Encryption

Sit in teams of 4 students from your lab. At least one person on your team needs to set up an account on code.org http://code.org Click "Sign in" to create your own account and join our class using the code KLCXNY

Review the pre-reading questions. During the pre-reading and questions assigned, you were asked to make a prediction about how long it would take you to crack a message encrypted with a simple Caesar shift cipher. Try It! Crack a Caesar Shift https://studio.code.org/s/csp4-2019/stage/7/puzzle/2

You also learned that a random substitution cipher could be broken using frequency analysis. Try It! Crack a Random Substitution Cipher https://studio.code.org/s/csp4-2019/stage/7/puzzle/4

An Encryption key is a specific input that dictates how to apply the method and can also be used to decrypt the message.
1. What is the key for a Caesar Cipher? What is the key for a random substitution cipher?

In encryption you should always assume that your 'enemy' knows the encryption algorithm and has access to the same tools that you do. What makes encryption REALLY strong is making it hard to guess or crack the “key,” even if the “enemy” knows the encryption technique you're using

Encryption with Keys - Go to the Vigenère Cipher Widget https://studio.code.org/s/csp4-2019/stage/8/puzzle/2
Click on buttons and try things out! Solve the mystery of what this tool is doing and how it’s doing it! You should try each of the following - check off the **DONE** column once you’ve tried it.

TRY THIS	**DETAILS**	**DONE**
Encrypt a few different messages using different secret keys	• Enter a text message in the box and secret key • Step through the encoding of each character to see what’s happening • Try a different secret key	
Decrypt a message	• Copy/paste the ciphertext of an encrypted message into the text message area. • Hit the button to “decrypt” • Now step through and see what happens	
Find a “bad” secret key	• Hint: try “A” or “AAAAA” or “GGGG” or any single character, what about other patterns? • What makes a key bad?	
Find a “good” secret key	• Use what you learned about bad keys and do the opposite • What are the characteristics of a good key?	
Try to decrypt without knowing the key (in other words: try to crack it!)	• Have one partner look away, while the other copy/pastes the ciphertext of an encrypted message into the text area, and deletes the secret key from view • Have the partner who looked away come back and try to crack the message	

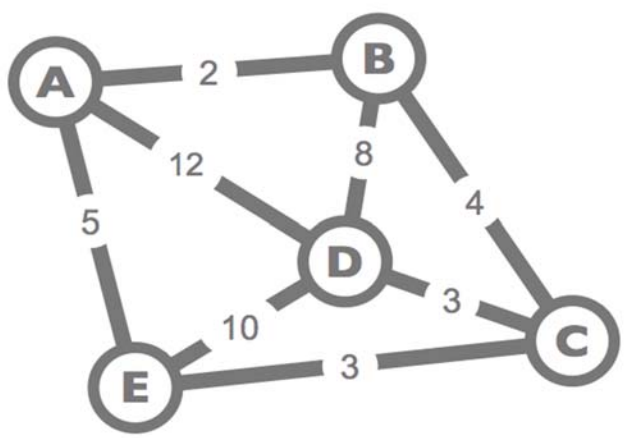
2. How does the Vigenère Cipher work? Why is it harder to crack than a random substitution cipher?

The Internet: Encryption & Public Keys (6:39, start at 2:12)
https://www.youtube.com/watch?v=ZghMPWGXexs&feature=youtu.be

Computationally Hard Problems
Today we're going to look at one of the most famous problems that is thought to be computationally hard. We'll see if we can find an algorithm to solve it, and along the way, we'll get a better sense of what "computationally hard" really means. The Traveling Salesperson Problem (TSP) imagines that a salesperson, starting at his home or office, needs to visit a number of customers at different locations. The goal is for the salesperson to visit each location exactly once and end up back where he started, as efficiently as possible, without retracing any part of the route. We are just using a graph problem as an analogy for encryption.

Given a graph of nodes and edges, where the weights of the edges represent some cost (distance, time, money, etc.), find the set of edges that make the lowest-cost "tour" of all the nodes. A tour is a route that visits each node *exactly once* and returns to the starting node.

This graph shows the direct routes between stores, and the distance of each route. You can assume any node is your home, the starting point.



Here are some possible routes, with their total distances, that visit every node once and return to wherever you started are:

BADECB 2+12+10+3+4 = 31
AEDCBA 5+10+3+4+2 = 24

The shortest route that visits every node once and returns to wherever you started is:
AECDBA 5+3+3+8+2 = 21

For all of the examples on the next page, you're also going to try to solve the traveling salesperson problem to find the shortest route that visits every vertex once and returns to where you started. You are encouraged to mark up these diagrams as you go.

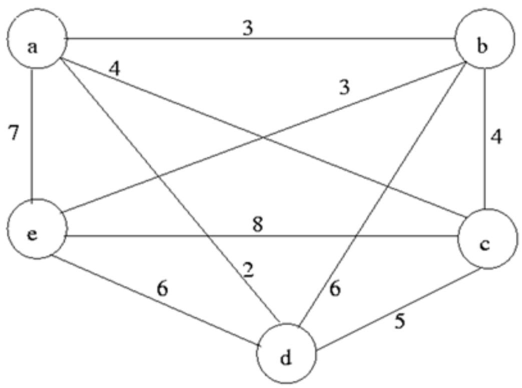
...but you should be thinking...

- **Remember:** In computer science, "solving a problem" doesn't mean finding an answer to an instance of a problem; it means finding an algorithm that might be able to solve *any instance* of that problem.
- As you look at the problems, your brain is working to find a solution. You might think you're just trying "random stuff" but you're not. You are using your human intelligence to help you.
- ***Think about your own thinking process.***
- ***Could you express a way to solve this route-finding problem as an algorithm?***

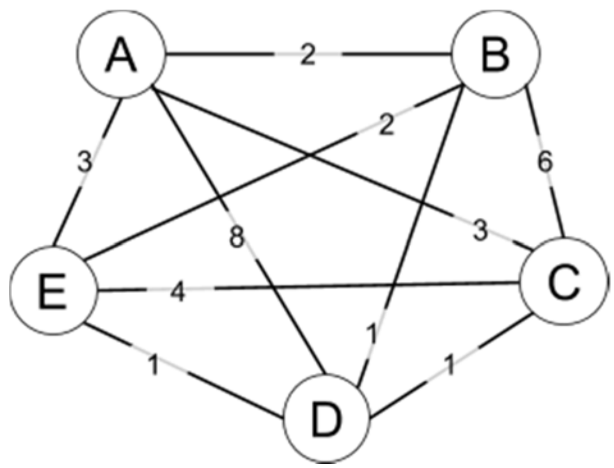
Directions:

- Find the shortest path that visits each node exactly once (i.e. makes a cycle) in each of the graphs below. Highlight the route and make a note of the total distance.
- When you're done, compare with a partner to see if you found the same things.
- In the "Heuristics/Algorithms" area, jot down a few ideas for how an algorithm to find the shortest route might work. Maybe make a few notes about what's potentially tricky, what things you want to be sure to remember.

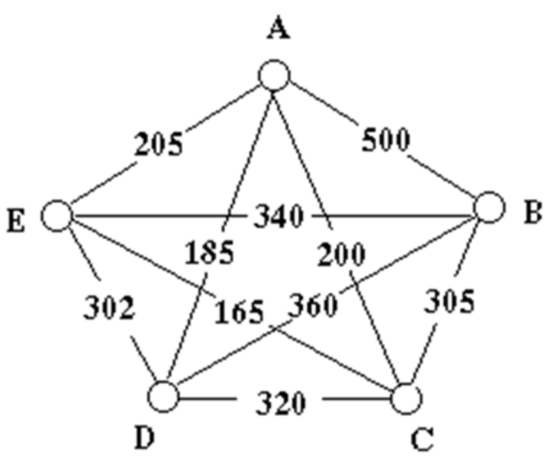
Graph 1:



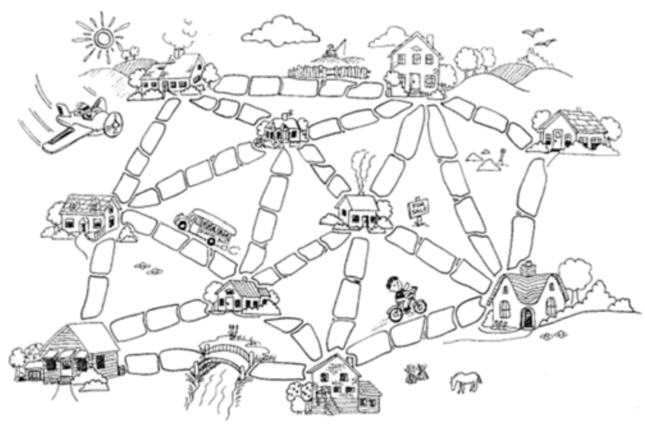
Graph 2:



Graph 3:



Graph 4:



3. Heuristics/Algorithms:

Many people for many years have tried to design an algorithm that works in all cases to find the exact optimum route for the traveling salesperson problem, with no success. So there is currently no algorithm to find the exact answer besides “brute force,” trying all the possibilities. Since we have to calculate the distances of all the tours anyway, we might as well **count how many tours there are in the first place, just so we know how many calculations we have to perform.** To make it easier to count, and to assume the worst case, the largest number of possible tours, let’s assume there is an edge between every 2 vertices (the graph is “fully connected”). Recall a tour is a way to visit each vertex and end up back where you started. And for our purposes it does not matter where we start.

4a. For three vertices, draw how many possible tours are there? How about four vertices?

4b. For five vertices, draw how many possible tours are there?

4c. Based on the previous answers, can you make an estimate for how many possible tours are there for six vertices?

Reasonable or Unreasonable Time - Computers work fast, but they have limits. In computer science, we have an actual mathematical hard line between reasonable and unreasonable runtimes.

"Reasonable" means the number of things the computer has to do is proportional to the size of the input to the problem. Certain graph algorithms are solved them by considering every edge in the graph once. The amount of time it takes is proportional to the number of edges. If the number of edges is n, even if there was an algorithm that had to look at the edge n^2 times, or n^3 times, that’s still reasonable.

“Unreasonable” means the number of things the computer has to do grows as an exponent of the size of the input. So if you discovered that an algorithm made the computer do 2^n things, that’s not reasonable, because it means every time the size of the input (n) gets bigger, the solution gets massively further out of reach. n! is another running time that is considered unreasonable. In real life, “unreasonable” problems would take a modern computer trillions and trillions of years to churn through all the possibilities.

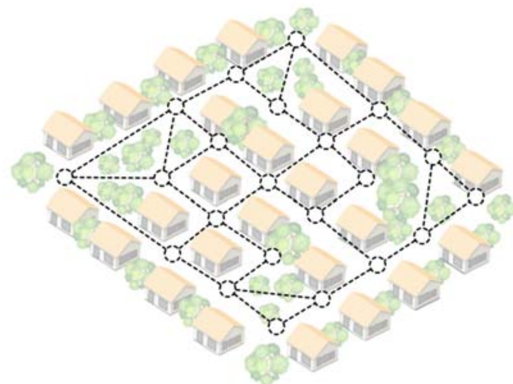
So the brute force solution to TSP is unreasonable -- at least as far as we know. So now we know what “hard” is for a computer.

We are going to continue with using solving graph problems as an analogy for encryption. Getting back to the asymmetric keys and public key encryption idea from the video, we'd like to have some way to construct a problem that is computationally hard to solve (public key) but to which we know the answer (private key). What we need is a one-way function.

<u>One-way Functions</u> - The WiFi Hotspot Problem
A town has decided to provide free wireless access to its citizens. You have been hired to find the best places to install WiFi hotspots throughout the town so that <u>*everyone*</u> has coverage.

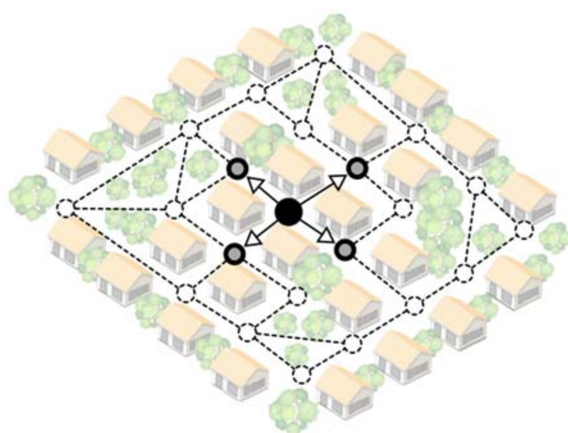
It costs $500,000 to install a hotspot, so the town would like to minimize the cost by installing the lowest number of hotspots necessary to give coverage to the *entire* town.

The town has a lot of buildings, trees and other obstacles that interfere with wireless communication, so placement might get tricky.



A Schematic Diagram of the Town - The town has provided you with the diagram shown to the right. Each **circle** indicates a location in town that *must* be reached by a WiFi signal. The **dotted lines** indicate clear paths where a WiFi signal can travel between circles; these are not blocked or interfered with by trees or buildings.

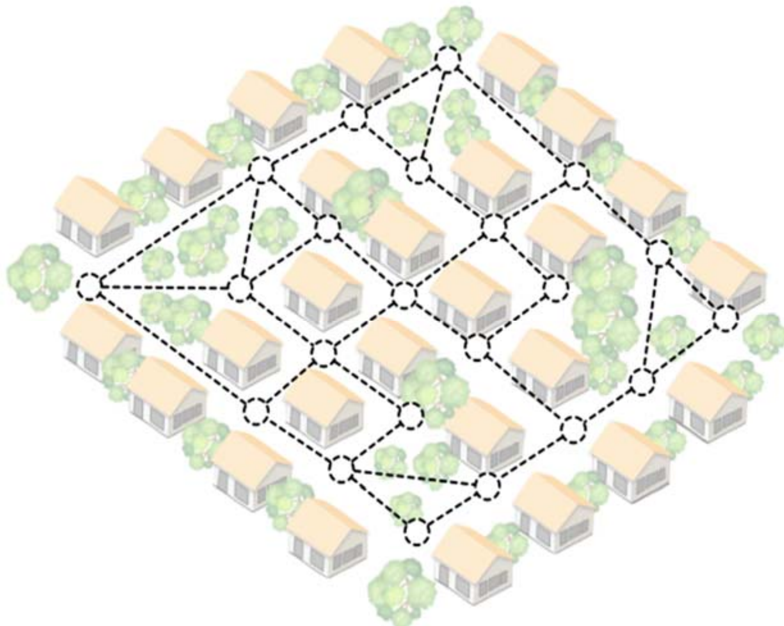
A hotspot can be placed on a circle and it provides coverage to that circle and any circle it's connected to by a dotted line.

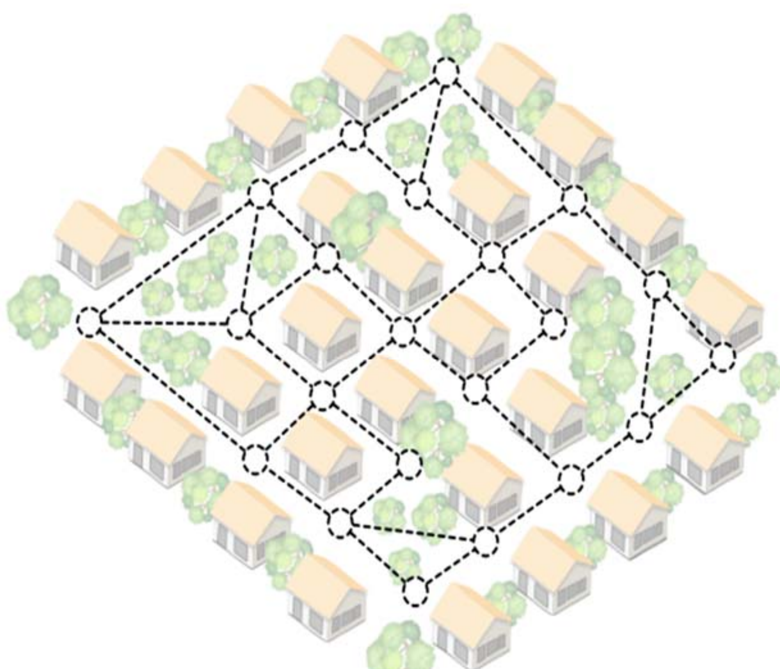


What to Do - Shade in the circles of locations which should have hotspots installed. When you place a hotspot on a circle, it gives "coverage" to any circles it has a clear path to. The diagram to the right shows one circle filled-in with black to indicate a hotspot placement. It also shows the 4 other circles that get WiFi coverage when a hotspot is placed there.

The Goal - Because the town would like to minimize its costs, you must 1) find the *minimum* number of WiFi hotspots that you need to place, and 2) prove that it gives coverage to the whole town by marking on the diagram *where* you need to place them.

The following pages contain a few blank copies of the diagram you can use to work the problem. In the "Heuristics/Algorithms" area, jot down a few ideas for how an algorithm to find the minimal placement of WiFi hotspots might work. Maybe make a few notes about what's potentially tricky, what things you want to be sure to remember.





5. Heuristics/Algorithms: