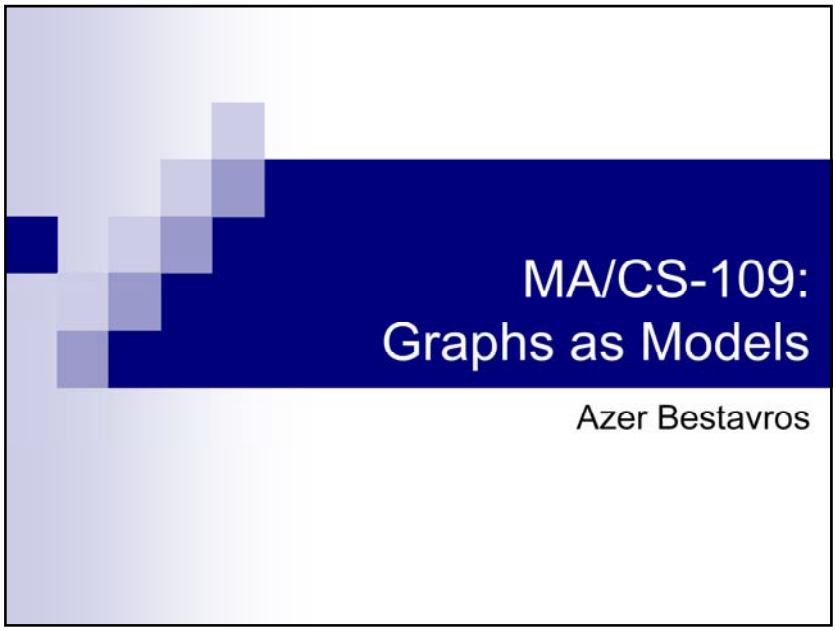
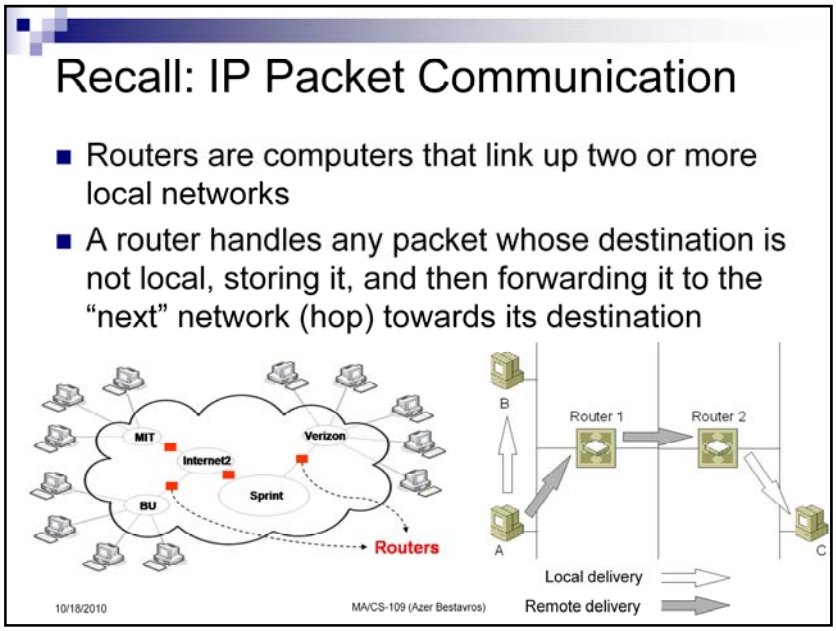
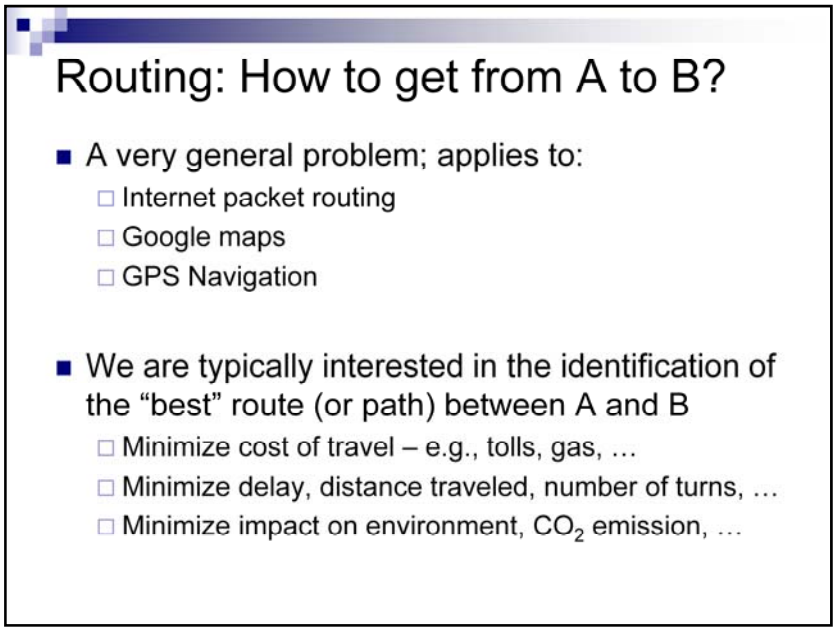

MA/CS-109:
Graphs as Models
Azer Bestavros



Recall that routers are computers that link up **two** or more local networks. A router handles any packet whose destination is not local, storing it, and then forwarding it to the "next" network (hop) towards its destination. But how does a router know where to send a packet on towards its destination? What is needed is a way for the router to figure out the path that the packet should take towards its destination and then forward that packet to the next router on that path, accordingly. This functionality is called "routing" (hence the name "router") and it is what we need to implement for the Internet to deliver packets from a source to a destination.

Routing is the process of figuring out how to go from point A to point B given how the various networks are connected to one another.



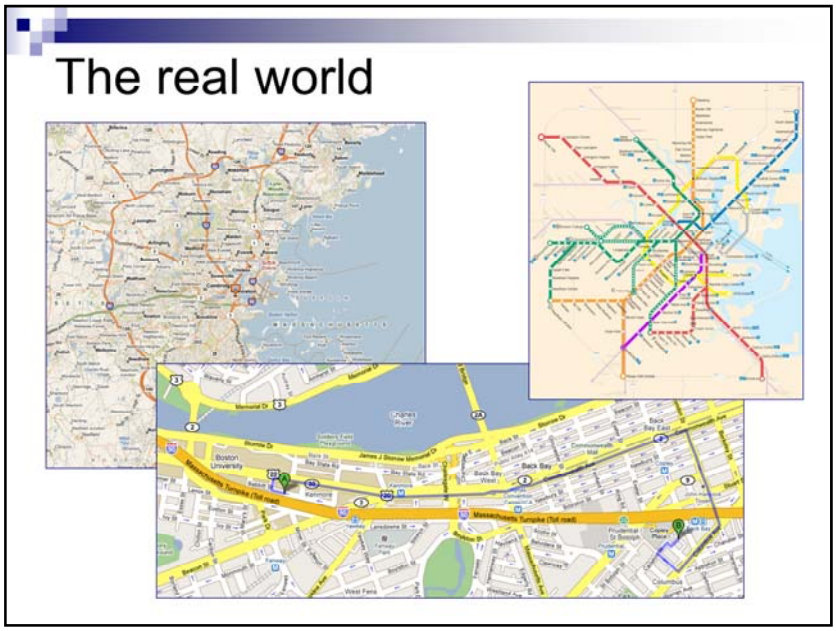
A very general problem that we often encounter in many settings is that of finding "the best way" to get from point A to point B when navigating in some space. For example:

1. As we saw in our coverage of how we are able to send data over the Internet, an Internet Protocol (IP) packet needs to be routed from a sender to a receiver (through interconnected routers).
2. Mapping services, such as MapQuest, Google Maps, or Bing must be able to compute directions from a starting address to a destination, given a map of roads.
3. GPS devices must be able to compute directions from a starting address to a destination, given a map of roads as well as other information, such as traffic conditions, etc.

In all of these examples, there are clearly many ways (typically a very large number of ways) to get from point A to point B. Of all these "ways" (which we call paths), we are typically interested in identifying only the "best" path according to some criterion, which we typically think of as the cost of the path. Examples of costs include:

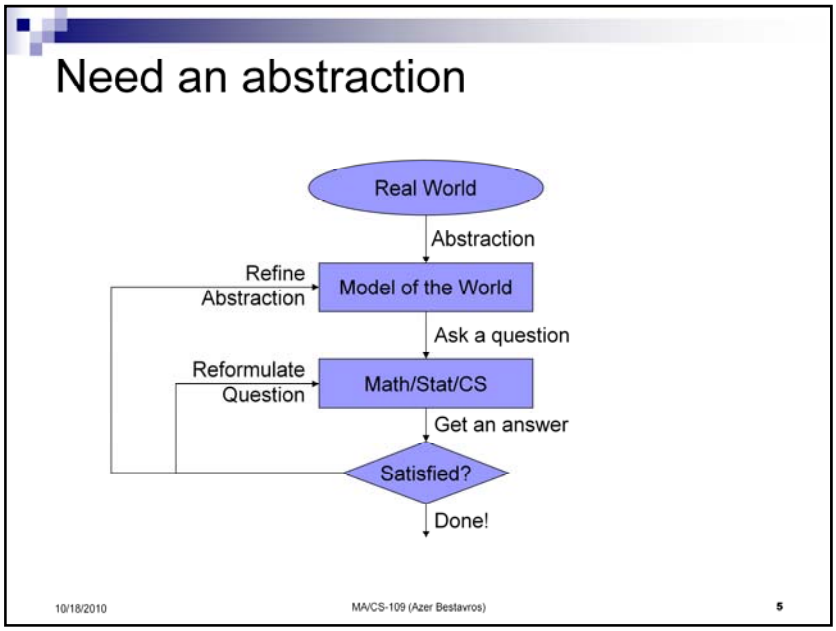
- Minimize the real cost of travel. Here the cost is "dollars" paid, e.g., in tolls, gas, …
- Minimize the time it takes to get from A to B. Here the cost is the "delay".
- Minimize the distance traveled to get from A to B. Here the cost is the "distance". Notice that minimizing distance does not necessarily minimize delay (why?)
- Minimize the difficulty of following directions from A to B to get an "easy" route. Here the cost is the "number of turns".
- Minimize the carbon footprint of travel from A to B. Here the cost is the "gallons of gas".

In general, one may define any arbitrary objective (expressed as a cost) and the goal of routing would be to minimize such cost.



The routes we will evaluate come from a “real” world (e.g., of streets connecting intersections as portrayed on a street map, or of subway lines and stops as portrayed on a subway map).

In order to reason about this “real world” (of streets and subway stops) we need to make it simpler – retaining only information pertinent to the problem at hand.



So as we have done many times in this course, we will have to work with a “model” or an abstraction of the real world. We do so using graphs.

Let’s go through an example.

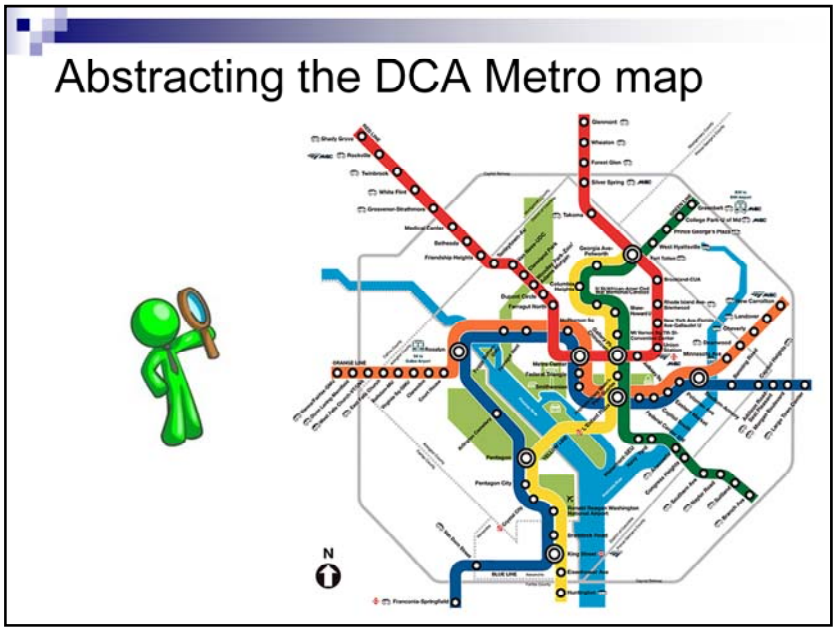
Computing the shortest path

- Problem: Can you find the minimum-cost path from a start location to all possible destinations?
- Cost: Number of subway stops along the way.

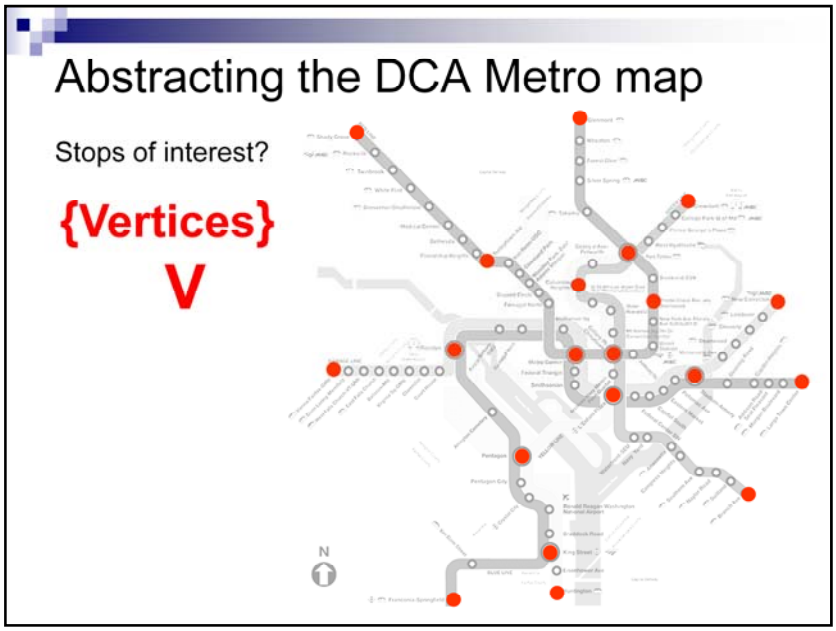
Subway Map of Washington DC

Consider the subway (Metro) map of Washington D.C. and assume that the task at hand is to compute the "best" path (also called shortest path) from a given starting Metro station to all other Metro stations in D.C.

Moreover, let's assume that for our purposes, we will consider the *number of stops* to be our cost. In other words, we want to minimize the number of stops that we go from a starting station to a destination.



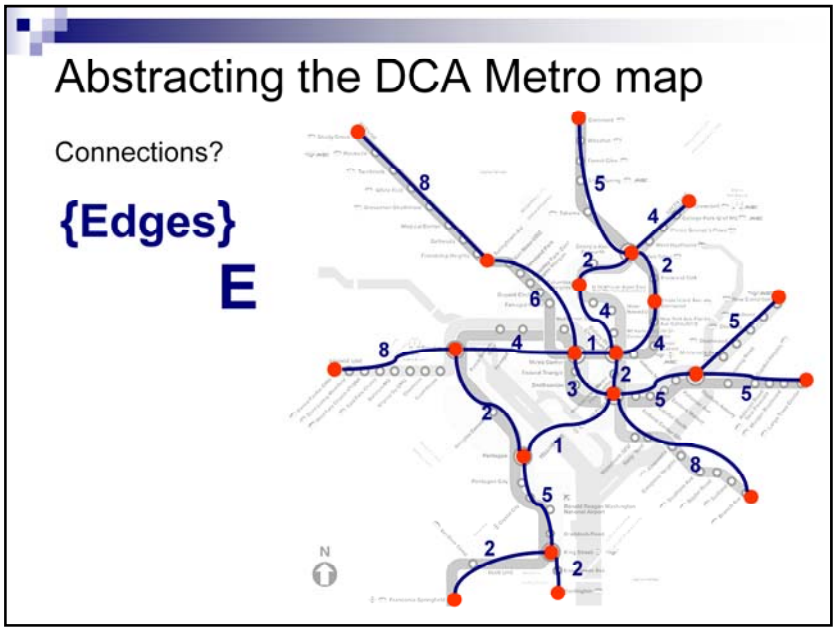
What information do we need to retain from the Metro map to allow us to reason about the problem at hand (that of finding the shortest path from a given starting Metro station to all other Metro stations in D.C. ?



First, we will need to identify the Metro stations that interest us. For example, it may be the case that there are Metro stations that we will never consider as a destination. Clearly, such stations need not remain in our model of the DC Metro.

This step gives us a set of **vertices** which we denote with the letter **V**.

Notice that words such as “nodes” and “points” are often used to mean “vertices”. In these notes, we will try to stick to “vertices”, noting that in some cases, the use of “nodes” (e.g., of a network) may be more natural.

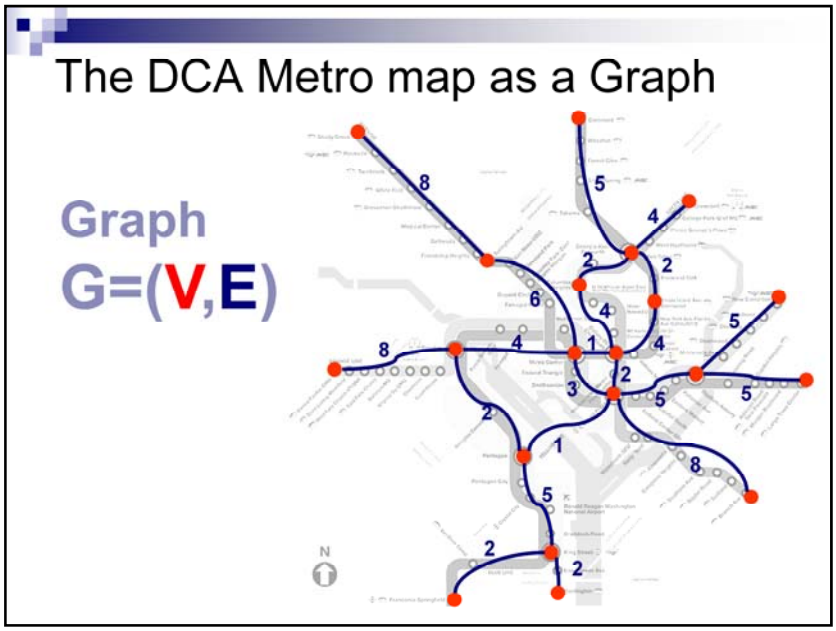


Next, we identify the specific connections between the set of vertices V.

A vertex (station) is connected to another if one can get from the first to the second vertex without going through any other vertex. For each such connection, we also need to identify the cost associated with it. In our example, we are taking the cost to be the number of intermediate stops.

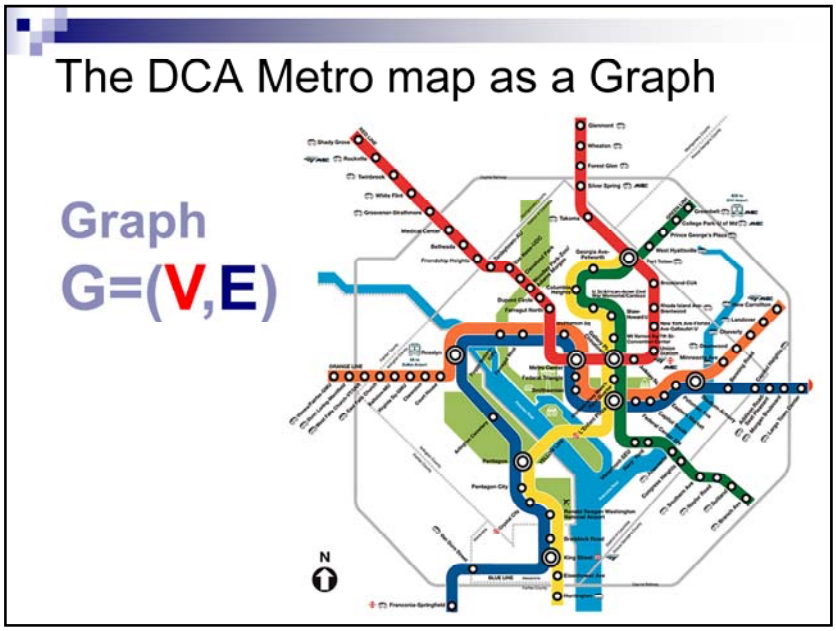
In our model, we will represent such connections with a set of **Edges**, which we denote using the letter **E**.

Notice that words such as “links”, “arcs”, and “lines” are sometimes used to mean “edges”. In these notes, we will try to stick to “edges”, noting that in some cases, the use of “links” (e.g., between web pages, or between routers) may be more natural.

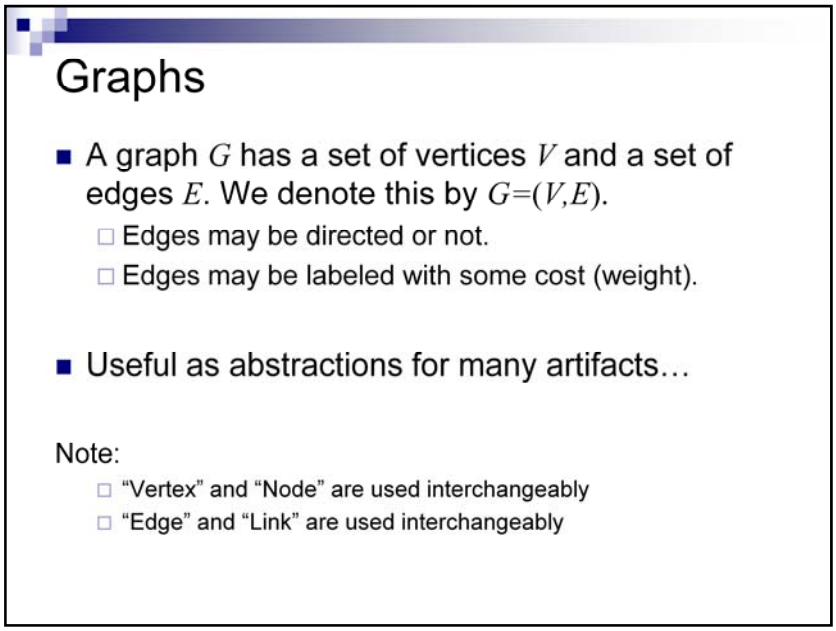


Together the set of vertices (nodes) and edges (links) define a **Graph** which we denote by the letter **G**.

Once we have the graph G, we need not retain the details of the “real world” anymore. Indeed, details such as the geographic location of the stations on the map, or the orientation (which way is north), etc. are all details that we “abstract out”.



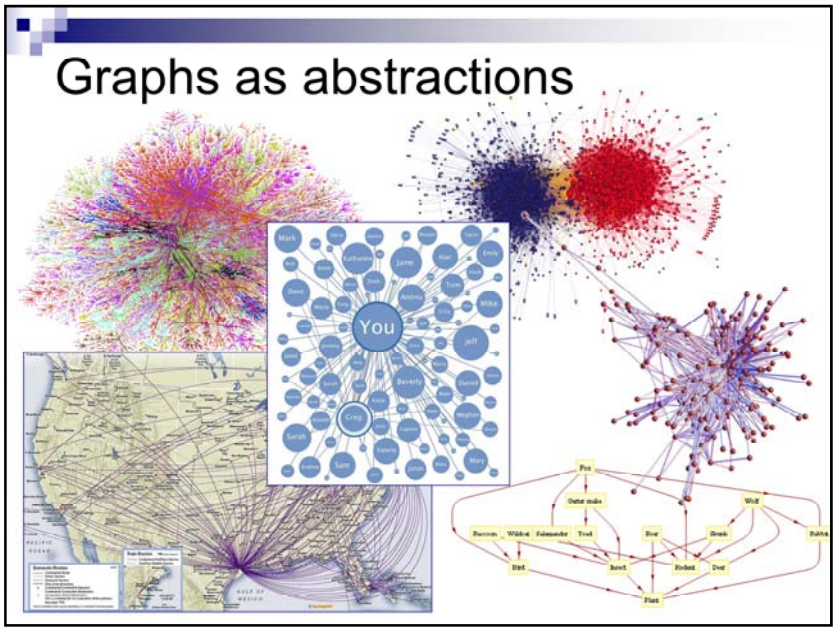
To summarize, by identifying a set of vertices and the edges between them (along with the costs of such edges) we have modeled the DC Metro Map as a graph.



A few additional notes on graphs:

- The edges of a graph may be directed or undirected (not directed). A directed edge is one where one can go in one direction (on that edge) but not necessarily in the other direction. An example of a directed edge would be a one-way street connecting an intersection to another. Notice that directed edges are sometimes called "directional" edges and those that are not directed are sometimes called "bi-directional". A graph whose edges are directed is called a **Directed Graph**.
- An edge may be labeled with a cost, which we also call a weight. In this course, we will concern ourselves with positive costs, but in general the cost of traversing an edge may be negative (a negative cost means "profit") or zero. Can you think of example graphs where costs may be negative?

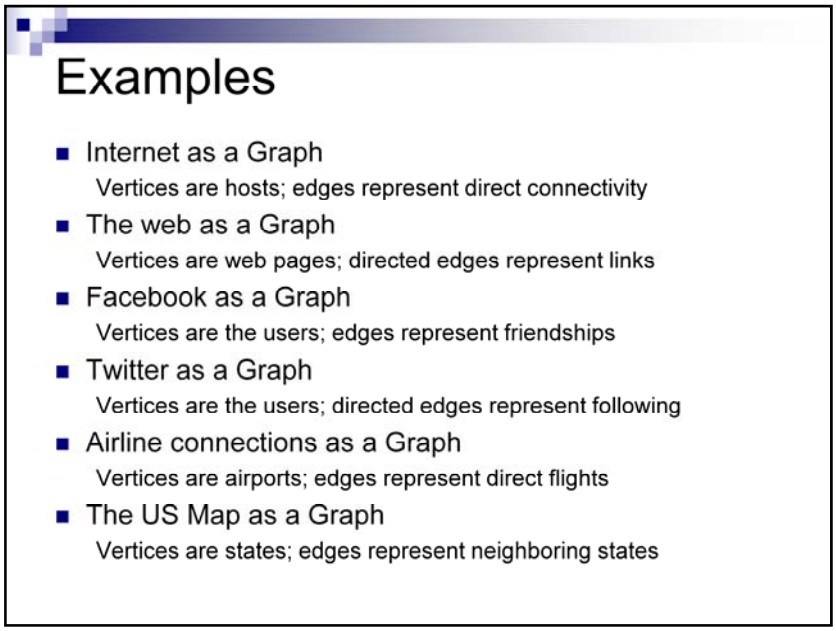
Graphs are fantastic mathematical objects that come in very handy in modeling many artifacts.



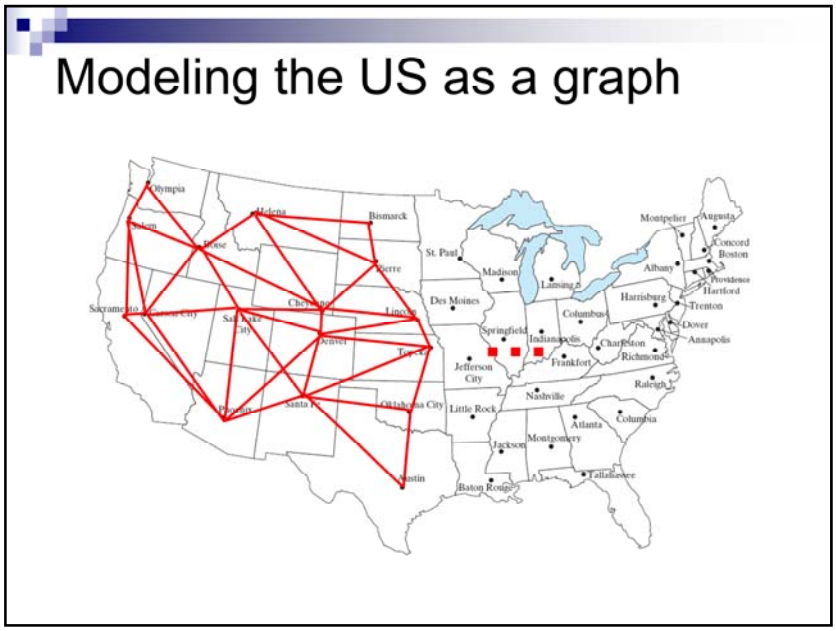
Let's examine a few such artifacts, illustrated in the visualizations above – namely, the Internet connectivity, blogs, airline travel routes, relationships defined by social networks, food chain, etc.

For each such artifact (and many more), we can model the underlying real-world using a graph. To do so, we need to answer the following questions:

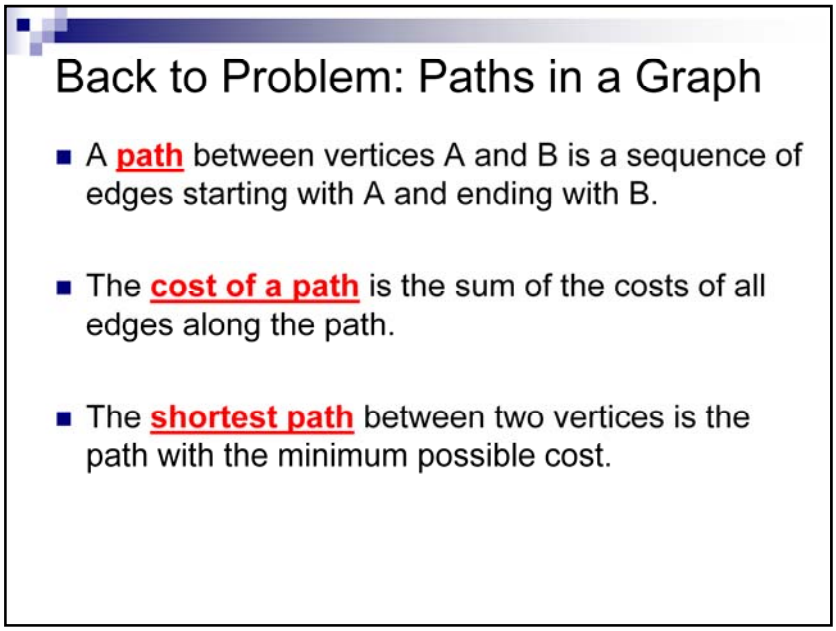
1. What are the vertices of the graph model
2. What are the edges of the graph model (what do they represent)
3. Are the edges directed or not?
4. What is the cost of each edge (if needed)?



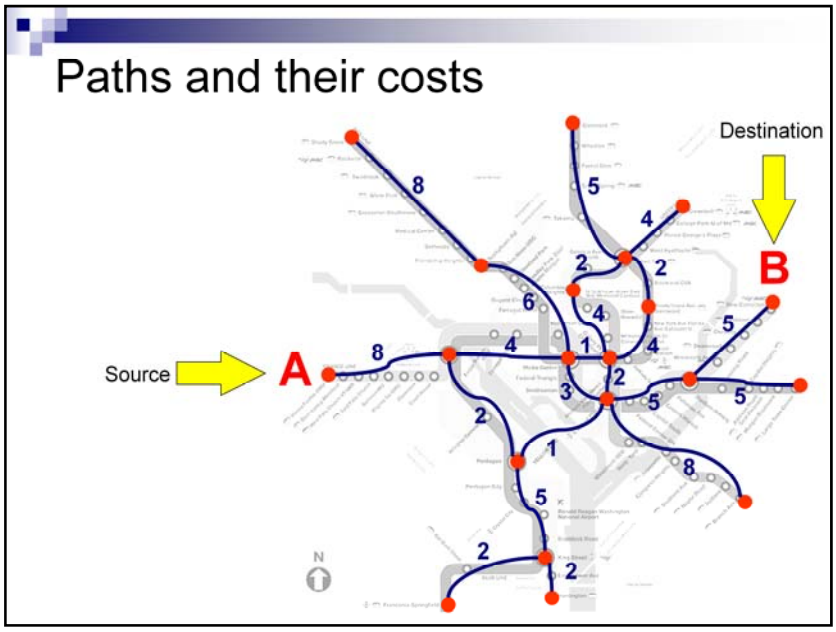
In these examples, we note that some graphs are directed (e.g., the web and twitter graphs) while others are not (e.g., the Facebook and US Map graphs). Recall that a directed graph is one where edges have a specific direction. For example, when a web page points to another web page, then clearly this linking is directional. In particular, if web page A points to web page B, then web page B does not necessarily point back to web page A. The same observation can be made about Twitter: user A following user B does not necessarily mean that user B follows user A. On the other hand, a friendship relationship underscored by an edge in the Facebook graph, or a neighboring state relationship underscored by an edge in the US Map graph are examples of bidirectional edges – if A is a friend of B, then B is a friend of A, and if state A shares a border with state B, the state B shares a border with A.



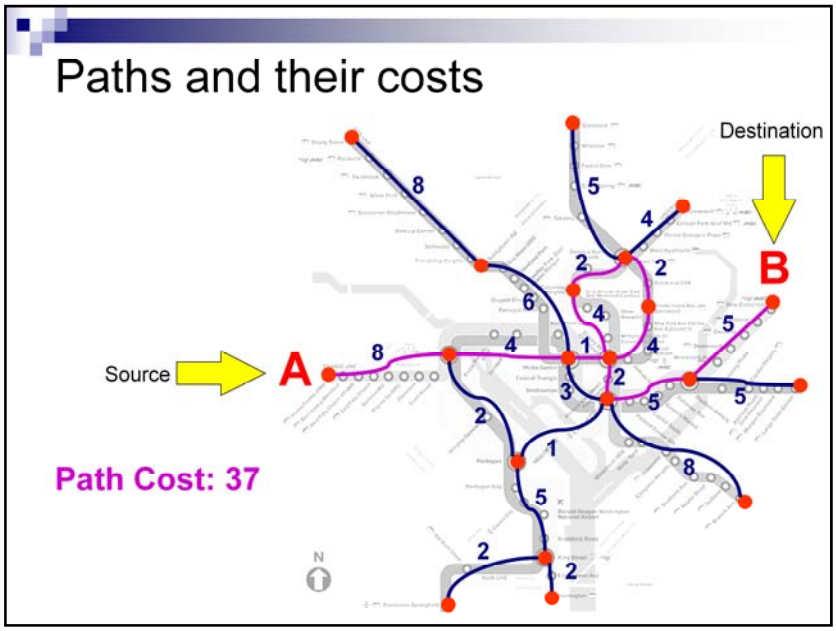
What are the vertices? What are the edges? What are possible costs (labels) to put on the edges?



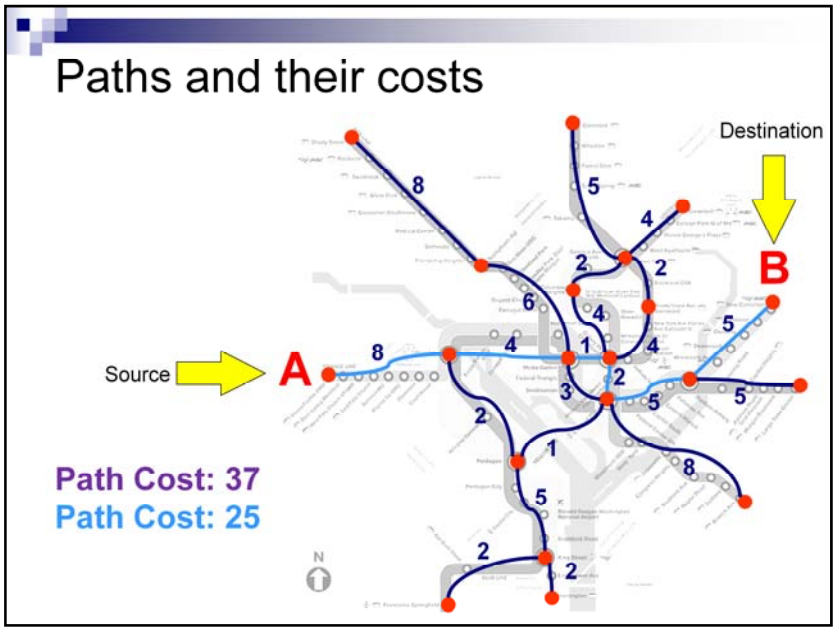
We now turn our attention back to the original problem of finding the best way to go from one vertex to another. To do so, we need to define what we mean by a path, the cost of a path, and what constitutes the shortest path.



We exemplify these terms on the DC Metro graph for going from A to B.

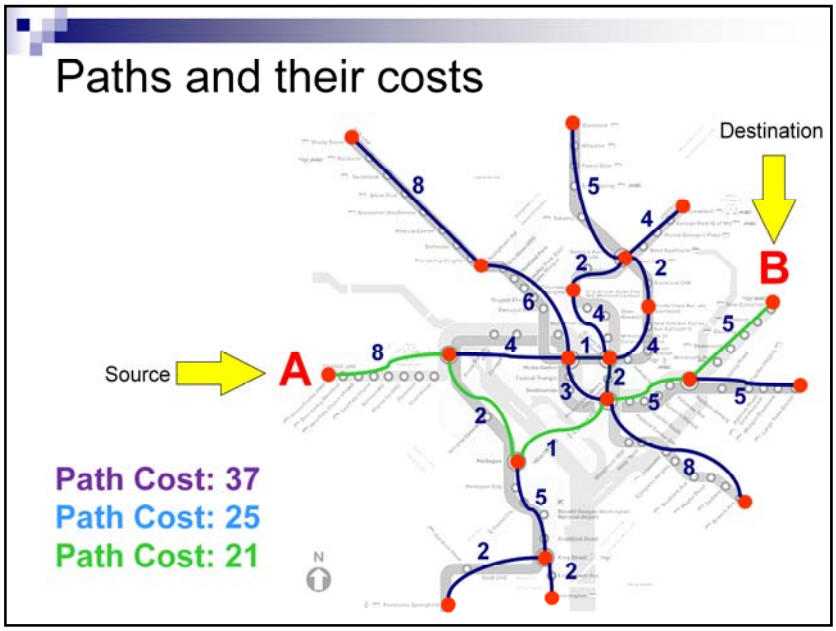


Here is one path whose cost is 37. Notice that this path goes through one vertex twice! While doing so is “legal” in the sense that a path may have a vertex appear more than once, it is clearly not a desirable thing to go around in “circles”. In particular, going around in circles is not a good way to minimize the cost of a path!



Removing the “going around in a circle” we get a second path, whose cost is lower than the first (as expected). But, is this the shortest path? Is this the best we can do.

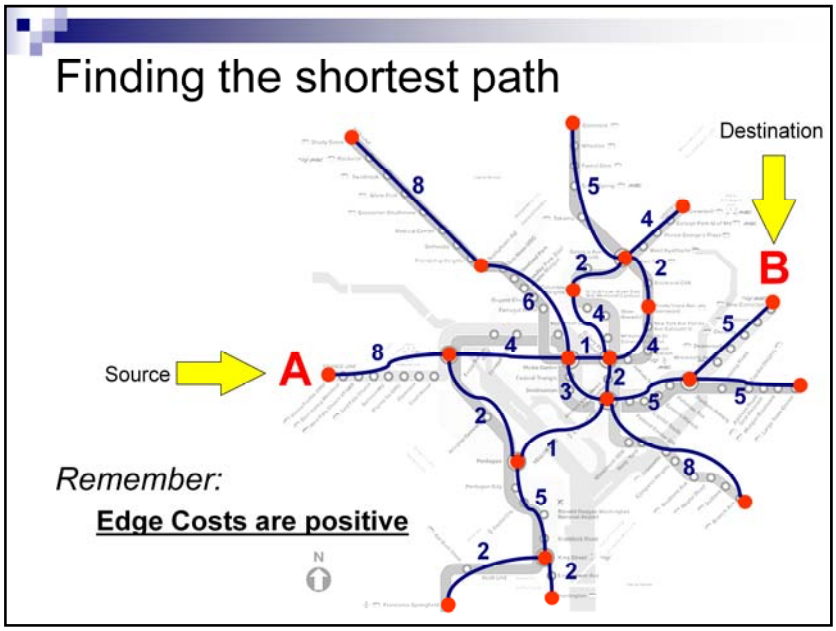
Not really!



We can do better as shown in this third path, reducing the cost to 21.

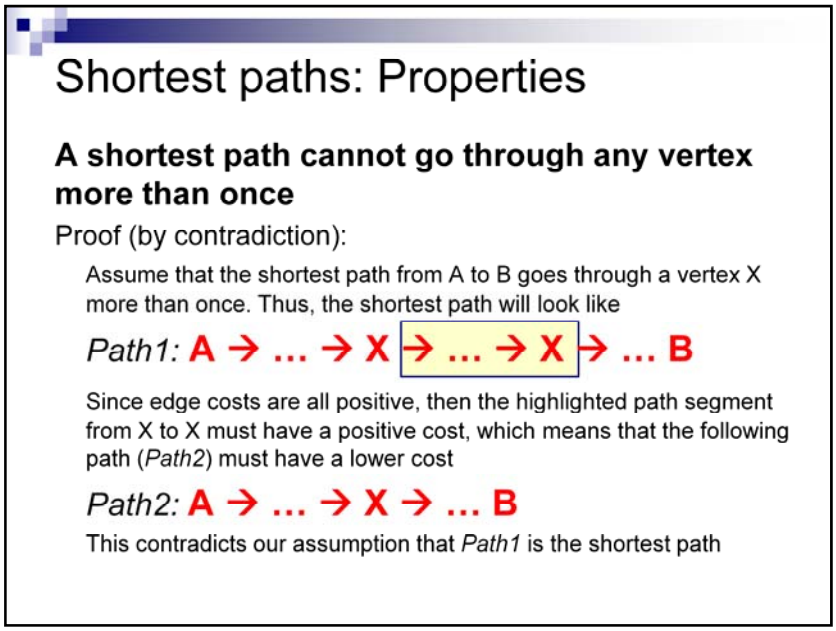
Notice that while on the illustration, this new path "looks" longer, it really isn't because what we consider as cost has nothing to do with where we draw the vertices. Recall that our use of "number of stops" as the cost implied that we don't care about physical distances – so we would take a path of length 10 miles with one stop over a path of length 2 miles with two stops. Needless to say, you may not "like" using the number of stops as the determinant of cost, and you may opt for something else. Doing so will result in a different graph and the shortest path from A to B in that graph may be different from the one we get here.

So, we now have a path that costs 21. Is this the lowest-cost (shortest) path from A to B? Well, given the toy-nature of this graph, we may be able to convince ourselves that this is the one… But, what we need is a systematic way (i.e., an algorithm) to find the shortest path in any arbitrary graph.



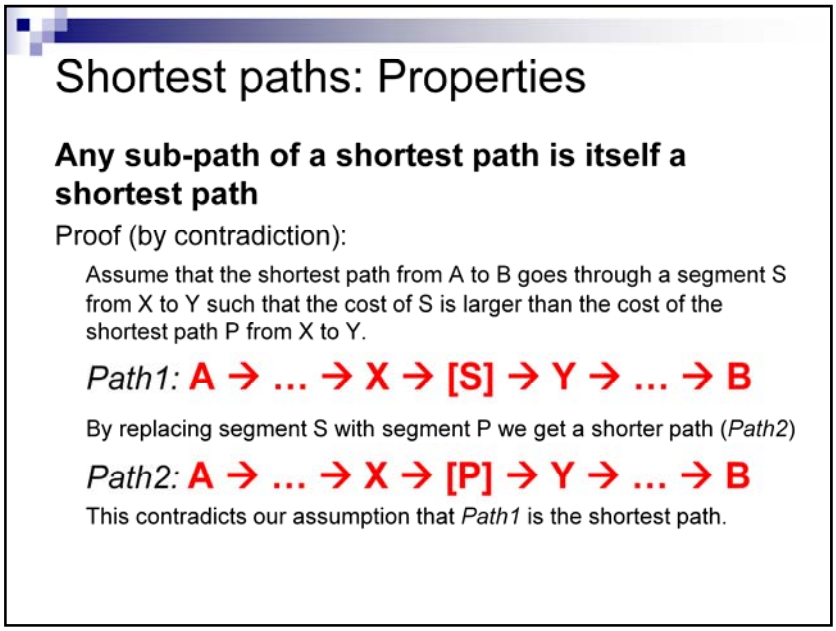
Before doing so, let's summarize our observations about the properties of shortest paths.

Recall, that we restrict our attention to graphs with edges that have positive costs – i.e., the cost on an edge cannot be negative or even zero.



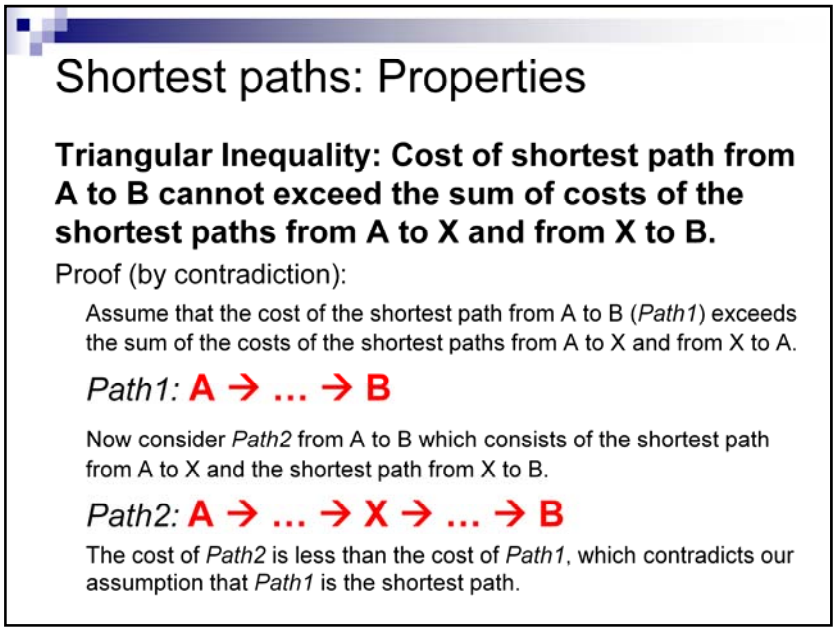
The first property formalizes our observation that shortest paths should not admit cycles – going in circles is not what one wants to do to minimize cost!

The proof is by contradiction.



The second property formalizes our observation that any segment between two intermediate vertices on a shortest path must itself be the shortest path between these two vertices – if the shortest path from Boston to NYC goes through Harford and New Haven, then the segment of that path from Hartford to New Haven must be the shortest path between Harford and New Haven.

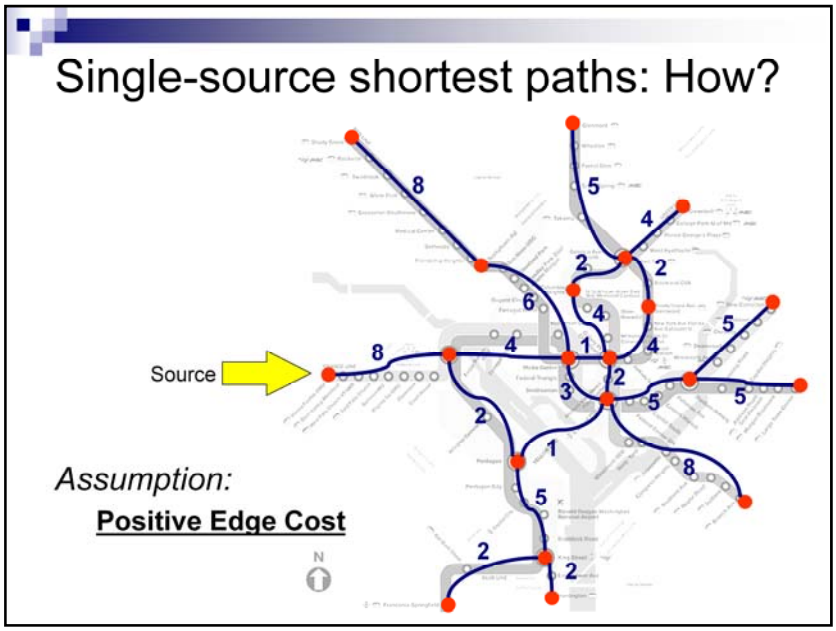
The proof is again by contradiction.



The last property of interest is the so-called “triangular inequality” of paths in a graph, which states that the shortest path from A to B must have a cost that is no higher than the costs of going from A to a transit point X and then from that transit point X to B.

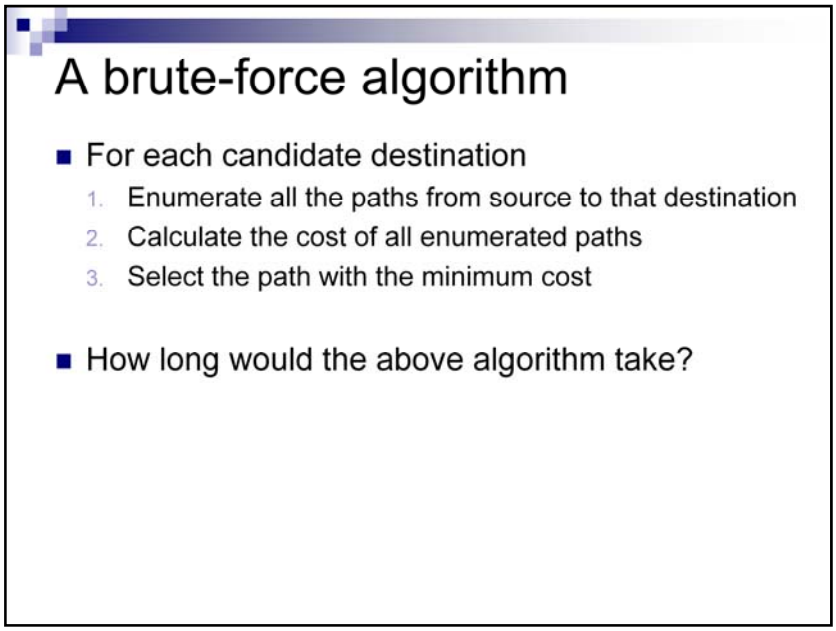
This property formalizes what we may already know, but perhaps never bothered to prove ☺: By taking a detour from a shortest path, the resulting cost cannot be lower!

The proof of this property is also by contradiction.



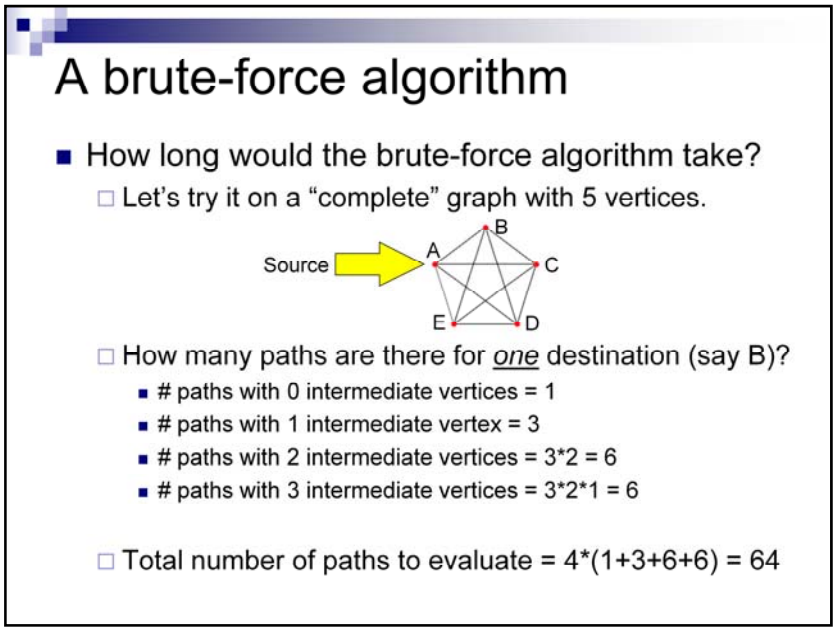
Armed with a good understanding of what shortest paths are, and what properties they exhibit, we now go back to answering the "how" question: Given a graph with positive edge costs, **how** do we find the shortest path? This question is about the "algorithm" one needs to follow to find shortest paths.

Instead of finding the shortest path from a specific starting point (source) to a specific destination, our goal will be to **find all the shortest paths from a specific vertex to all other vertices in the graph**.



Well, the first natural approach to follow would be to enumerate all the paths from the starting point, figure out how much each such path costs, and then for each destination find the path with the minimum cost... That's the algorithm spelled out above.

How long would this brute-force approach take?



Well, for a small graph (such as the one shown here), perhaps it is not too bad…

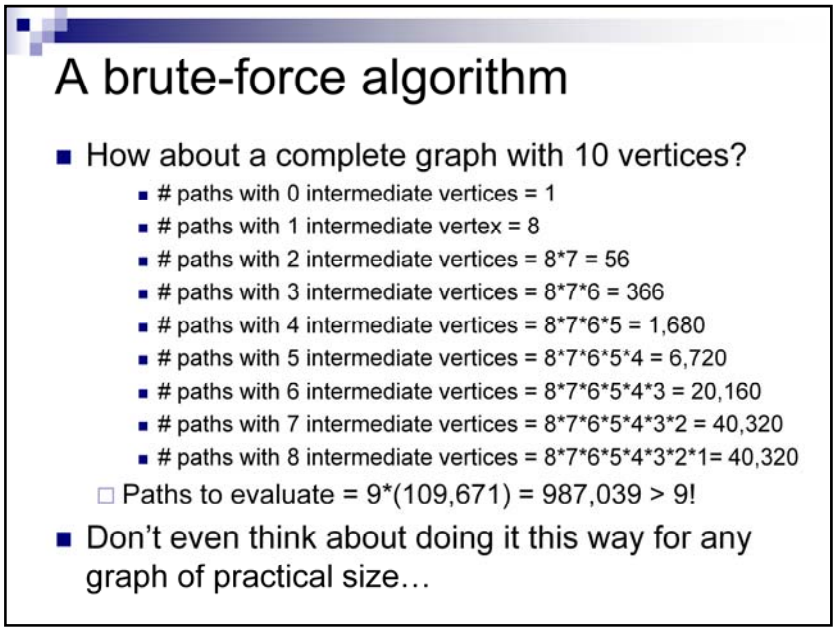
The graph we are using here is a complete graph with 5 vertices. A complete graph has an edge between any pair of vertices. To reduce clutter, the illustration above does not show the costs on the edges. Moreover, we note that such costs may not be symmetric. So, we have two directed edges between any two vertices with possibly different costs on each direction.

First, we consider each possible destination. Starting from A, we have 4 possibilities. For each one of these destinations we need to enumerate the paths from A to that destination. For example, let's enumerate the paths from A to B.

We can:

- Go directly. Only one possibility exists: A→B.
- Go through one intermediate vertex. Three possibilities exist, corresponding to the three possible choices of intermediate vertices: A→C→B, A→D→B, A→E→B.
- Go through two intermediate vertices: Six possibilities exist, corresponding to the six possible choices of two intermediate vertices, namely one of three for the first intermediate vertex, and then one of two for the second intermediate vertex: A→C→D→B, A→C→E→B, A→D→C→B, A→D→E→B, A→E→C→B, A→E→D→B.
- Go through three intermediate vertices: Six possibilities exist, corresponding to the six possible choices of three intermediate vertices, namely one of three for the first intermediate vertex, and then one of two for the second intermediate vertex, and then the last intermediate vertex: A→C→D→E→B, A→C→E→D→B, A→D→C--E→B, A→D→E→C→B, A→E→C→D→B, A→E→D→C→B.

This is a total of 16 paths per destination, and thus a total of 64 paths all together…

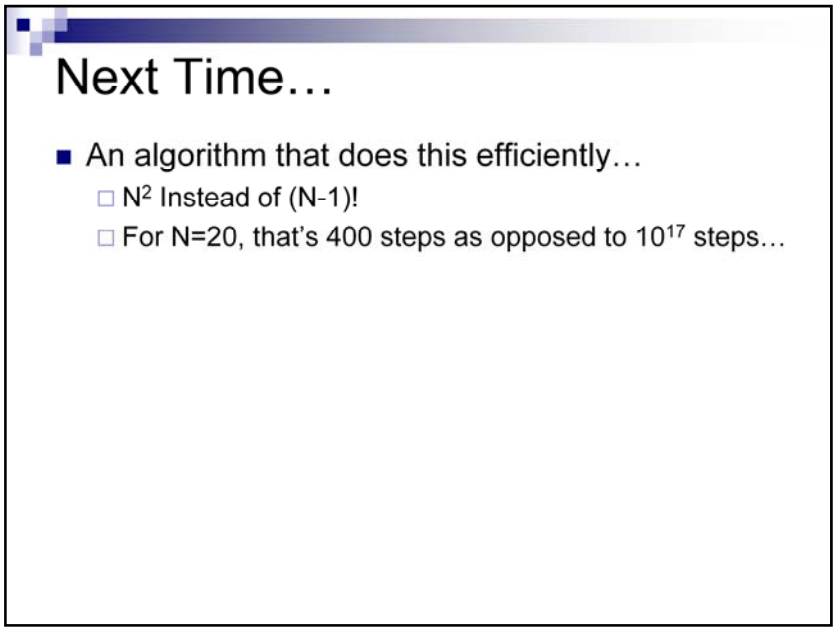


Oh no! Just look at how this explodes, if we just go to a complete graph with 10 vertices (instead of 5).

Basically, the number of paths we have to consider grows with the factorial of (N-1) where N is the number of vertices. And, from our excursion to the functions zoo, we recall that factorials grow very very very fast!

This brute-force (by enumeration) approach is not going to cut it.

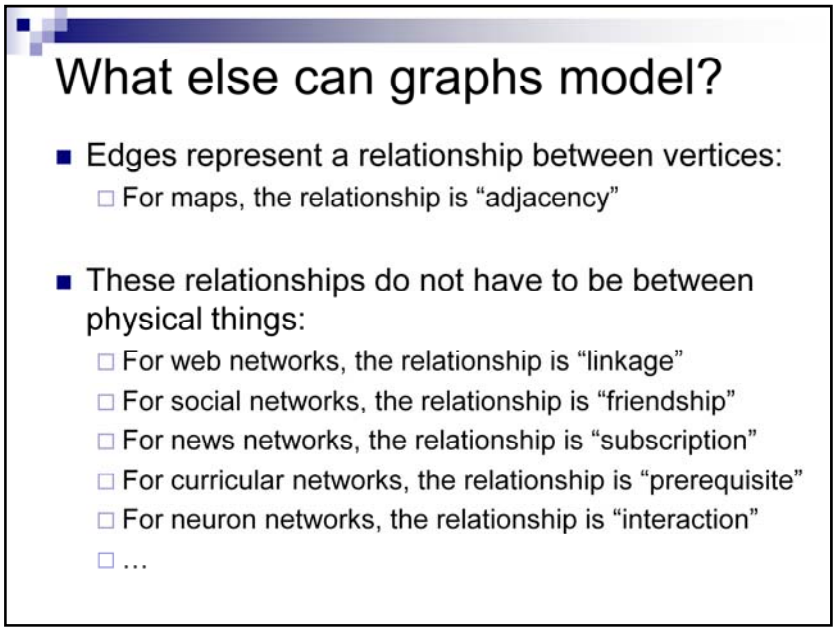
Well, clearly, there must be a way to do this efficiently (otherwise, how could Google Maps, or your tiny GPS figure out shortest paths in the blink of an eye?)



We will discuss an efficient algorithm to just do this next. It is an algorithm that gives us an answer is significantly (and I mean significantly) less time – quadratic in the worst case (versus factorial in the best case using brute force). The difference between the "efficient" (400 steps for N=20) and "inefficient" (100,000,000,000,000,000 steps for N=20) ways to solve this problem should make you really appreciate the importance of algorithmic efficiency.

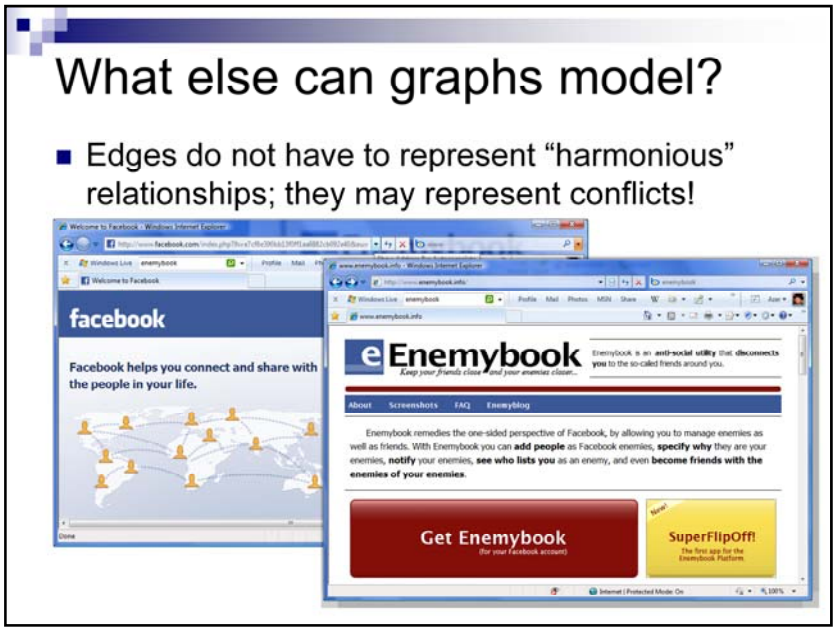
To summarize: Graphs are models of "real world" artifacts. Once we have a model, we can then ask whatever question we want or seek solutions to whatever problem we may define on the graph. One such question/problem is finding the shortest path from a source to a destination (or to all destinations). This is what we focused on in this lecture (and also the next).

That said, there are certainly many other questions/problems that we can ask/solve on graphs.

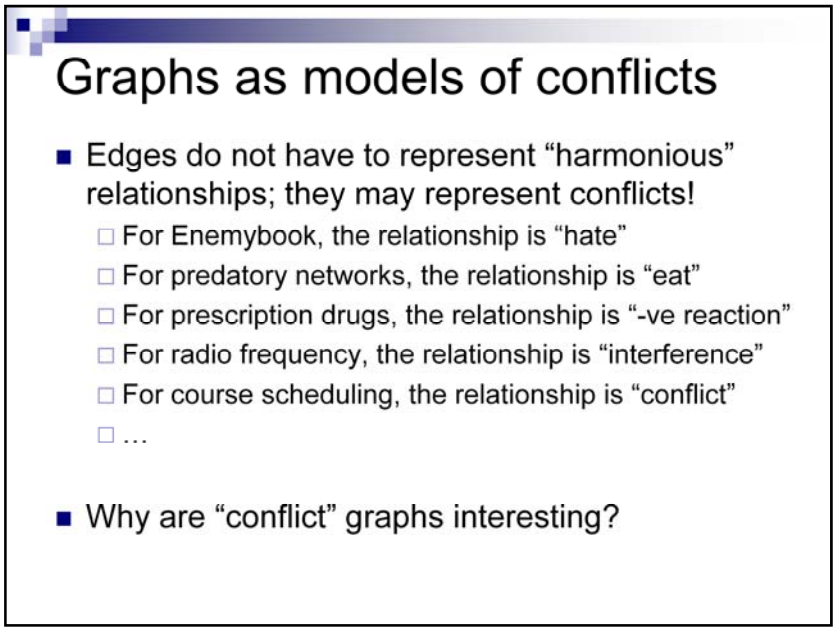


So far, we have seen how a graph is used to model a (subway or internet) map. In such a model, the edges in the graph represented the “adjacency” relationship between the vertices in the graph – two vertices (e.g., subway stops or Internet routers) have an edge between them if they are adjacent to one another. For the subway map, adjacency means that the two stations (modeled by the vertices) are within one stop from one another on a subway line. For the Internet map, adjacency means that the two routers (modeled by the vertices) are directly connected with a physical wire.

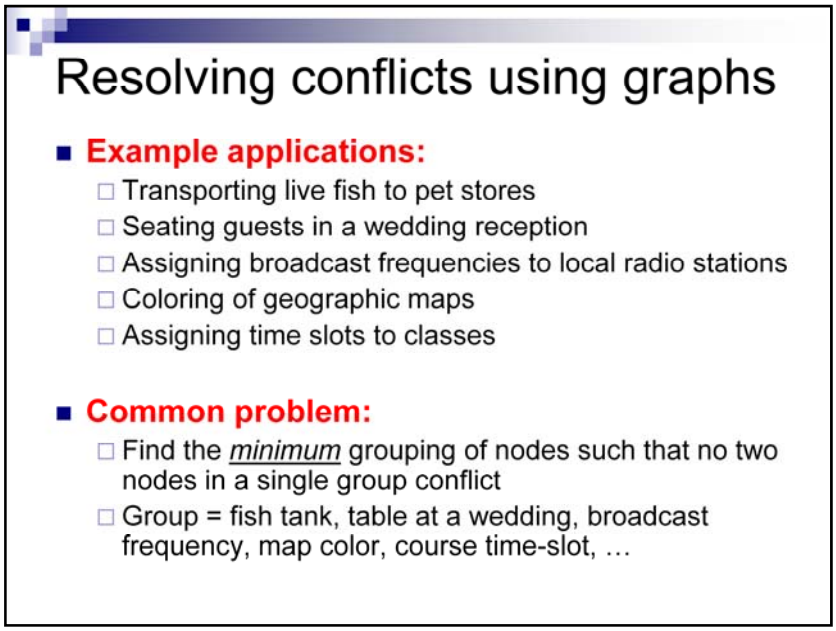
In general, the edges of a graph could represent any relationship between the vertices – e.g., linkage between web pages, friendships between users on Facebook, subscribers to a news feed or followers of a user on Twitter, prerequisite relationships between courses, interactions between neurons in a brain, …



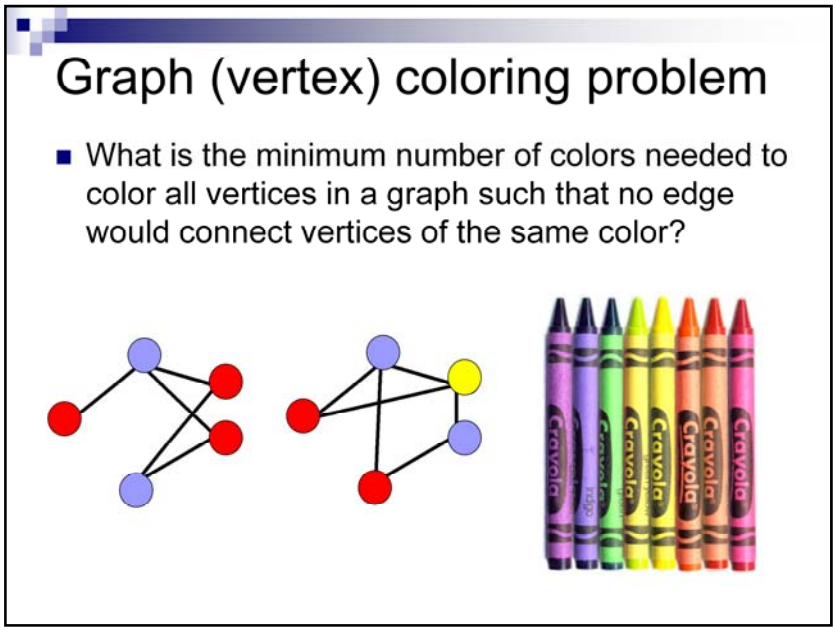
In all the examples we considered so far, the relationship that edges represented was “harmonious” – but that is not necessarily the case. Edges in a graph can also represent conflict relationships!



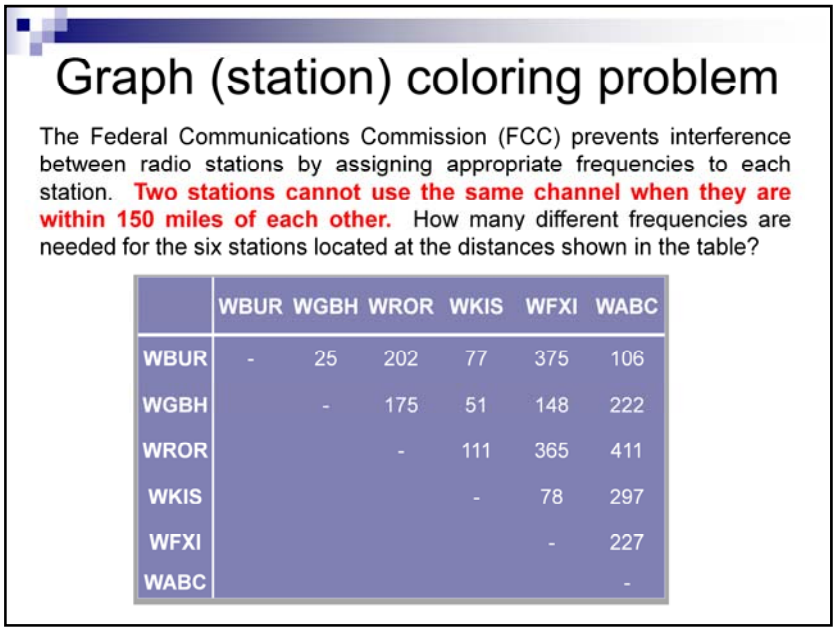
One may use a graph to represent animosity between users (instead of friendship), or a predatory relationship between animals (instead of a symbiotic relationship), …



Many applications require us to reason about such conflicts:
-We would not want to put fish that eat one another in the same tank
-We would not want to seat sworn enemies on the same table at a United Nations function (or one's wedding for that matter!)
-We would not want to assign the same frequencies to radio stations broadcasting in close proximity from one another
-We would not want two states on the US map to have the same color (the map will be confusing)
-We would not want to schedule exams for two classes in the same time slot if a student is taking both of these classes



One such interesting question/problem is “graph vertex coloring”, namely what is the minimum number of colors needed to color vertices of a graph in such a way that any two adjacent vertices (i.e., vertices connected by an edge) are of different color. Finding an answer to this question is very important for many problems.



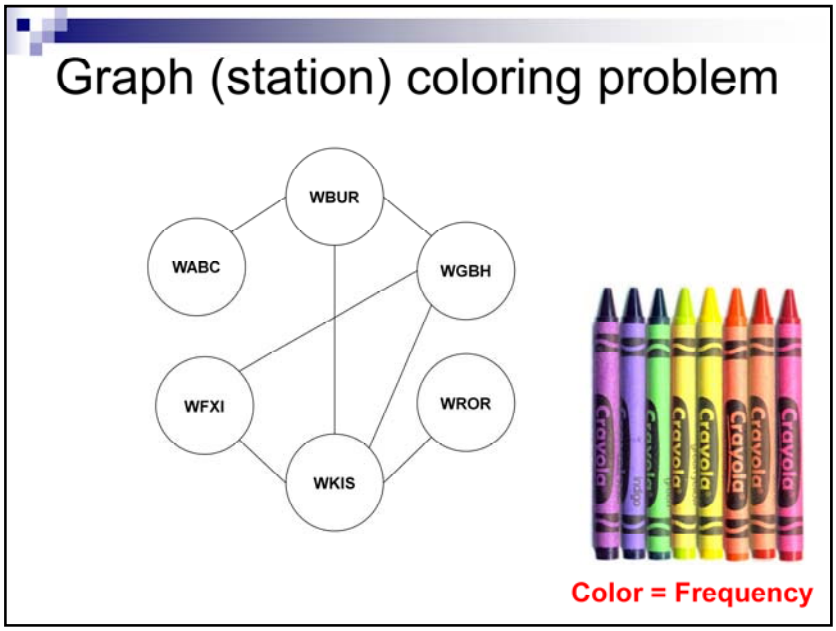
Along the same lines, one can use graph vertex coloring to solve other (quite important) problems, including assigning the minimum number of radio frequencies to various radio transmitters who may interfere with one another – here the radio stations would be the vertices, and if two stations interfere with one another (because of geographical proximity) then we draw an edge between them. Now coloring the graph is akin to assigning frequencies since we would not want to give the same frequency (color) to interfering stations (adjacent vertices).

Graph (station) coloring problem

The Federal Communications Commission (FCC) prevents interference between radio stations by assigning appropriate frequencies to each station. **Two stations cannot use the same channel when they are within 150 miles of each other.** How many different frequencies are needed for the six stations located at the distances shown in the table?

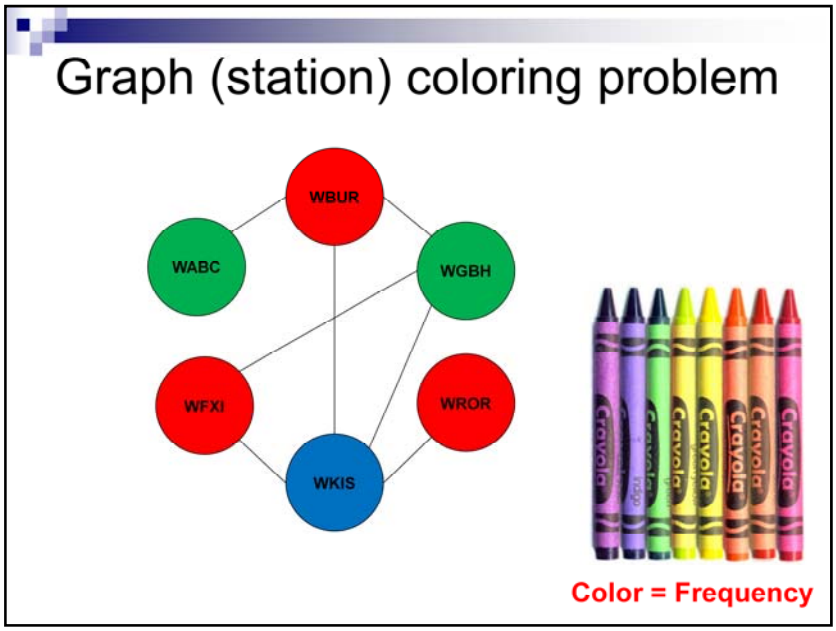
	WBUR	WGBH	WROR	WKIS	WFXI	WABC
WBUR	-	X	202	X	375	X
WGBH		-	175	X	X	222
WROR			-	X	365	411
WKIS				-	X	297
WFXI					-	227
WABC						-

By marking in the table the interfering stations (shown with X) we identify the conflict relationships.

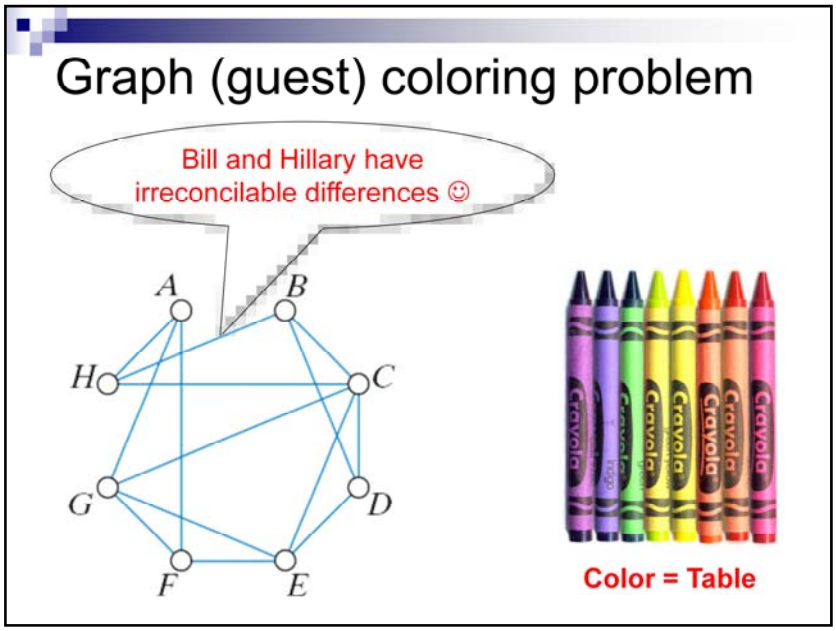


Which leads us to the conflict graph that we can now color to figure out the minimum number of frequencies needed.

How many colors (frequencies) would be needed??



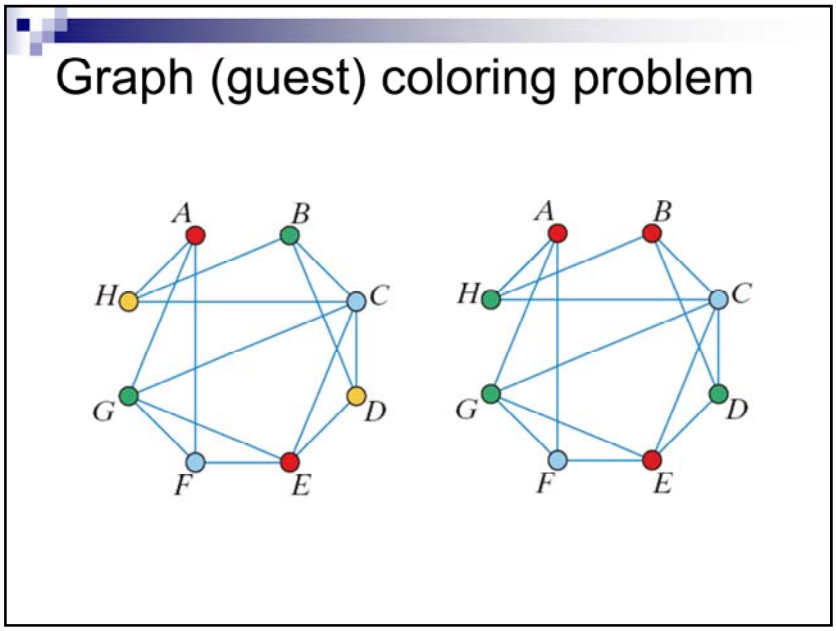
Three colors (radio frequencies) will do it! Can you see why it cannot be done in less?



Yet, another example is the problem of arranging guests attending a wedding around tables. Given a list of "irreconcilable differences" between invitees, you task is to find a seating arrangement that uses the least number of tables while avoiding to have two individuals with "irreconcilable differences" sitting around the same table. This is a graph coloring problem! Namely, we can solve the problem by (1) modeling each individual as a vertex in a graph, (2) representing the fact that two individuals have irreconcilable differences by drawing an edge between the vertices corresponding to these individuals, and (3) minimizing the number of colors (tables) used to color the vertices (seat the guests) such that no two vertices connected with an edge (two guests with irreconcilable differences) have the same color (are sitting on the same table).

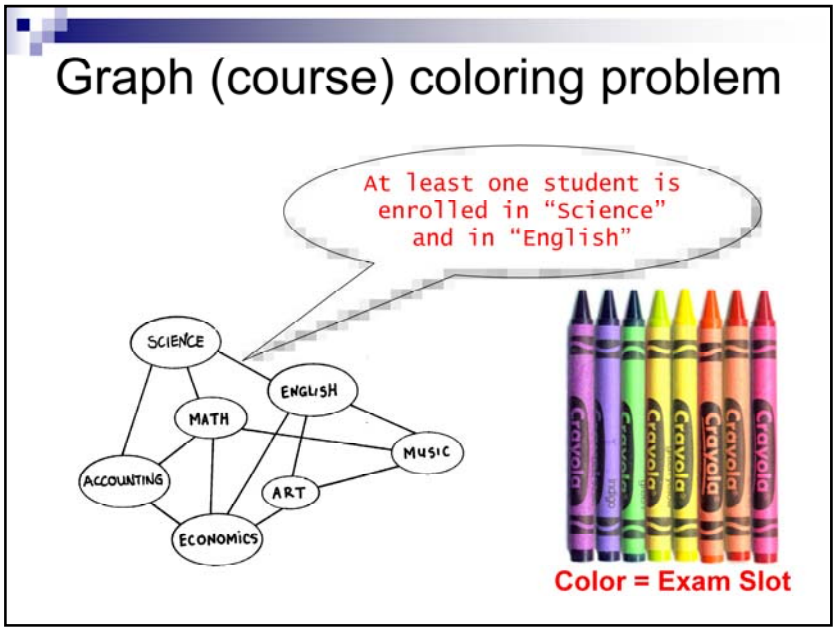
Along the same lines, one can use graph vertex coloring to solve other (quite important) problems, including assigning the minimum number of radio frequencies to various radio transmitters who may interfere with one another – here the radio stations would be the vertices, and if two stations interfere with one another (because of geographical proximity) then we draw an edge between them. Now coloring the graph is akin to assigning frequencies since we would not want to give the same frequency (color) to interfering stations (adjacent vertices).

Yet, another famous application of graph coloring is "map coloring". Here our job is to color adjacent states (vertices) on a planar map (e.g., map of the US) with different colors , but use the minimum number of colors. This fairly classical problem was settled by proving that one need no more than 4 colors for such (planar) graphs.

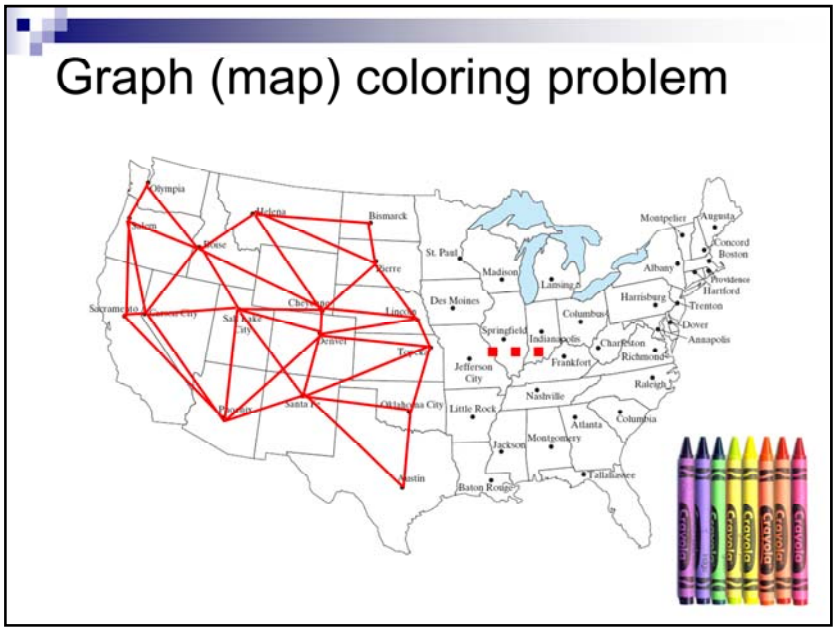


The attempt on the left shows a coloring (assignment of guests to tables) that uses 4 colors (i.e., we need 4 tables). The one on the right shows a coloring that uses only 3 colors. One can show that for this graph, three is the minimum number of colors. Try to prove it!

Accordingly, we would seat A, B, and E on the first (red) table; D, G, and H on the second (green) table; and C, and F on the third (blue) table.



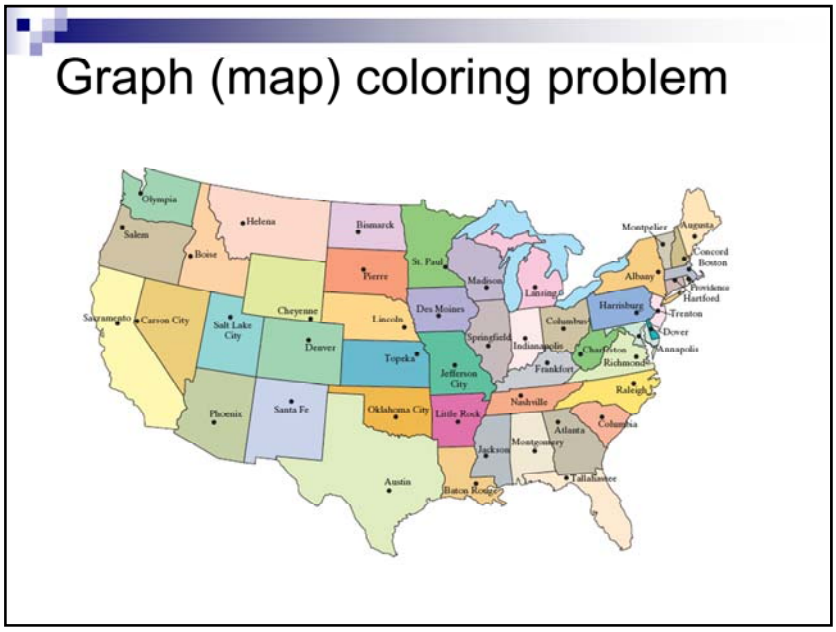
For example, consider the problem of assigning time-slots to the final exams of seven classes (say in Science, Math, Accounting, English, Music, Economics, and Art). Now consider students enrolling in these classes. If a student is enrolled in two classes (say English and Music) then these two classes cannot be assigned the same time slot for the final exam, since the student taking both classes cannot take both finals in the same time. We can model this "conflict" relationship between classes using a graph and by finding the minimum number of colors for that graph, we would identify the minimum number of exam slots needed (in this case, each color would correspond to one exam slot).



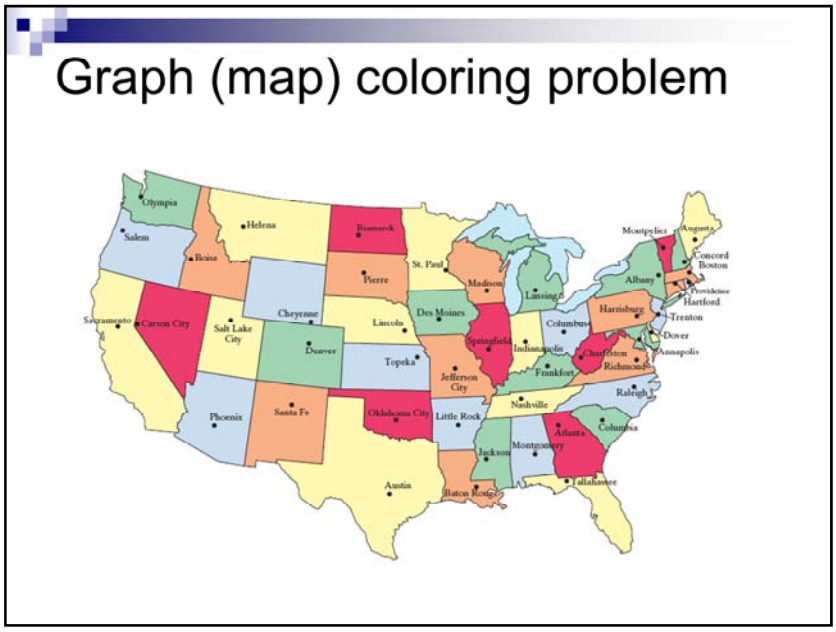
Yet, another famous application of graph coloring is “map coloring”. Here our job is to color adjacent states (vertices) on a planar map (e.g., map of the US) with different colors , but use the minimum number of colors.

Here the states

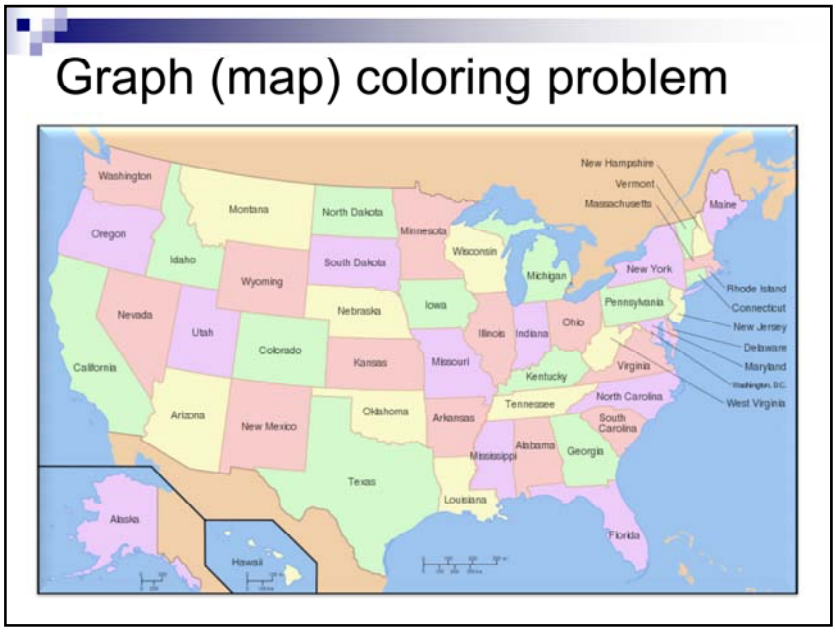
Note: You may ask why is it important to use a small number of colors. Well, not too long ago, printing a map in color was expensive, and the cost would increase significantly as the number of colors used increases.



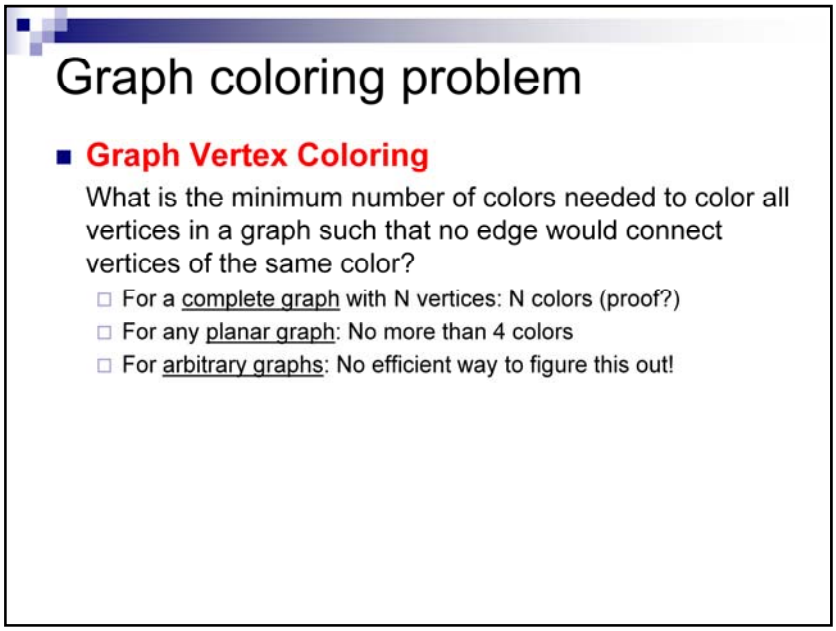
Here is a not-so-good attempt that uses too many colors…



Here is an attempt at a coloring with less colors (using 5 colors)…



Incidentally, this fairly classical problem was settled by proving that one need no more than 4 colors for such (planar) graphs – coloring is shown above.



So, how many colors do we need?

Well, the answer is relatively easy for some special graphs:

1. For a complete graph with N nodes (a graph where each pair of vertices are connected with an edge), we would obviously need N colors. One can prove this very easily by contradiction – Assume that one can color a complete graph with less than N colors such that no two adjacent vertices have the same color. Since the graph has N vertices, then it must be the case that at least two vertices have the same color. But since there is an edge between any pair of vertices, it follows that two adjacent vertices have the same color, which is a contradiction.
2. For a ring with N nodes (a graph where each vertex is connected to exactly one other vertex forming a connected cycle). One can prove that if N is even, then one needs two colors, and if N is odd, one needs 3 colors.
3. For a planar graph (a graph that can be drawn on a plane without having any of the graph's edges intersect), as we mentioned before, it was shown that one needs at most 4 colors…

How about arbitrary graphs? Unfortunately, one can show that finding out the minimum number of colors for arbitrary graphs cannot be done in an efficient way (unless the graph is relatively small).