this is larger than the previously recorded distance from Y to MR, we will *not* replace it.



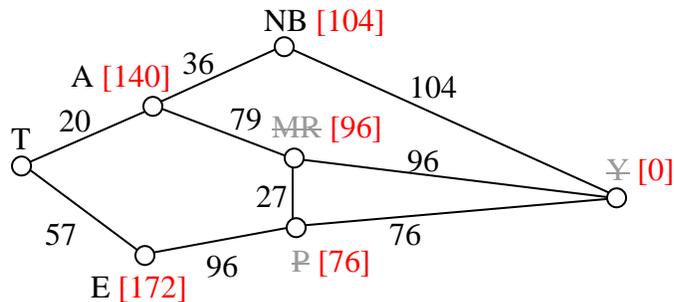
Step 3 & 4 (#2): We mark P as visited, and designate the vertex with the smallest recorded distance as current: MR. Back to step 2.

Step 2 (#3): For each vertex leading to MR (and not leading to a visited vertex) we find the distance to the end. The only vertex to be considered is A, since we've already visited Y and P. Adding MR's distance 96 to the length from A to MR gives the distance $96+79 = 175$ minutes from A to Y.



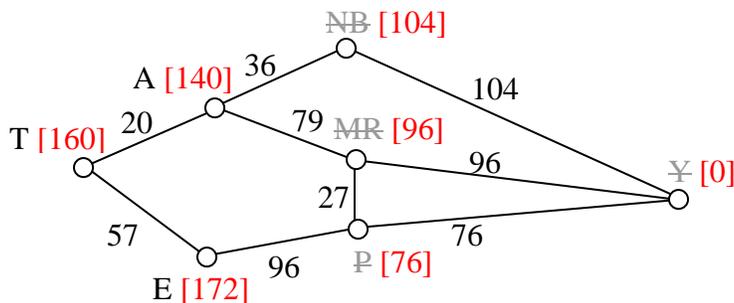
Step 3 & 4 (#3): We mark MR as visited, and designate the vertex with smallest recorded distance as current: NB. Back to step 2.

Step 2 (#4): For each vertex leading to NB, we find the distance to the end. We know the shortest distance from NB to Y is 104 and the distance from A to NB is 36, so the distance from A to Y through NB is $104+36 = 140$. Since this distance *is* shorter than the previously calculated distance from Y to A through MR, we replace it.



Step 3 & 4 (#4): We mark NB as visited, and designate A as current, since it now has the shortest distance.

Step 2 (#5): T is the only non-visited vertex leading to A, so we calculate the distance from T to Y through A: $20+140 = 160$ minutes.



Step 3 & 4 (#5): We mark A as visited, and designate E as current.

Step 2 (#6): The only non-visited vertex leading to E is T. Calculating the distance from T to Y through E, we compute $172+57 = 229$ minutes. Since this is longer than the existing marked time, we do not replace it.

Step 3 (#6): We mark E as visited. Since all vertices have been visited, we are done.

From this, we know that the shortest path from Tacoma to Yakima will take 160 minutes. Tracking which sequence of edges yielded 160 minutes, we see the shortest path is T-A-NB-Y.

Dijkstra's algorithm is an **optimal algorithm**, meaning that it always produces the actual shortest path, not just a path that is pretty short, provided one exists. This algorithm is also **efficient**, meaning that it can be implemented in a reasonable amount of time. Dijkstra's algorithm takes around V^2 calculations, where V is the number of vertices in a graph³. A graph with 100 vertices would take around 10,000 calculations. While that would be a lot to do by hand, it is not a lot for computer to handle. It is because of this efficiency that your car's GPS unit can compute driving directions in only a few seconds.

In contrast, an inefficient algorithm might try to list all possible paths then compute the length of each path. An inefficient algorithm could easily take 10^{25} calculations to compute the shortest path with only 25 vertices; that's a 1 with 25 zeros after it! To put that in perspective, the fastest computer in the world would still spend over 1000 years analyzing all those paths.

Example: A shipping company needs to route a package from Washington, DC to San Diego, CA. To minimize costs, the package will first be sent to their processing center in Baltimore, MD then sent as part of mass shipments between their various processing centers, ending up in their processing center in Bakersfield, CA. From there it will be delivered in a small truck to San Diego.

The travel times, in hours, between their processing centers are shown in the table below. Three hours has been added to each travel time for processing. Find the shortest path from Baltimore to Bakersfield.

	Baltimore	Denver	Dallas	Chicago	Atlanta	Bakersfield
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	*	
Bakersfield		19	25			*

While we could draw a graph, we can also work directly from the table.

Step 1: The ending vertex, Bakersfield, is marked as current.

Step 2: All cities connected to Bakersfield have their distances calculated which are Denver and Dallas; we'll mark those distances in the column headers.

Step 3 & 4: Mark Bakersfield as visited, and the Denver as current, since it is the vertex with the shortest distance.

³ It can be made to run faster through various optimizations to the implementation.

	Baltimore	Denver [19]	Dallas [25]	Chicago	Atlanta	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	*	
Bakersfield		19	25			*

Step 2 (#2): For cities connected to Denver, calculate distance to the end. For example, Chicago is 18 hours from Denver, and Denver is 19 hours from the end, the distance for Chicago to the end is $18+19 = 37$ (Chicago to Denver to Bakersfield)

Step 3 & 4 (#2): We mark Denver as visited and mark Dallas as current.

	Baltimore	Denver [19]	Dallas [25]	Chicago [37]	Atlanta	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	*	
Bakersfield		19	25			*

Step 2 (#3): For cities connected to Dallas, calculate the distance to the end. For Chicago, the distance from Chicago to Dallas is 18 and from Dallas to the end is 25, so the distance from Chicago to the end through Dallas would be $18+25 = 43$. Since this is longer than the currently marked distance for Chicago, we do not replace it. For Atlanta, we calculate $15+25 = 40$, and mark that distance.

Step 3 & 4 (#3): We mark Dallas as visited, and mark Chicago as current.

	Baltimore	Denver [19]	Dallas [25]	Chicago [37]	Atlanta [40]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	*	
Bakersfield		19	25			*

Step 2 (#4): Baltimore and Atlanta are the only non-visited cities connected to Chicago. For Baltimore, we calculate $15+37 = 52$ and mark that distance. For Atlanta, we calculate $14+37 = 51$. Since this is longer than the existing distance of 40 for Atlanta, we do not replace that distance.

Step 3 & 4 (#4): Mark Chicago as visited and Atlanta as current.

	Baltimore [52]	Denver [19]	Dallas [25]	Chicago [37]	Atlanta [40]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	*	
Bakersfield		19	25			*

Step 2 (#5): The distance from Atlanta to Baltimore is 14. Adding that to the distance already calculated for Atlanta gives a total distance of $14+40 = 54$ hours from Baltimore to Bakersfield through Atlanta. Since this is larger than the currently calculated distance, we do not replace the distance for Baltimore.

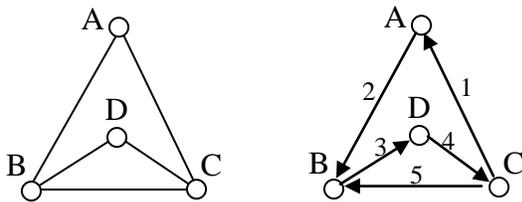
Step 3 & 4 (#5): We mark Atlanta as visited. All cities have been visited and we are done. The shortest route from Baltimore to Bakersfield will take 52 hours, and will route through Chicago and Denver.

Euler Circuits and the Chinese Postman Problem

In the first section, we created a graph of the Königsberg bridges and asked whether it was possible to walk across every bridge once. Because Euler first studied this question, they are named after him.

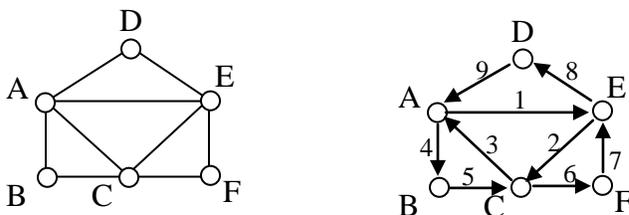
An **Euler path** is a path that uses every edge in a graph with no repeats. Being a path, it does not have to return to the starting vertex.

Example: In the graph shown below, there are several Euler paths. One such path is CABDCB. The path is shown in arrows to the right, with the order of edges numbered.



An **Euler circuit** is a circuit that uses every edge in a graph with no repeats. Being a circuit, it must start and end at the same vertex.

Example: The graph below has several possible Euler circuits. Here's a couple, starting and ending at vertex A: ADEACEFCBA and AECABCFEDA. The second is shown in arrows.



Look back at the example used for Euler paths – does that graph have an Euler circuit? A few tries will tell you no; that graph does not have an Euler circuit. When we were working with shortest paths, we were interested in the optimal path. With Euler paths and circuits, we're primarily interested in whether an Euler path or circuit *exists*.

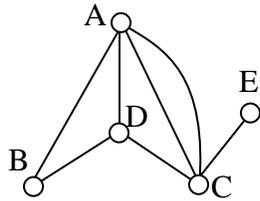
Why do we care if an Euler circuit exists? Think back to our housing development lawn inspector from the beginning of the chapter. The lawn inspector is interested in walking as little as possible. The ideal situation would be a circuit that covers every street with no repeats. That's an Euler circuit! Luckily, Euler solved the question of whether or not an Euler path or circuit will exist.

Euler's Path and Circuit Theorems

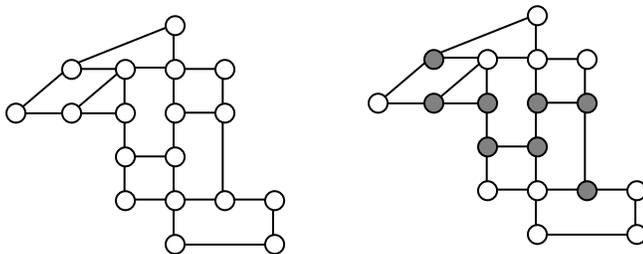
A graph will contain an Euler path if it contains at most two vertices of odd degree.

A graph will contain an Euler circuit if all vertices have even degree

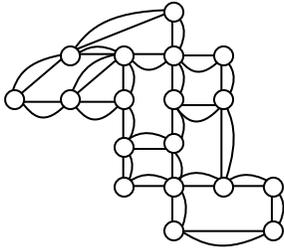
Example: In the graph below, vertices A and C have degree 4, since there are 4 edges leading into each vertex. B is degree 2, D is degree 3, and E is degree 1. This graph contains two vertices with odd degree (D and E) and three vertices with even degree (A, B, and C), so Euler's theorems tell us this graph has an Euler path, but not an Euler circuit.



Example: Is there an Euler circuit on the housing development lawn inspector graph we created earlier in the chapter? All the highlighted vertices have odd degree. Since there are more than two vertices with odd degree, there are no Euler paths or Euler circuits on this graph. Unfortunately our lawn inspector will need to do some backtracking.



Example: When it snows in the same housing development, the snowplow has to plow both sides of every street. For simplicity, we'll assume the plow is out early enough that it can ignore traffic laws and drive down either side of the street in either direction. This can be visualized in the graph by drawing two edges for each street, representing the two sides of the street.

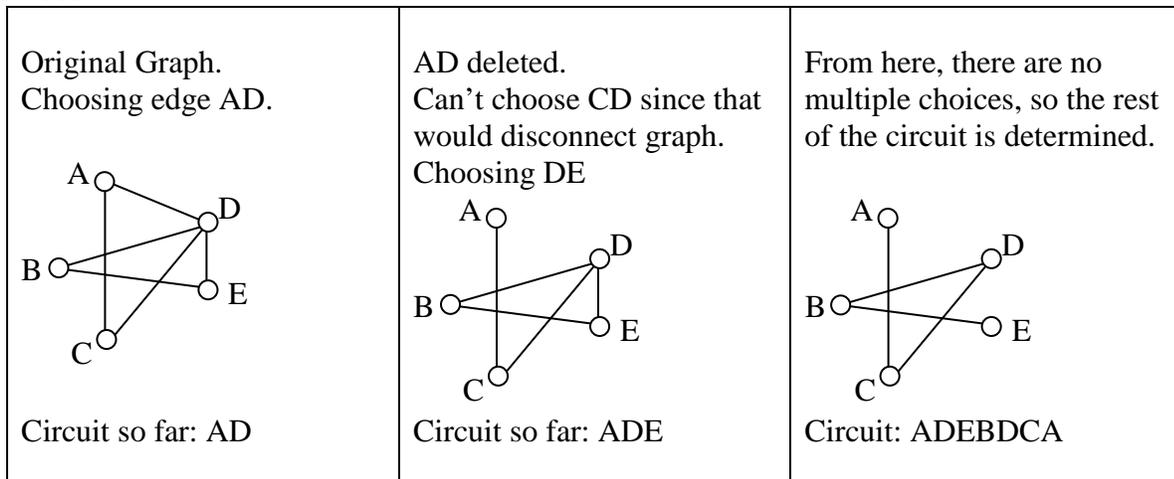


Notice that every vertex in this graph has even degree, so this graph does have an Euler circuit. Now how do we find one? While it usually is possible to find an Euler circuit just by pulling out your pencil and trying to find one, the more formal method is **Fleury's algorithm**.

Fleury's Algorithm

- 1) Start at any vertex if finding an Euler circuit. If finding an Euler path, start at one of the two vertices with odd degree.
- 2) Choose any edge leaving your current vertex, provided deleting that edge will not separate the graph into two disconnected sets of edges.
- 3) Add that edge to your circuit, and delete it from the graph.
- 4) Continue until you're done.

Example: Let's find an Euler Circuit on this graph using Fleury's algorithm, starting at vertex A.



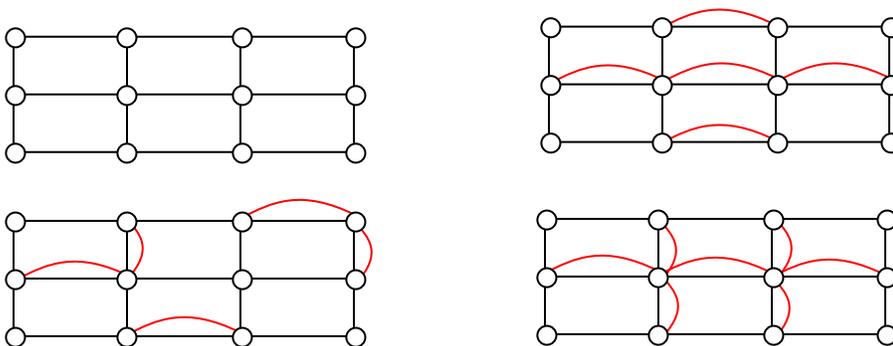
Eulerization and the Chinese Postman Problem

Not every graph has an Euler path or circuit, yet our lawn inspector still needs to do her inspections. Her goal is to minimize the amount of walking she has to do. In order to do that, she will have to duplicate some edges in the graph until an Euler circuit exists.

Eulerization is the process of adding edges to a graph to create an Euler circuit on a graph. To eulerize a graph, edges are duplicated to connect pairs of vertices with odd degree. Connecting two odd degree vertices increases the degree of each, giving them both even degree. When two odd degree vertices are not directly connected, we can duplicate all edges in a path connecting the two.

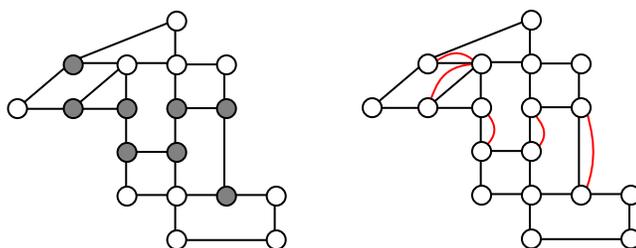
Note that we can only duplicate edges, not create edges where there wasn't one before; that's the difference between driving down a road twice and installing a new road!

Example: For the rectangular graph shown, three possible eulerizations are shown. Notice in each of these cases the vertices that started with odd degrees have even degrees after eulerization, allowing for an Euler circuit.



In the example above, you'll notice that the last eulerization required duplicating seven edges, while the first two only required duplicating five edges. If we were eulerizing the graph to find a walking path, we would want the eulerization with minimal duplications. If the edges had weights representing distances or costs, then we would want to select the eulerization with the minimal total added weight.

Example: For our lawn inspector route, the vertices with odd degree are shown highlighted. We can eulerize this graph by duplicating only four edges – the least possible with eight vertices to address. Without weights we can't be certain this is the eulerization that minimizes walking distance, but it looks pretty good.



The problem of finding the optimal eulerization is called the Chinese Postman Problem, a name given by an American in honor of the Chinese mathematician Mei-Ko Kwan who first studied the problem in 1962 while trying to find optimal delivery routes for postal carriers. This is important in determining efficient routes for garbage trucks, school buses, parking meter checkers, street sweepers, and more.

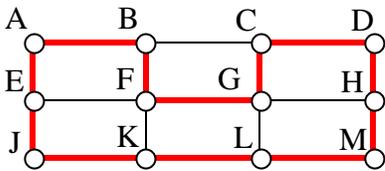
Unfortunately, algorithms to solve this problem are fairly complex. Some simpler cases are considered in the exercises.

Hamiltonian Circuits and the Traveling Salesman Problem

In the last section, we considered optimizing a walking route for a postal carrier. How is this different than the requirements of a package delivery driver? While the postal carrier needed to walk down every street (edge) to deliver the mail, the package delivery driver instead needs to visit every one of a set of delivery locations. Instead of looking for a circuit that covers every edge once, the package deliverer is interested in a circuit that visits every vertex once.

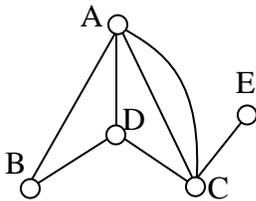
A **Hamiltonian circuit** is a circuit that visits every vertex once with no repeats. Being a circuit, it must start and end at the same vertex. A **Hamiltonian path** also visits every vertex once with no repeats, but does not have to start and end at the same vertex. Hamiltonian circuits are named for William Rowan Hamilton who studied them in the 1800's.

Example: One Hamiltonian circuit is shown on the graph below. There are several other Hamiltonian circuits possible on this graph. Notice that the circuit only has to visit every vertex once; it does not need to use every edge. This circuit could be notated by the sequence of vertices visited, starting and ending at the same vertex: ABFGCDHMLKJEA. Notice that the same circuit could be written in reverse order, or starting and ending at a different vertex.



Unlike with Euler circuits, there is no nice theorem that allows us to instantly determine whether or not a Hamiltonian circuit exists for all graphs.⁴

Example: Does a Hamiltonian path or circuit exist on the graph below?

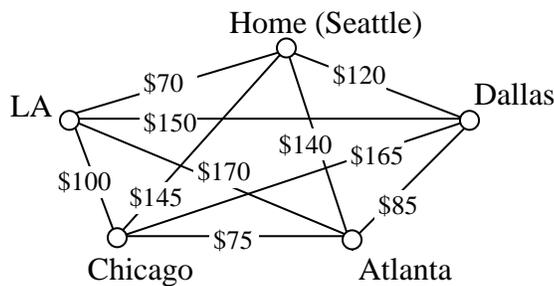


⁴ There are some theorems that can be used in specific circumstances, such as Dirac's theorem, which says that a Hamiltonian circuit must exist on a graph with n vertices if each vertex has degree $n/2$ or greater.

We can see that once we travel to vertex E there is no way to leave without returning to C, so there is no possibility of a Hamiltonian circuit. If we start at vertex E we can find several Hamiltonian paths, such as ECDAB and ECABD.

With Hamiltonian circuits, our focus will not be on existence, but on the question of optimization; given a graph where the edges have weights, can we find the optimal Hamiltonian circuit; the one with lowest total weight.

This problem is called the **Traveling salesman problem (TSP)** because the question can be framed like this: Suppose a salesman needs to give sales pitches in four cities. He looks up the airfares between each city⁵, and puts the costs in a graph. In what order should he travel to visit each city once then return home with the lowest cost?

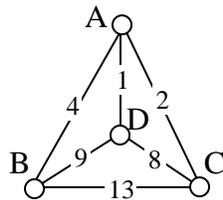


To answer this question, we will consider some possible approaches.

Brute Force Algorithm (a.k.a. exhaustive search)

- 1) List all possible Hamiltonian circuits
- 2) Find the length of each circuit by adding the edge weights
- 3) Select the circuit with minimal total weight.

Example: Apply the Brute force algorithm to find the minimum cost Hamiltonian circuit on the graph below.



To apply the Brute force algorithm, we list all possible Hamiltonian circuits and calculate their weight:

Circuit	Weight
ABCD A	$4+13+8+1 = 26$
ABDCA	$4+9+8+2 = 23$
ACBDA	$2+13+9+1 = 25$

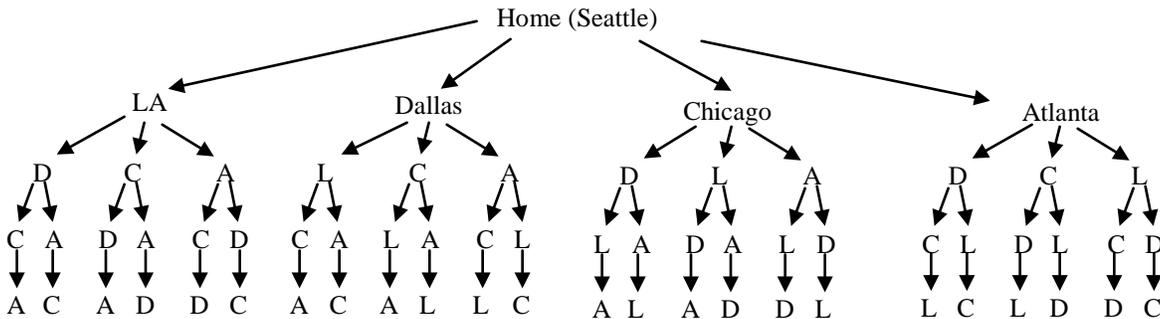
⁵ Travelocity.com. Retrieved 5/14/2009 for travel on 8/12/2009.

Note: These are the unique circuits on this graph. All other possible circuits are the reverse of the listed ones or start at a different vertex, but result in the same weights.

From this we can see that the second circuit, ABDCA, is the optimal circuit.

The Brute force algorithm is optimal; it will always produce the Hamiltonian circuit with minimum weight. Is it efficient? To answer that question, we need to consider how many Hamiltonian circuits a graph could have. For simplicity, let's look at the worst-case possibility, where every vertex is connected to every other vertex. This is called a **complete graph**.

Suppose we had a complete graph with five vertices like the air travel graph above. From Seattle there are four cities we can visit first. From each of those, there are three choices. From each of those cities, there are two possible cities to visit next. There is then only one choice for the last city before returning home. This can be shown visually:



Counting the number of routes, we can see there are $4 \times 3 \times 2 \times 1 = 24$ routes. For six cities there would be $5 \times 4 \times 3 \times 2 \times 1 = 120$ routes. For N cities there would be $(n - 1)! = (n - 1)(n - 2)(n - 3) \dots 3 \times 2 \times 1$ routes. The exclamation symbol is read “factorial” and is shorthand for the product shown.

Example: A complete graph with 8 vertices would have $(8 - 1)! = 7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$ possible Hamiltonian circuits. Half of the circuits are duplicates of other circuits but in reverse order, leaving 2520 unique routes. While this is a lot, it doesn't seem unreasonably huge. But consider what happens as the number of cities increase:

Cities	Unique Hamiltonian Circuits
9	$8!/2 = 20,160$
10	$9!/2 = 181,440$
11	$10!/2 = 1,814,400$
15	$14!/2 = 43,589,145,600$
20	$19!/2 = 60,822,550,204,416,000$

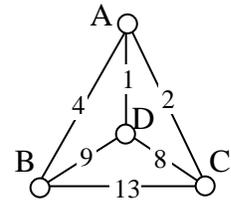
As you can see the number of circuits is growing extremely quickly. If a computer looked at one billion circuits a second, it would still take almost two years to examine all the possible circuits with only 20 cities! Certainly Brute Force is **not** an efficient algorithm.

Unfortunately, no one has yet found an efficient *and* optimal algorithm to solve the TSP. Since it is not practical to use brute force to solve the problem, we turn instead to heuristic algorithms; efficient algorithms that give approximate solutions. In other words, heuristic algorithms are fast, but may or may not produce the optimal circuit.

Nearest Neighbor Algorithm (NNA)

- 1) Select a starting point.
- 2) Move to the nearest unvisited vertex (the edge with smallest weight).
- 3) Repeat until the circuit is complete.

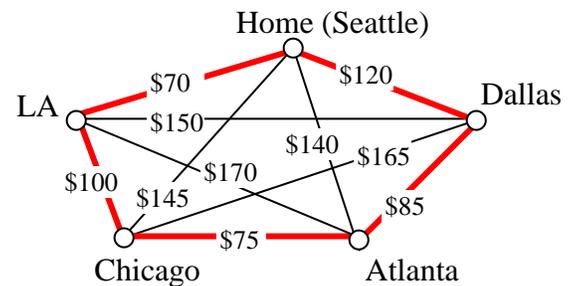
Example: Consider our earlier graph, shown to the right. Starting at vertex A, the nearest neighbor is vertex D with a weight of 1. From D, the nearest neighbor is C, with a weight of 8. From C, our only option is to move to vertex B, the only unvisited vertex, with a cost of 13. From B we return to A with a weight of 4. The resulting circuit is ADCBA with a total weight of $1+8+13+4 = 26$.



We ended up finding the worst circuit in the graph! What happened? Unfortunately, while it is very easy to implement, the NNA is a **greedy** algorithm, meaning it only looks at the immediate decision without considering the consequences in the future. In this case, following the edge AD forced us to use the very expensive edge BC later.

Example: Consider again our salesman. Starting in Seattle, the nearest neighbor (cheapest flight) is to LA, at a cost of \$70. From there:

LA to Chicago: \$100
 Chicago to Atlanta: \$75
 Atlanta to Dallas: \$85
 Dallas to Seattle: \$120
 Total cost: \$450



In this case, nearest neighbor did find the optimal circuit.

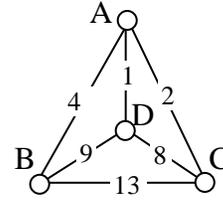
Going back to our first example, how could we improve the outcome? One option would be to redo the nearest neighbor algorithm with a different starting point to see if the result changed. Since nearest neighbor is so fast, doing it several times isn't a big deal.

Repeated Nearest Neighbor Algorithm (RNNA)

- 1) Do the Nearest Neighbor Algorithm starting at each vertex
- 2) Choose the circuit produced with minimal total weight

Example: Starting at vertex A resulted in a circuit with weight 26.

Starting at vertex B, the nearest neighbor circuit is BADCB with a weight of $4+1+8+13 = 26$. This is the same circuit we found starting at vertex A. No better.



Starting at vertex C, the nearest neighbor circuit is CADBC with a weight of $2+1+9+13 = 25$. Better!

Starting at vertex D, the nearest neighbor circuit is DACBA, the same circuit we found starting at C.

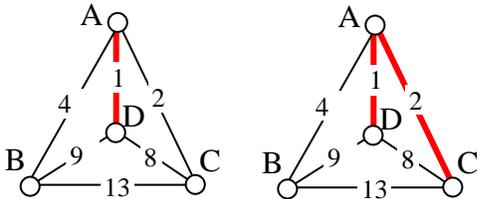
So the RNNA was able to produce a slightly better circuit with a weight of 25, but still not the optimal circuit. Notice that even though we found the circuit by starting at vertex C, we could still write the circuit starting at A: ADBCA or ACBDA.

Unfortunately, the RNNA is still greedy and will produce very bad results for some graphs. As an alternative, our next approach will step back and look at the “big picture” – it will select first the edges that are shortest, and then fill in the gaps.

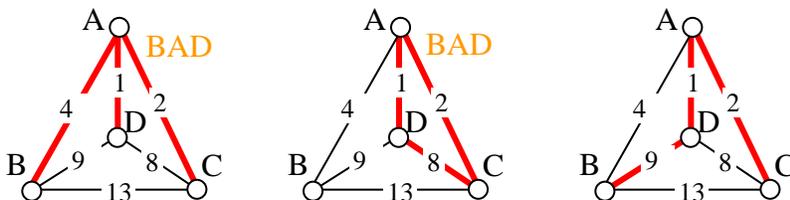
Sorted Edges Algorithm (a.k.a. Cheapest Link Algorithm)

- 1) Select the cheapest unused edge in the graph; highlight it.
- 2) Repeat step 1, adding the cheapest unused edge to the graph, unless:
 - a. adding the edge would create a circuit that doesn't contain all vertices, or
 - b. adding the edge would give a vertex degree 3.
- 3) Repeat until a circuit containing all vertices is formed.

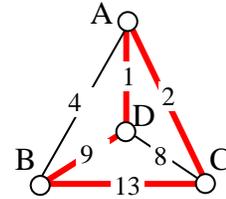
Example: Using the four vertex graph from above, the cheapest edge is AD, with a cost of 1. We highlight that edge. The next shortest edge is AC, with a weight of 2, so we highlight that edge.



For the third edge, we'd like to add AB, but that would give vertex A degree 3, which shouldn't happen in a Hamiltonian circuit. The next shortest edge is CD, but that edge would create a circuit ACDA that does not include vertex B, so we reject that edge. The next shortest edge is BD, so we add that edge to the graph.



We then add the last edge to complete the circuit: ACBDA with weight 25. Notice that the algorithm did not produce the optimal circuit in this case (circuit ACDBA with weight 23). While the Sorted Edge algorithm overcomes some of the shortcomings of NNA, it is still only a heuristic algorithm, and does not guarantee the optimal circuit.



Example: Your teacher's band, *Derivative Work*, is doing a bar tour in Oregon. The driving distances are shown below. Plan an efficient route for your teacher to visit all the cities and return to the starting location. Use NNA starting at Portland, and Sorted Edges.

	Ashland	Astoria	Bend	Corvallis	Crater Lake	Eugene	Newport	Portland	Salem	Seaside
Ashland	-	374	200	223	108	178	252	285	240	356
Astoria	374	-	255	166	433	199	135	95	136	17
Bend	200	255	-	128	277	128	180	160	131	247
Corvallis	223	166	128	-	430	47	52	84	40	155
Crater Lake	108	433	277	430	-	453	478	344	389	423
Eugene	178	199	128	47	453	-	91	110	64	181
Newport	252	135	180	52	478	91	-	114	83	117
Portland	285	95	160	84	344	110	114	-	47	78
Salem	240	136	131	40	389	64	83	47	-	118
Seaside	356	17	247	155	423	181	117	78	118	-

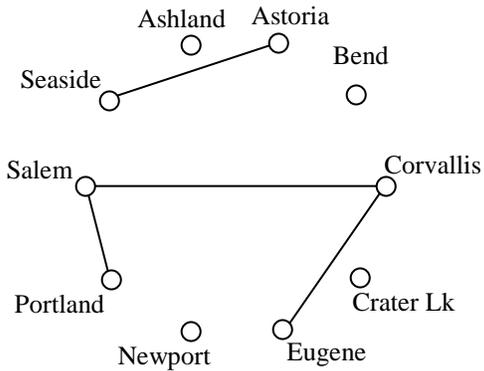
Using NNA with a large number of cities, you might find it helpful to mark off the cities as they're visited to keep from accidentally visiting them again. Looking in the row for Portland, the smallest distance is 47, to Salem. Following that idea, our circuit will be:

Portland to Salem	47
Salem to Corvallis	40
Corvallis to Eugene	47
Eugene to Newport	91
Newport to Seaside	117
Seaside to Astoria	17
Astoria to Bend	255
Bend to Ashland	200
Ashland to Crater Lake	108
Crater Lake to Portland	344
Total trip length:	1266 miles

Using Sorted Edges, you might find it helpful to draw an empty graph, perhaps by drawing vertices in a circular pattern. Adding edges to the graph as you select them will help you visualize any circuits or vertices with degree 3.

- Seaside to Astoria 17 miles
- Corvallis to Salem 40 miles
- Portland to Salem 47 miles
- Corvallis to Eugene 47 miles

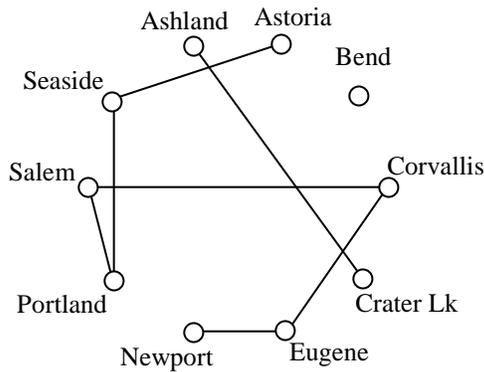
The graph after adding these edges is shown to the right. The next shortest edge is from Corvallis to Newport at 52 miles, but adding that edge would give Corvallis degree 3.



Continuing on, we can skip over any edge pair that contains Salem or Corvallis, since they both already have degree 2.

- Portland to Seaside 78 miles
- Eugene to Newport 91 miles
- Portland to Astoria (reject – closes circuit)
- Ashland to Crater Lk 108 miles

The graph after adding these edges is shown to the right. At this point, we can skip over any edge pair that contains Salem, Seaside, Eugene, Portland, or Corvallis since they already have degree 2.

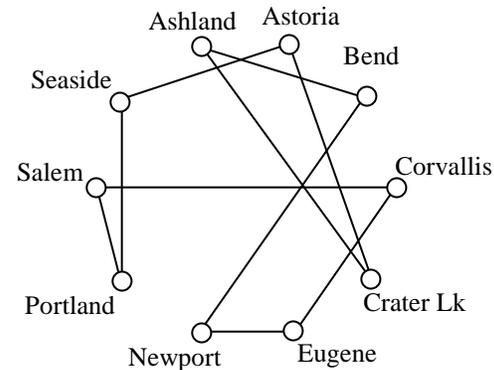


- Newport to Astoria (reject – closes circuit)
- Newport to Bend 180 miles
- Bend to Ashland 200 miles

At this point the only way to complete the circuit is to add:
Crater Lk to Astoria 433 miles

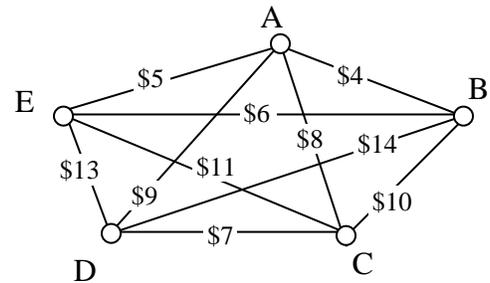
Total trip length: 1241 miles

Neither algorithm produced the optimal route. The following route can make the tour in 1069 miles:
Ashland, Crater Lake, Bend, Salem, Portland, Astoria, Seaside, Newport, Corvallis, Eugene, Ashland



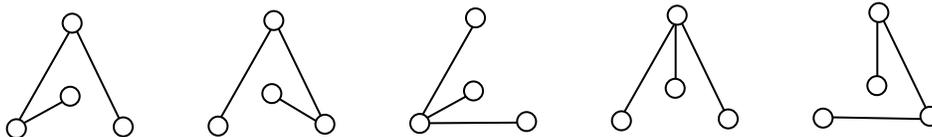
Spanning Trees

A company requires reliable internet and phone connectivity between their five offices (named A, B, C, D, and E for simplicity) in New York, so they decide to lease dedicated lines from the phone company. The phone company will charge for each link made. The costs, in thousands of dollars per year, are shown in the graph.



In this case, we don't need to find a circuit, or even a path; all we need to do is make sure there is a way to get from any vertex to any other vertex.

A **spanning tree** is a connected graph using all vertices in which there are no circuits. In other words, there is a path from any vertex to any other vertex, but no circuits. Some examples of spanning trees are shown below. Notice there are no circuits in the trees, and it is fine to have vertices with degree higher than two.



Usually we have a starting graph to work from, like in the phone example above. In this case, we form our spanning tree by finding a **subgraph** – a new graph formed using all the vertices and some of the edges from the original graph. No edges will be created where they didn't already exist.

Of course, any random spanning tree isn't really what we want. We want the **minimum cost spanning tree (MCST)**, the spanning tree with the smallest total edge weight. A nearest neighbor style approach doesn't make as much sense here since we don't need a circuit, so instead we will take an approach similar to sorted edges.

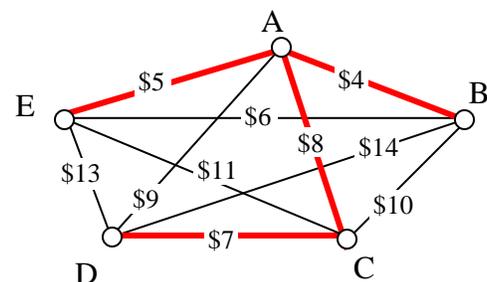
Kruskal's Algorithm

- 1) Select the cheapest unused edge in the graph; highlight it.
- 2) Repeat step 1, adding the cheapest unused edge to the graph, unless:
 - a. adding the edge would create a circuit
- 3) Repeat until a spanning tree is formed

Example: Using our phone line graph from above, begin adding edges:

AB	\$4	OK
AE	\$5	OK
BE	\$6	reject – closes circuit ABEA
DC	\$7	OK
AC	\$8	OK

At this point we stop – every vertex is now connected, so we have formed a spanning tree with cost \$24 thousand a year.



Remarkably, Kruskal’s algorithm is both optimal and efficient; we are guaranteed to produce the optimal MCST.

Example: The power company needs to lay updated distribution lines connecting the ten Oregon cities below to the power grid. How can they minimize the amount of new line to lay?

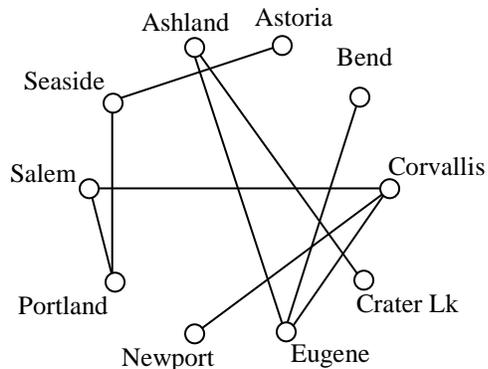
	Ashland	Astoria	Bend	Corvallis	Crater Lake	Eugene	Newport	Portland	Salem	Seaside
Ashland	-	374	200	223	108	178	252	285	240	356
Astoria	374	-	255	166	433	199	135	95	136	17
Bend	200	255	-	128	277	128	180	160	131	247
Corvallis	223	166	128	-	430	47	52	84	40	155
Crater Lake	108	433	277	430	-	453	478	344	389	423
Eugene	178	199	128	47	453	-	91	110	64	181
Newport	252	135	180	52	478	91	-	114	83	117
Portland	285	95	160	84	344	110	114	-	47	78
Salem	240	136	131	40	389	64	83	47	-	118
Seaside	356	17	247	155	423	181	117	78	118	-

Using Kruskal’s algorithm, we add edges from cheapest to most expensive:

Seaside to Astoria 17 miles
 Corvallis to Salem 40 miles
 Portland to Salem 47 miles
 Corvallis to Eugene 47 miles
 Corvallis to Newport 52 miles
 Salem to Eugene reject – closes circuit
 Portland to Seaside 78 miles
 Newport to Salem reject – closes circuit
 Corvallis to Portland reject – closes circuit
 Eugene to Newport reject – closes circuit
 Portland to Astoria reject – closes circuit
 Ashland to Crater Lk 108 miles
 Eugene to Portland reject – closes circuit
 Newport to Portland reject – closes circuit
 Newport to Seaside reject – closes circuit
 Salem to Seaside reject – closes circuit
 Bend to Eugene 128 miles

Bend to Salem reject – closes circuit
 Astoria to Newport reject – closes circuit
 Salem to Astoria reject – closes circuit
 Corvallis to Seaside reject – closes circuit
 Portland to Bend reject – closes circuit
 Astoria to Corvallis reject – closes circuit
 Eugene to Ashland 178 miles

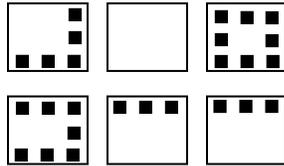
This connects the graph. The total length of cable to lay would be 695 miles.



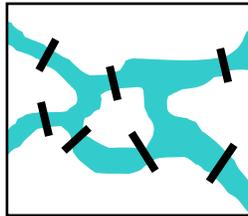
Exercises

Skills

- To deliver mail in a particular neighborhood, the postal carrier needs to walk to along each of the streets with houses (the dots). Create a graph with edges showing where the carrier must walk to deliver the mail.



- Suppose that a town has 7 bridges as pictured below. Create a graph that could be used to determine if there is a path that crosses all bridges once.



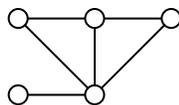
- The table below shows approximate driving times (in minutes, without traffic) between five cities in the Dallas area. Create a weighted graph representing this data.

	Plano	Mesquite	Arlington	Denton
Fort Worth	54	52	19	42
Plano		38	53	41
Mesquite			43	56
Arlington				50

- Shown in the table below are the one-way airfares between 5 cities⁶. Create a graph showing this data.

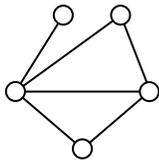
	Honolulu	London	Moscow	Cairo
Seattle	\$159	\$370	\$654	\$684
Honolulu		\$830	\$854	\$801
London			\$245	\$323
Moscow				\$329

- Find the degree of each vertex in the graph below.

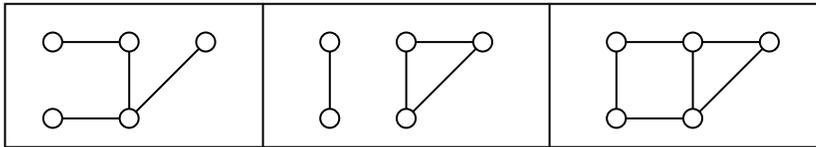


⁶ Cheapest fares found when retrieved Sept 1, 2009 for travel Sept 22, 2009

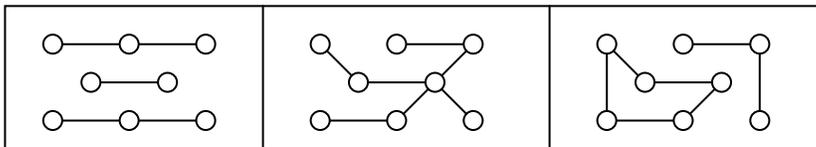
6. Find the degree of each vertex in the graph below.



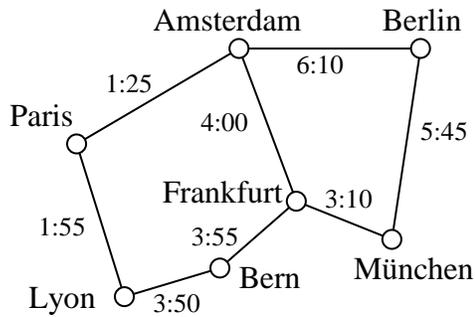
7. Which of these graphs are connected?



8. Which of these graphs are connected?

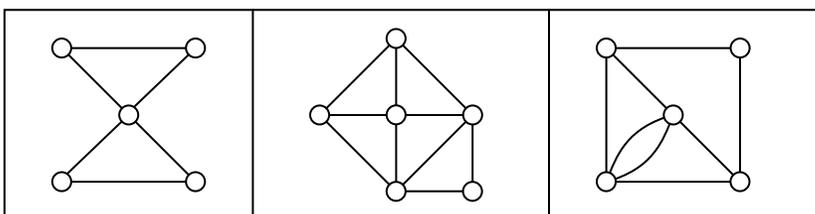


9. Travel times by rail for a segment of the Eurail system is shown below with travel times in hours and minutes⁷. Find path with shortest travel time from Bern to Berlin by applying Dijkstra's algorithm.



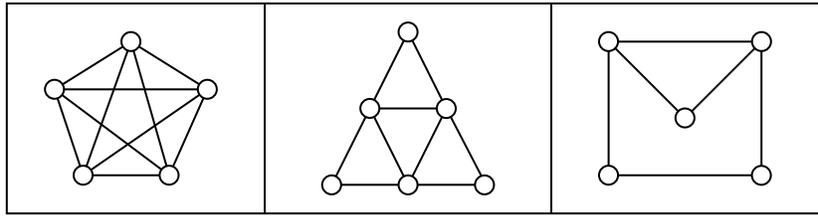
10. Using the graph from the previous problem, find the path with shortest travel time from Paris to München.

11. Does each of these graphs have an Euler circuit? If so, find it.

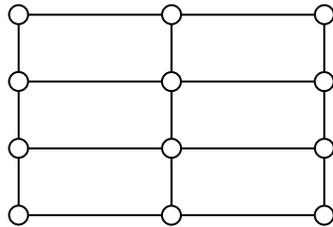


⁷ From <http://www.eurail.com/eurail-railway-map>

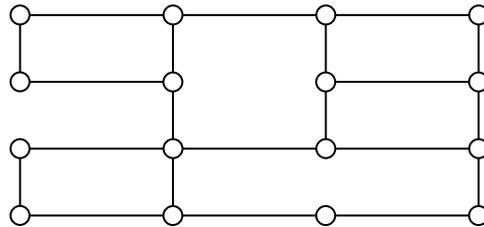
12. Does each of these graphs have an Euler circuit? If so, find it.



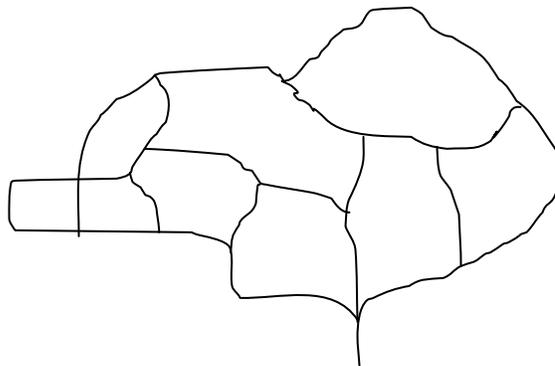
13. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.



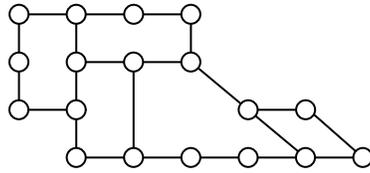
14. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.



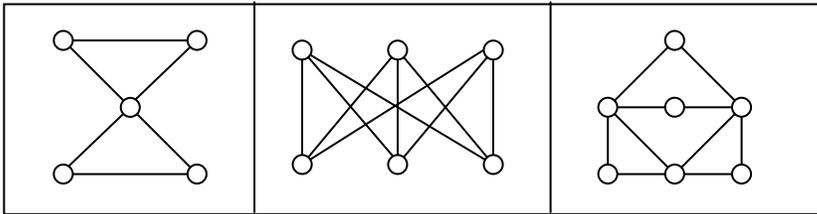
15. The maintenance staff at an amusement park need to patrol the major walkways, shown in the graph below, collecting litter. Find an efficient patrol route by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.



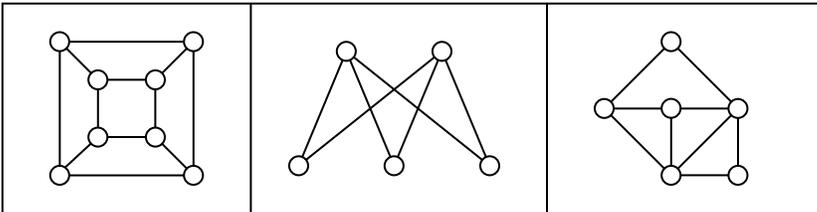
16. After a storm, the city crew inspects for trees or brush blocking the road. Find an efficient route for the neighborhood below by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.



17. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.



18. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.

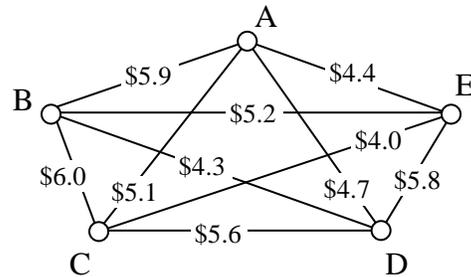


19. A company needs to deliver product to each of their 5 stores around the Dallas, TX area. Driving distances between the stores are shown below. Find a route for the driver to follow, returning to the distribution center in Fort Worth:
- Using Nearest Neighbor starting in Fort Worth
 - Using Repeated Nearest Neighbor
 - Using Sorted Edges

	Plano	Mesquite	Arlington	Denton
Fort Worth	54	52	19	42
Plano		38	53	41
Mesquite			43	56
Arlington				50

20. A salesperson needs to travel from Seattle to Honolulu, London, Moscow, and Cairo. Use the table of flight costs from problem #4 to find a route for this person to follow:
- Using Nearest Neighbor starting in Seattle
 - Using Repeated Nearest Neighbor
 - Using Sorted Edges

21. When installing fiber optics, some companies will install a sonet ring; a full loop of cable connecting multiple locations. This is used so that if any part of the cable is damaged it does not interrupt service, since there is a second connection to the hub. A company has 5 buildings. Costs (in thousands of dollars) to lay cables between pairs of buildings are shown below. Find the circuit that will minimize cost:
- Using Nearest Neighbor starting at building A
 - Using Repeated Nearest Neighbor
 - Using Sorted Edges



22. A tourist wants to visit 7 cities in Israel. Driving distances between the cities are shown below⁸. Find a route for the person to follow, returning to the starting city:
- Using Nearest Neighbor starting in Jerusalem
 - Using Repeated Nearest Neighbor
 - Using Sorted Edges

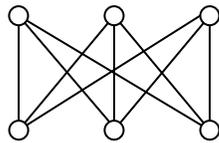
	Jerusalem	Tel Aviv	Haifa	Tiberias	Beer Sheba	Eilat
Jerusalem	--					
Tel Aviv	58	--				
Haifa	151	95	--			
Tiberias	152	134	69	--		
Beer Sheba	81	105	197	233	--	
Eilat	309	346	438	405	241	--
Nazareth	131	102	35	29	207	488

- Find a minimum cost spanning tree for the graph you created in problem #3
- Find a minimum cost spanning tree for the graph you created in problem #22
- Find a minimum cost spanning tree for the graph from problem #21

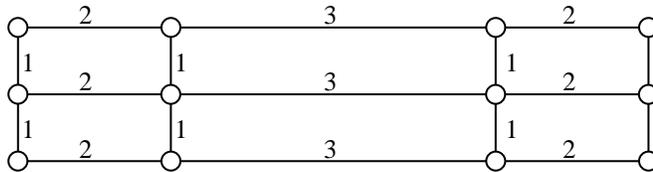
⁸ From <http://www.ddtravel-acc.com/Israel-cities-distance.htm>

Concepts

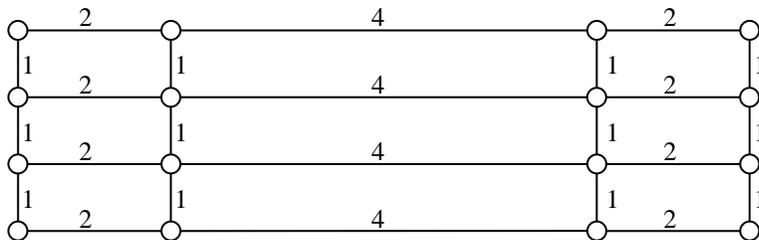
- 26. Can a graph have one vertex with odd degree? If not, are there other values that are not possible? Why?
- 27. A complete graph is one in which there is an edge connecting every vertex to every other vertex. For what values of n does complete graph with n vertices have an Euler circuit? A Hamiltonian circuit?
- 28. Create a graph by drawing n vertices in a row, then another n vertices below those. Draw an edge from each vertex in the top row to every vertex in the bottom row. An example when $n=3$ is shown below. For what values of n will a graph created this way have an Euler circuit? A Hamiltonian circuit?



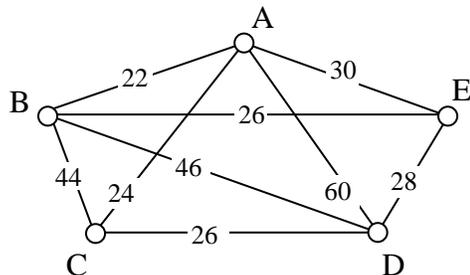
- 29. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



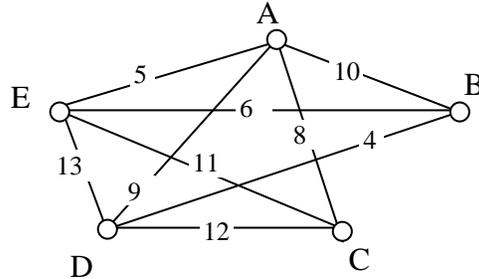
- 30. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



- 31. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



32. Eulerize this graph in the most efficient way possible, considering the weights of the edges.

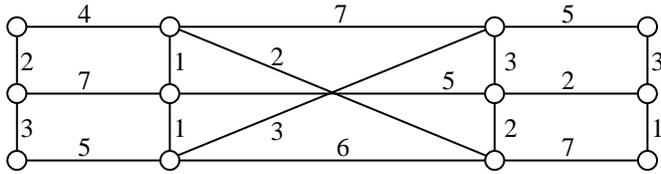


Explorations

33. Social networks such as Facebook and MySpace can be represented using graphs in which vertices represent people and edges are drawn between two vertices when those people are “friends.” The table below shows a friendship table, where an X shows that two people are friends.
- Create a graph of this friendship table
 - Find the shortest path from A to D. The length of this path is often called the “degrees of separation” of the two people.
 - Extension: Split into groups. Each group will pick 10 or more movies, and look up their major actors (www.imdb.com is a good source). Create a graph with each actor as a vertex, and edges connecting two actors in the same movie (note the movie name on the edge). Find interesting paths between actors, and quiz the other groups to see if they can guess the connections.
34. A spell checker in a word processing program makes suggestions when it finds a word not in the dictionary. To determine what words to suggest, it tries to find similar words. One measure of word similarity is the Levenshtein distance, which measures the number of substitutions, additions, or deletions that are required to change one word into another. For example, the words spit and spot are a distance of 1 apart; changing spit to spot requires one substitution (i for o). Likewise, spit is distance 1 from pit since the change requires one deletion (the s). The word spite is also distance 1 from spit since it requires one addition (the e). The word soot is distance 2 from spit since two substitutions would be required.
- Create a graph using words as vertices, and edges connecting words with a Levenshtein distance of 1. Use the misspelled word “moke” as the center, and try to find at least 10 connected dictionary words. How might a spell checker use this graph?
 - Improve the method from above by assigning a weight to each edge based on the likelihood of making the substitution, addition, or deletion. You can base the weights on any reasonable approach: proximity of keys on a keyboard, common language errors, etc. Use Dijkstra’s algorithm to find the length of the shortest path from each word to “moke”. How might a spell checker use these values?

35. The graph below contains two vertices of odd degree. To eulerize this graph, it is necessary to duplicate edges connecting those two vertices.

- a. Use Dijkstra's algorithm to find the shortest path between the two vertices with odd degree. Does this produce the most efficient eulerization and solve the Chinese Postman Problem for this graph?



- b. Suppose a graph has n odd vertices. Using the approach from part a, how many shortest paths would need to be considered? Is this approach going to be efficient?