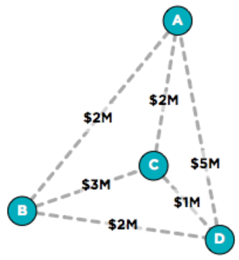
Name(s)____ Period ______ Date _____________

Activity Guide - Algorithms, Graphs, and the MST Problem

In class we asked: *Can you come up with a clear strategy, or process, for finding the minimal-cost connections you need, not only for this network, but any network, even one that's bigger than the example we did in class?*



Why do we care about coming up with a strategy or process for solving this problem, rather than just finding an answer?

Introducing *Algorithms*

Computers are "stupid" in the sense that they don't know the problem you're trying to solve, but they can follow instructions very precisely. So if you can think of a series of steps that will solve the problem, then in theory you can make a computer execute those steps. Often, when we say a computer is "solving a problem," what we really mean is that the computer is following the series of steps written by a person. The person who designed the series of steps is the intelligent one, not the computer. The computer is just fast and precise.

In computer science, we call a precise series of steps that can be executed by a computer to solve a problem an ***algorithm***. We'll learn how to write code for computers to execute algorithms later in the course. For now, we'll get some practice designing algorithms by writing our process with clear English instructions. This is usually how the problem-solving process starts. Real programmers and computer scientists solve problems by first talking about the problem and writing out a process for solving the problem in "pseudocode" or a list of steps in plain English.

Misconception Alert

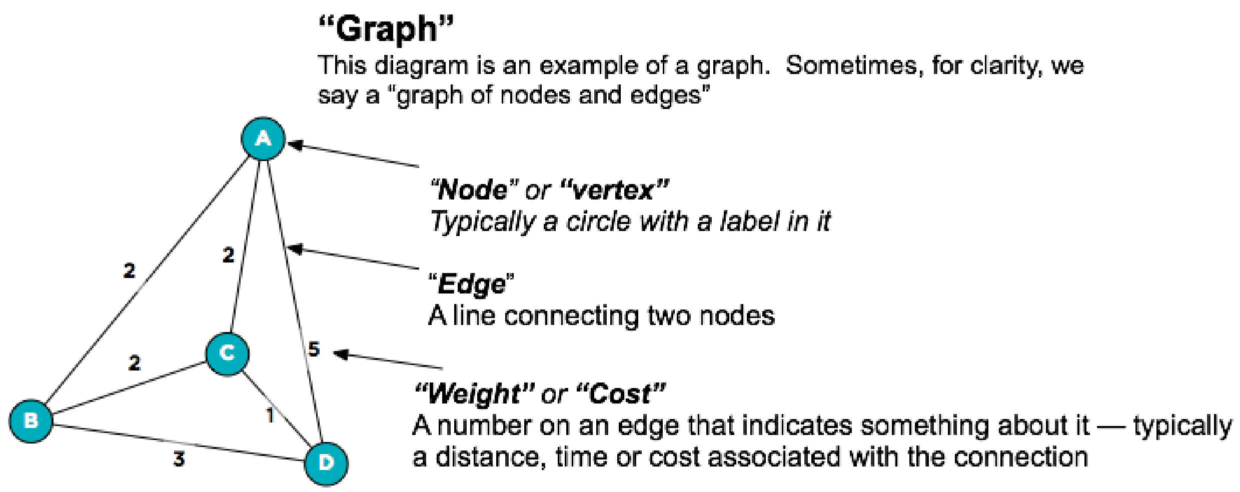
In computer science "solving a problem" usually doesn't mean coming up with the answer to a specific instance of a problem. Solving a problem means *describing the algorithm* - the series of steps - that will solve *every* instance of a particular *type* of problem, regardless of the specifics.

This is (another) one of the things that makes computer science different from math class. Rather than following a procedure to solve a problem, you get to *invent* the procedure. It's the difference between knowing the answer to 2+2, and describing how addition, in general, works. To do that, you must really understand all facets of the problem, so that your procedure can handle many different situations. As long as all the situations have the same fundamental properties in common, your algorithm should be able to solve any of them.

Today you will try to invent an algorithm to solve a problem like the one in the example. Its official name is ***the minimum spanning tree problem.***

Background Knowledge: Graph Problems

Computer networks are often drawn with diagrams similar to the one below. This diagram is actually called a *graph,* though it is probably different from what you typically think of as a graph or chart. Graphs have a number of terms associated with them that you should know, because it makes talking about graph-related problems easier and more clear.

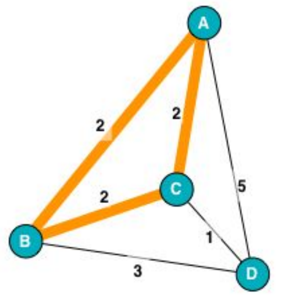
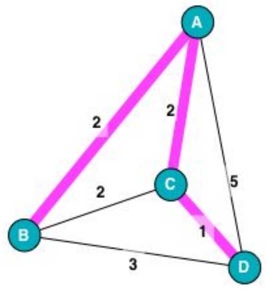


"Graph"
This diagram is an example of a graph. Sometimes, for clarity, we say a "graph of nodes and edges"

"Node"** or **"vertex"
Typically a circle with a label in it

"Edge"
A line connecting two nodes

"Weight"** or **"Cost"
A number on an edge that indicates something about it — typically a distance, time or cost associated with the connection

	
Cycle A set of edges in which, if you started at some node and followed the selected edges, you'd end up back where you started The bold edges shown here form a cycle. Graphs often have many different possible cycles.	**Tree** A set of edges that connects nodes together in a way that *does not* form a cycle If you started at some node and traced the selected edges, you couldn't end up back where you started without retracing your steps. The bold edges shown here are a *tree*. A tree could technically be as small as one edge.

The Minimum Spanning Tree Problem

This activity is about a graph problem known as the "Minimum Spanning Tree" problem. What the heck is a minimum spanning tree? Let's look at what each of these words means:

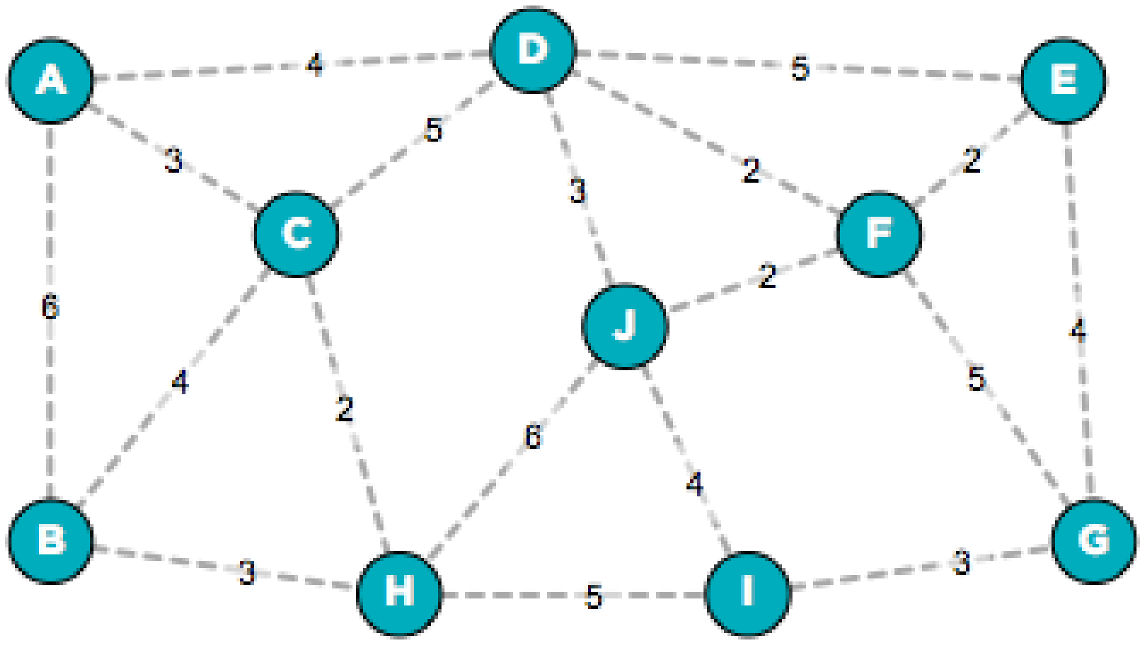
Minimum	the set of edges with the least total cost
Spanning	the set of edges connects *every* node in the graph
Tree	the set of edges *does not* form a cycle

Misconception Alert

You might confuse the minimum spanning tree problem with the "shortest path problem." Finding the minimum spanning tree is not about finding the paths in the graph. It's about finding the set of edges that minimally connects the nodes.

Your Task:
Your **goal** is to be thinking about a general strategy for finding the minimum spanning tree for any graph. To start, just try to solve the problem once. As you solve an instance of the problem you might realize some general properties that would be useful in an algorithm.

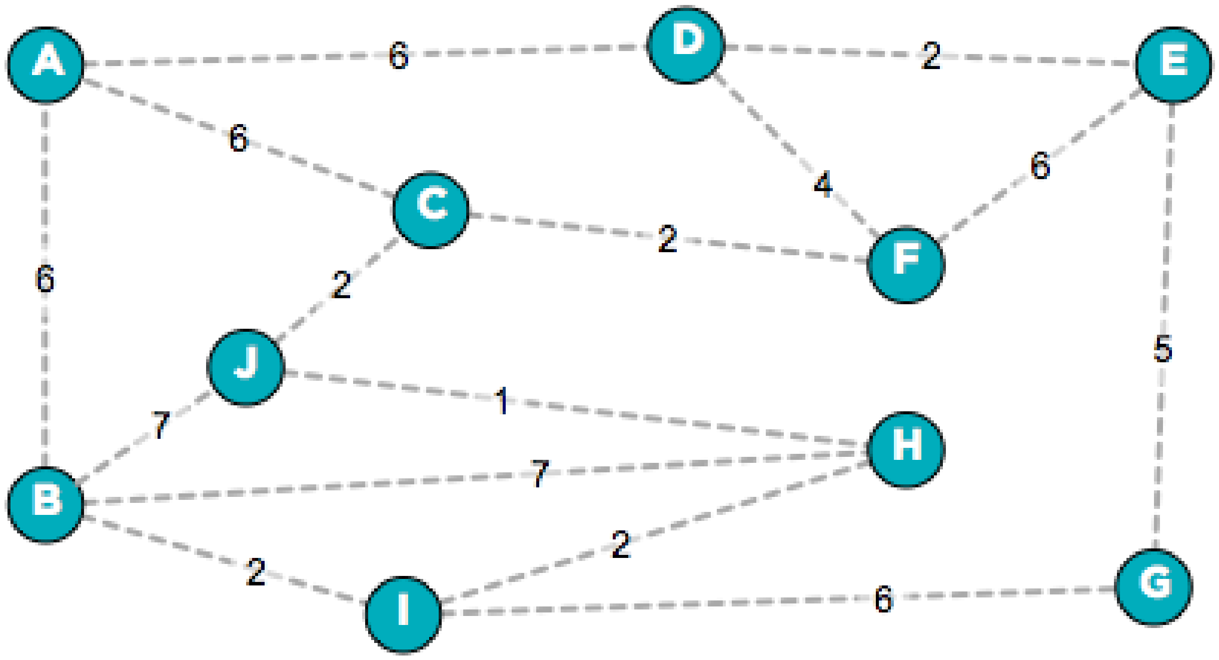
1) Explore the graph shown below to find a minimum spanning tree for it. Once you think you've found it, compare with a partner or neighbor.
    - What's the total cost of all the edges in the minimum spanning tree?



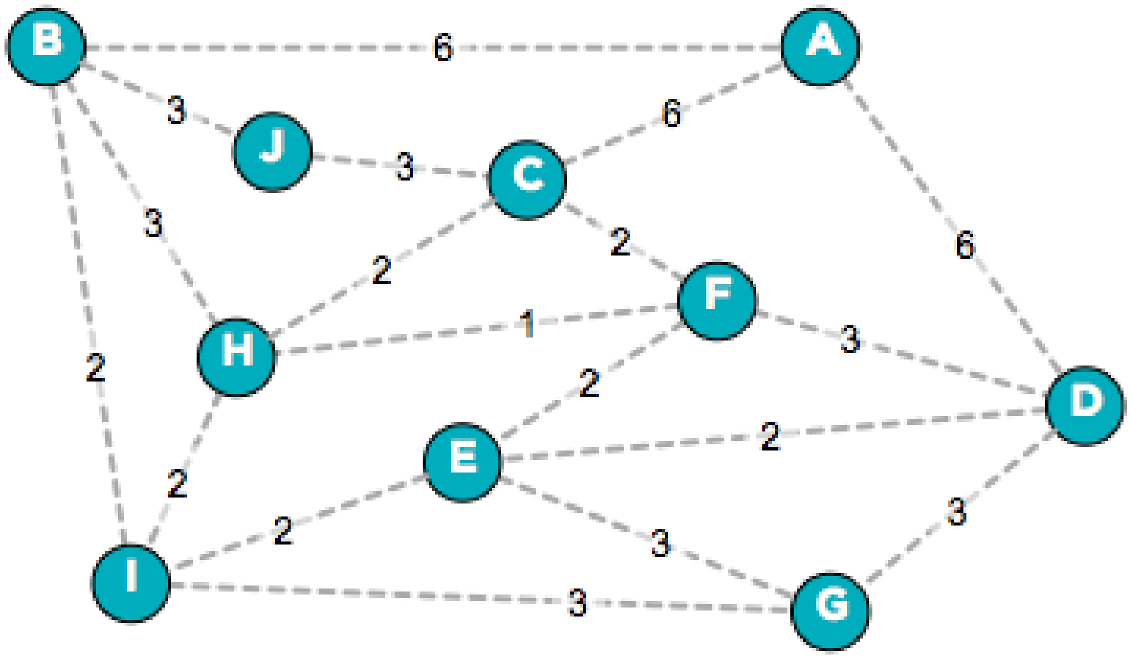
2) Based on what you did, write out a strategy, or a series of steps, that you think would find the minimum spanning tree for *any* graph.

The description of your strategy should be a series of steps in plain English that a person unfamiliar with the problem could follow to arrive at a correct solution. You can test your strategy on the graphs on the following pages to see if it works.

Here are some other graphs you can try your strategy on to see if it works:



Total minimum cost should be: 26



Total minimum Cost should be: 23