Faculty of Mathematics
Waterloo, Ontario N2L 3G1

Centre for Education in
Mathematics and Computing

Grade 6 Math Circles

November 25, 2020

Graph Theory - Solutions

Introduction

Graph theory is a field of mathematics that looks to study objects called graphs. The ideas and understanding gained from studying graphs can be applied to many other problems. Examples of these problems include matching organ donors to patients or finding the best routes from point A to B (like a GPS does). Today we will look at a couple famous algorithms that help us to find the best or shortest networks and routes.

First, we need to understand some basic properties of graphs:

Definitions

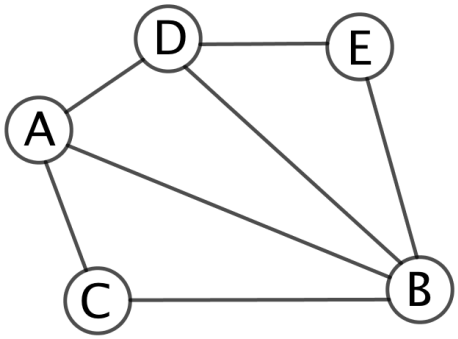
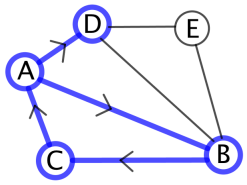


Figure 1: Graph used in examples below.

- A **vertex (plural: vertices)** is a point on a graph. Typically, it is represented by a small circle. It may also have a label.
  Example: The circles labelled A, B, C, D and E in Figure 1 are vertices.

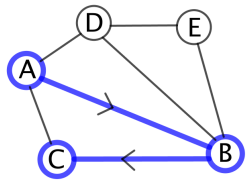
- An **edge** is a line that connects two vertices. The **endpoints** of an edge are the vertices that it connects.
  Example: The segment connecting vertex A and vertex B is an edge. We typically write an edge, e, as $e = \{A, B\}$.

- A **graph** is a collection of vertices with edges between some pairs of them.
  Example: Figure 1 shows a graph.



- A **walk** is a sequence of vertices and edges that lead from one vertex to another. A **path** is a walk with no vertex repeated, except possibly the starting and ending vertex.
  Example: $A, \{A, B\}, B, \{B, C\}, C, \{C, A\}, A, \{A, D\}, D$ is a walk in the example graph.
  $A, \{A, B\}, B, \{B, C\}, C$ is a path.

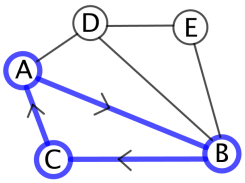
Walk



- A **cycle** is a path that has the same starting and ending vertex. The length of a cycle is the number of distinct vertices in the path.
  Example: $A, \{A, B\}, B, \{B, C\}, C, \{C, A\}, A$ represents a cycle of length 3 in the graph to the right.

Path

- The **neighbours** of a vertex are all vertices that are connected to the vertex by an edge. If a vertex is the neighbour of another vertex, the two vertices are considered **adjacent**.
  Example: The neighbours of vertex C are A and B.



- The **degree** of a vertex is the number of neighbours that it has.
  Example: Vertex C has degree 2.

Cycle

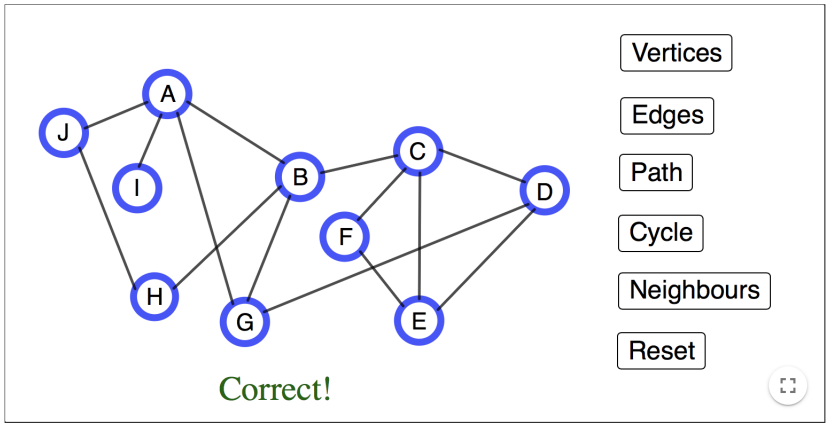
This week we will be looking only at **simple graphs**. A **simple graph** is a graph with three specific conditions.

1. Edges have no direction (we can travel in both directions on them when creating a walk or path).
2. There is at most one edge between two vertices.
3. There are no loops. In other words, no edges that connect back to the same vertex.

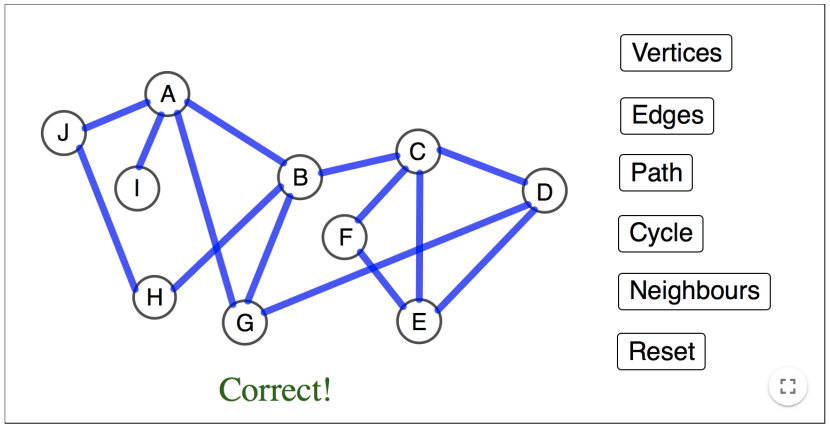
Exercise 1. Throughout this document there are online exercises to help practice what we have learned. Click on the link to try the activity: *https://www.geogebra.org/m/qeajsqya*

Solutions:

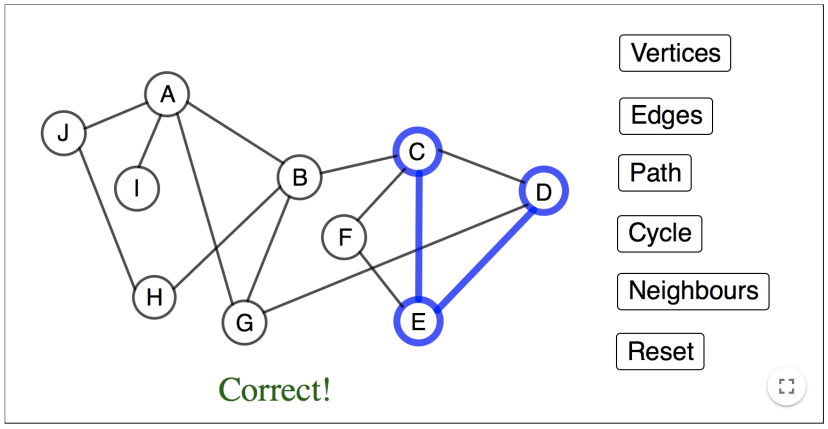
Vertices:



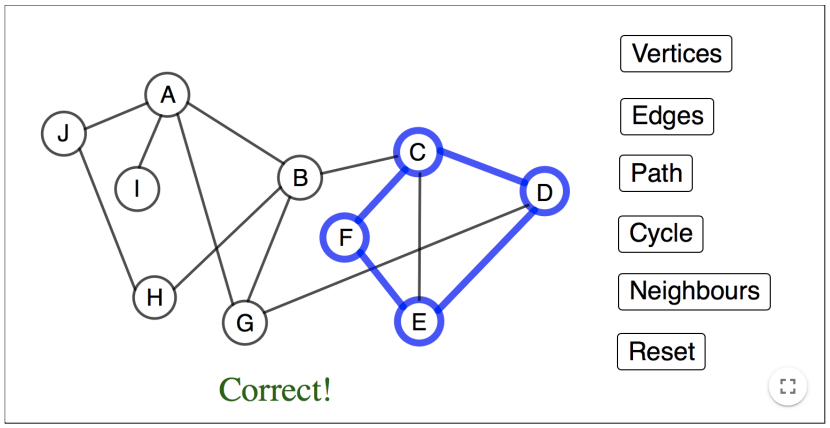
Edges:



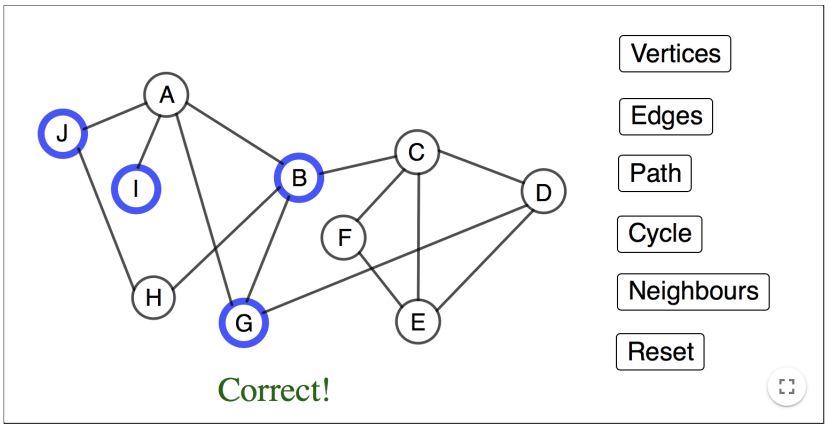
Path:



Cycle:



Neighbours:



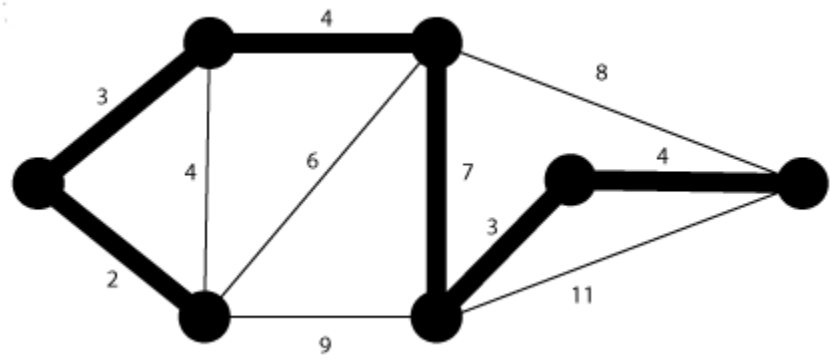
Prim's Algorithm

Prim's Algorithm was first developed in 1930 by Vojtěch Jarník, but was later rediscovered and published by Robert Prim in 1957, hence the name. Prim's algorithm provides an efficent solution to one of the neat problems in graph theory: determining the minimum spanning tree for a graph. To understand the problem, we need a couple more definitions:

- A **tree** is a graph with no cycles.
- A **weighted graph** is a graph that has a weight or value assigned to each edge.
- A **subgraph** is a graph whose edges and vertices are all part of a possibly larger graph.
- A **spanning tree** is a subgraph of a graph that is a tree. The tree must include all vertices from the initial graph.

Thus, a **minimum spanning tree (MST)** is a spanning tree that has the lowest total weight when all the edges in the tree are added together.

Example of a Minimum Spanning Tree:



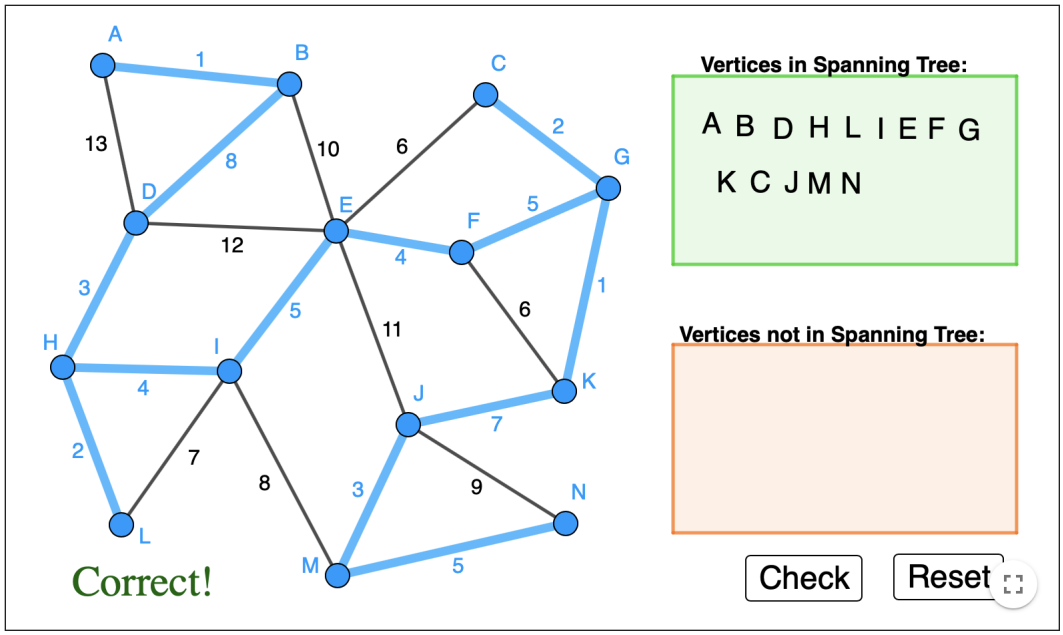
To find a MST of any graph, we can follow these steps, called Prim's Algorithm:

1. Pick a vertex in the graph, it is now in your spanning tree.
2. Consider all the edges that are connected to exactly one vertex in your current spanning tree. Pick the edge with the smallest weight and add it, along with the other vertex that it connects to, to your spanning tree. (If multiple edges have the same smallest weight, pick any of those edges. Note that this means that multiple trees could be MSTs, so long as they have the same smallest weight.)
3. Repeat step 2 until all vertices are in the spanning tree. You now have a minimum spanning tree!

Watch this video: *https://youtu.be/L_Z94bhPCGw* for an example of finding a MST.

Exercise 2. Try to find the MST of the graph here:
https://www.geogebra.org/m/uc3sqrgb.

Solution:



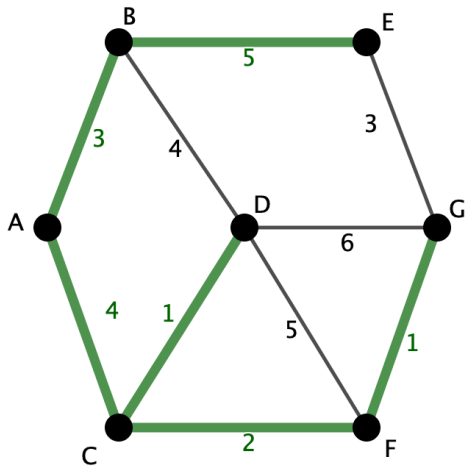
Dijkstra's Algorithm

The minimum spanning tree is great if we want to find, for example, the least costly way to create traintracks such that we can still visit each location. But what if I want to find the quickest way to get from point A to point B? That's where Dijkstra's algorithm comes in. It can help us to find the cheapest path between two vertices in a given graph.

Before we look at the algorithm, we need one more definition:

Definition 1. *Shortest Path Tree*
A shortest path tree is a tree that contains the shortest paths from one "start" vertex to any other vertex in a given weighted tree.

Example:



In the given tree, A is our start vertex. Notice that, if we don't want to repeat vertices or edges, there is only one path from A to each other vertex in the tree. That unique path is the shortest path from A to that vertex.

Dijkstra's Algorithm:

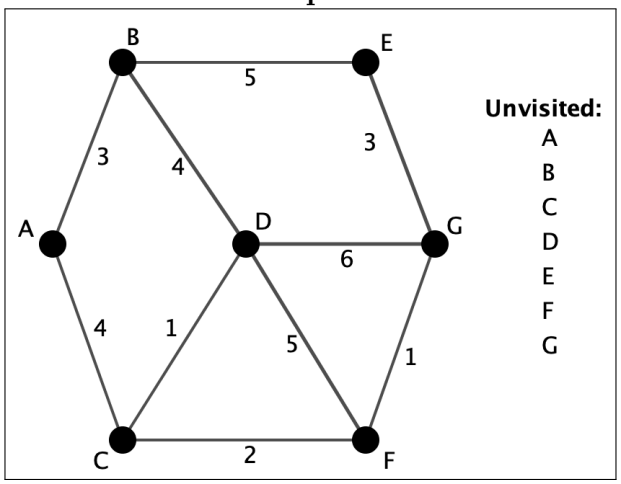
1. Make a list called "unvisited". Put all the vertices in your given graph on that list.
2. Label each vertex with a start cost. The vertex that you are starting from should be labelled 0 and the others infinity (∞).
3. Choose the vertex that has the smallest labelled cost that is in the unvisited list (if this is the first time through, it should be your starting vertex). If there are multiple vertices with the smallest labelled cost, chose any of those vertices. Note that like with Prim's Algorithm and MSTs, this can lead to multiple shortest path trees. Let's call that vertex our "current" vertex.
4. Remove our current vertex from your unvisited list.
5. Create a list called "check cost" that contains the neighbours of our current vertex that are still in the unvisited list.

6. For each vertex in your check cost list, add the labelled cost of our current vertex to the cost of using the edge between our current vertex and the neighbour. If that cost is less than the labelled cost on the neighbour, relabel the neighbour's cost as that added value.

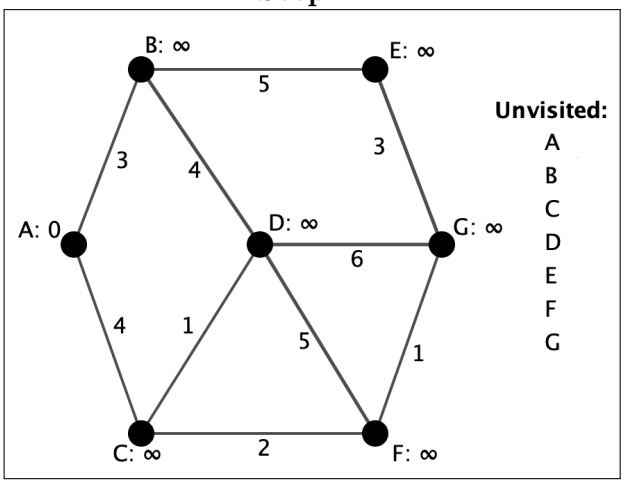
7. Repeat steps 3-6 until there are no vertices left in the unvisited list. You have now created a shortest path tree! You can now find the shortest path from your starting vertex to any other vertex in the graph.

Dijkstra's Algorithm can be tricky to understand without some visuals. Don't worry if you don't understand how it works quite yet! Take a look through the following images that show what each step might look like. Then, watch this video: *https://youtu.be/ozREmyMVPog* for a full example of creating a shortest path tree for a graph.

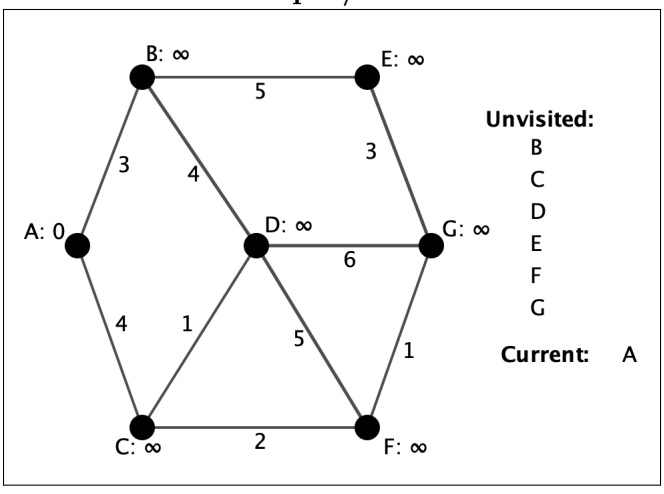
Step 1:



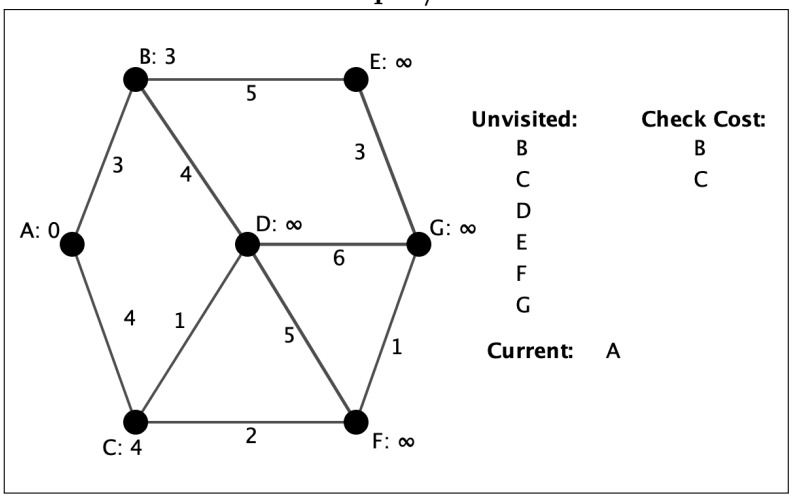
Step 2:



Step 3/4:

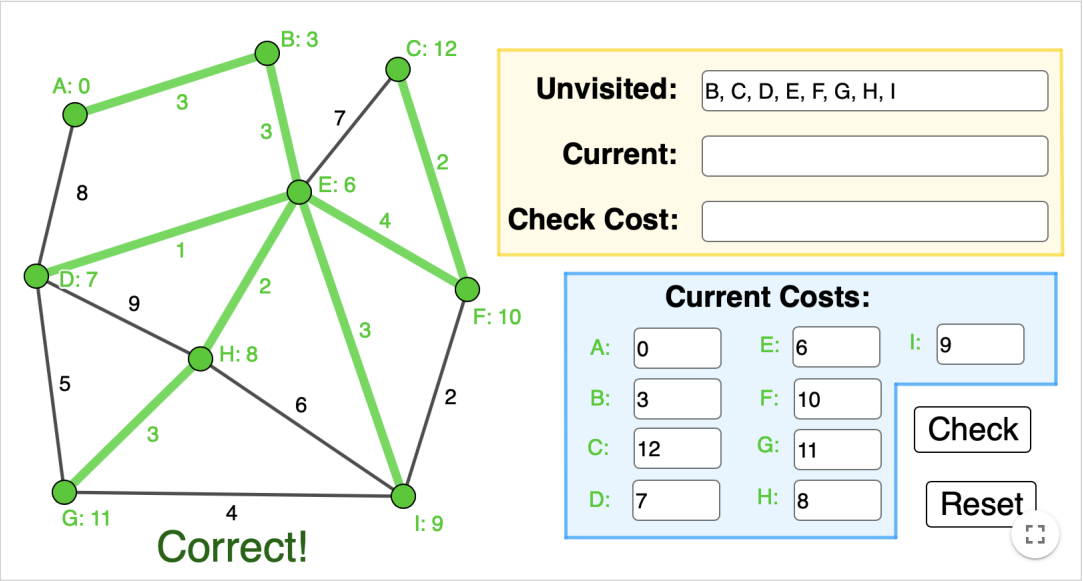


Step 5/6:



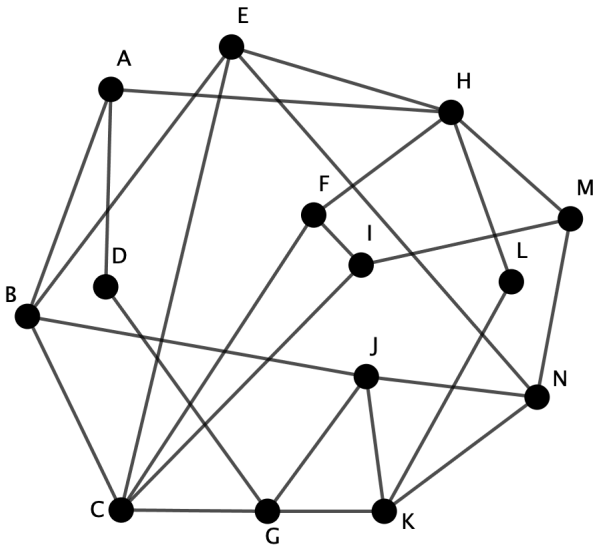
Exercise 3. Try to find the shortest path tree of the graph here:
https://www.geogebra.org/m/fygzyuqf.

Solution:


A: 0
B: 3
C: 12
D: 7
E: 6
F: 10
G: 11
H: 8
I: 9
3
3
7
2
8
4
1
2
9
3
5
6
3
2
4
Correct!
Unvisited:
B, C, D, E, F, G, H, I
Current:
Check Cost:
Current Costs:
A: 0
B: 3
C: 12
D: 7
E: 6
F: 10
G: 11
H: 8
I: 9
Check
Reset

Problem Set

1. Highlight the following walks in the graph below. For each, state whether it is a path. How do you know?



(a) A - H - L - K - J

(b) B - E - N - K - G - C -B - J

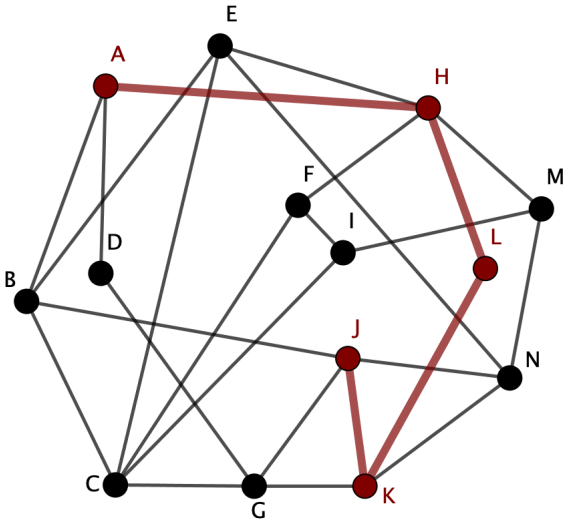
(c) M - I - F - C - E - N - J - B - A

(d) E - C - I - F - C - G - K

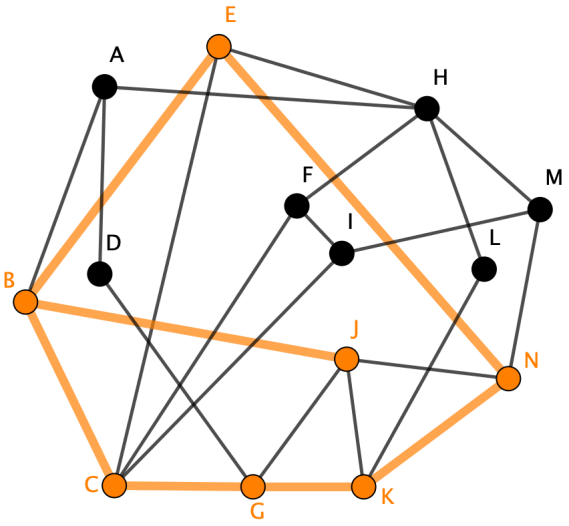
Solution:

a) and c) are paths as they don't repeat vertices. b) and d) are walks, but not paths as they have repeated vertices.

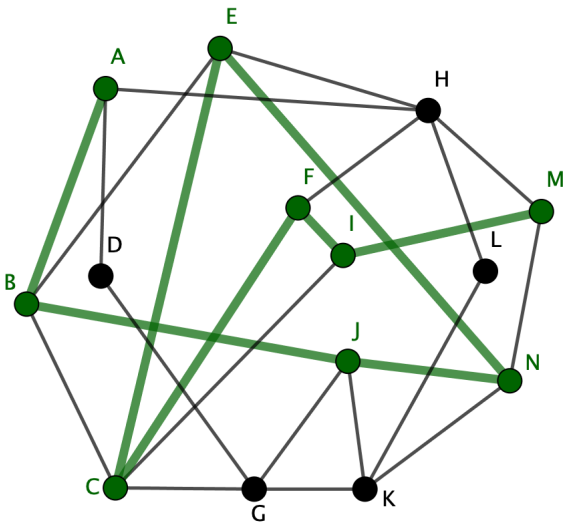
a)



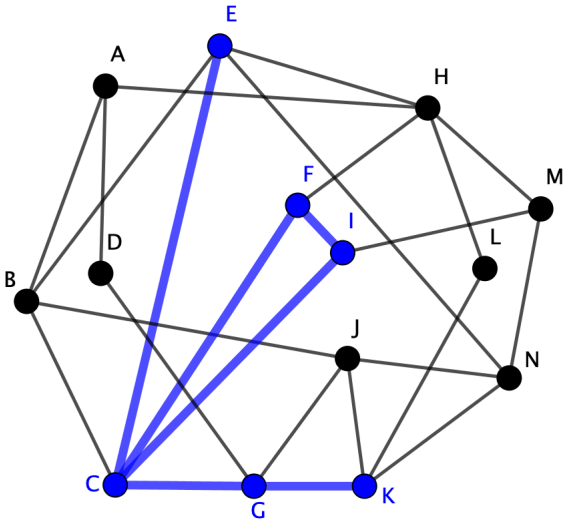
b)



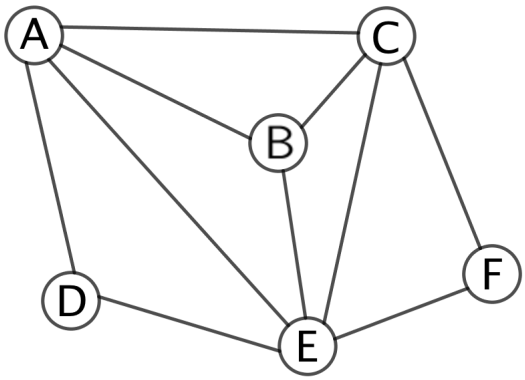
c)



d)



2. For the graph on the right:



   (a) Find all the cycles of length 3.
   Hint: There are 6 different cycles of length 3.

   (b) Find all the neighbours of vertex B.

   (c) Find a path from vertex A to vertex F.

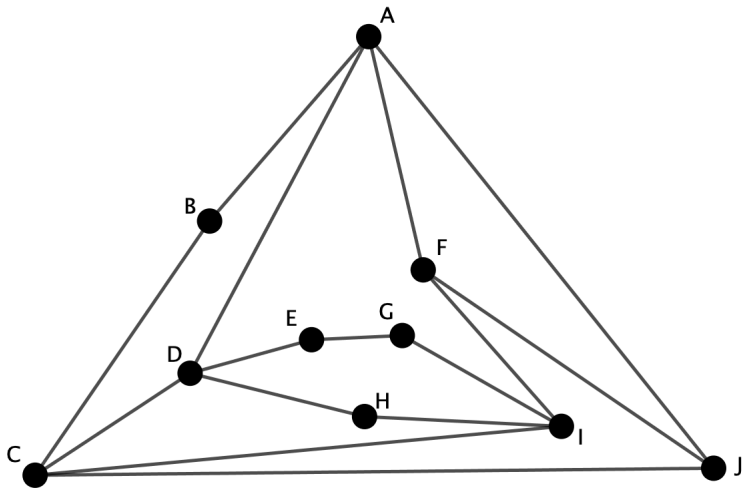
Solutions:

(a) The cycles of length 3 are: A - B - E - A, A - C - B - A, A - E - D - A, B - C - E - B, C - E - F - C and A - C - E - A. Note that you may have listed the cycles differently. This is ok! Something like B - E - A - B is the same cycle as A - B - E - A as it contains the same vertices.

(b) The neighbours of vertex B are A, C and E.

(c) A path from vertex A to vertex F could be A - C - F. Note that to be a path, we cannot repeat vertices.

3. For the graph below, find 5 different walks from A to I. How many of those walks are paths? Find 5 paths from A to I.

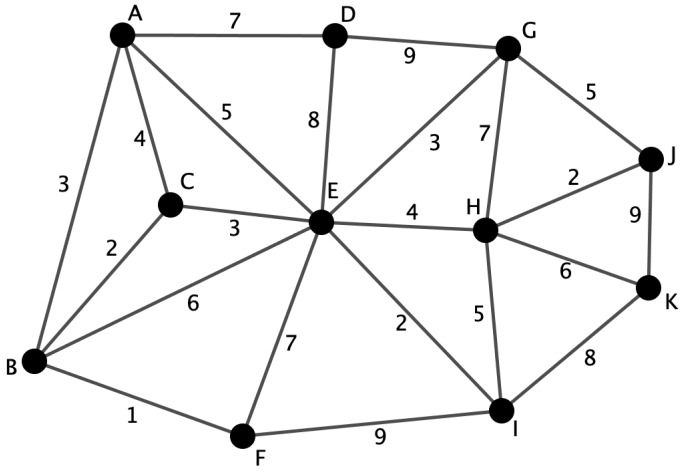


Solution:

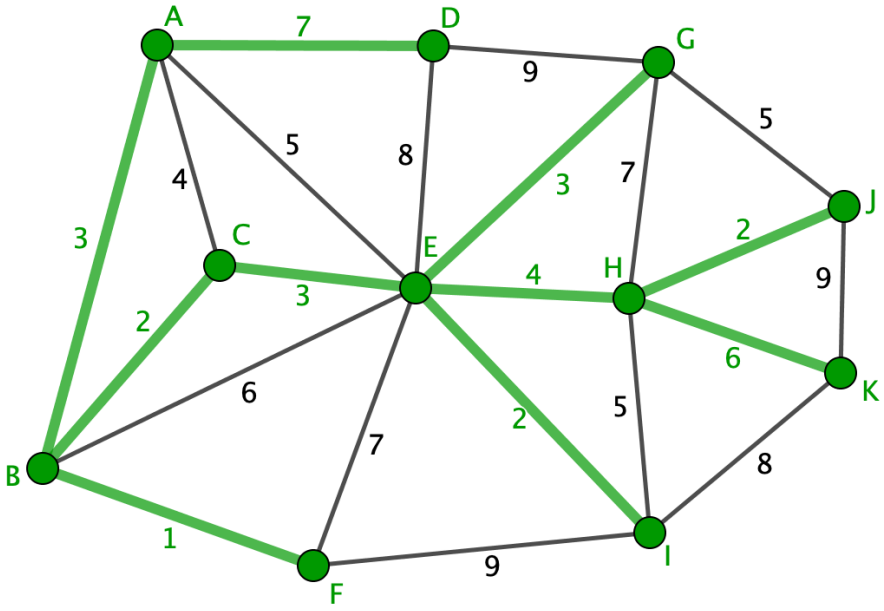
Here are 5 walks from A to I: A - F - I, A - D - H - I - F - I, A - D - E - G - I, A - J - A - F - I, A - B - C - D - H - I. The first, third and fifth walk are paths as they don't repeat vertices. 5 other paths from A to I are: A - J - F - I, A - J - C - I, A - B - C - D - E - G - I, A - J - C - D - H - I, A - F - J - C - I.

4. Use Prim's Algorithm to find the minimum spanning tree for the below graphs.
Hint: If there is a tie for the smallest edge, you can choose any of the edges with the smallest weight.

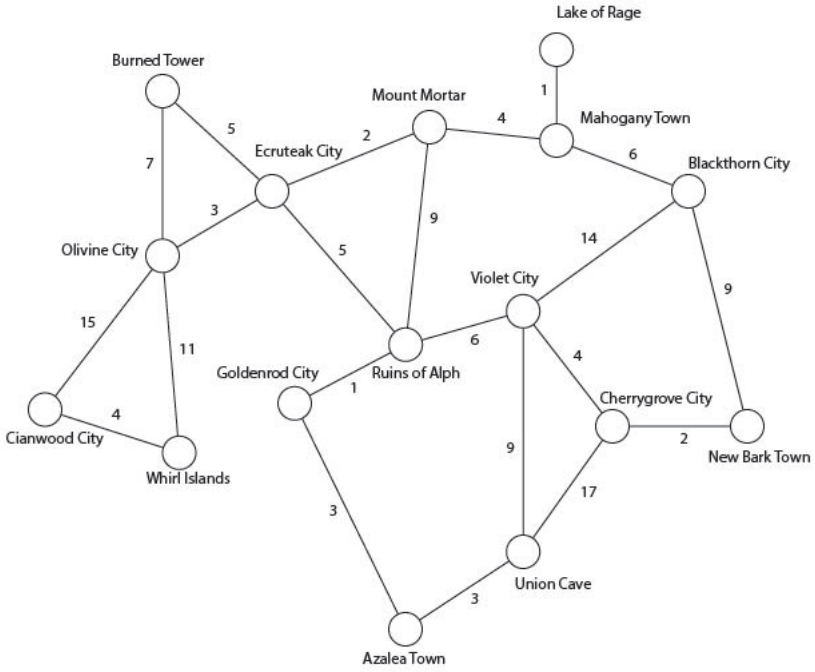
Graph 1:



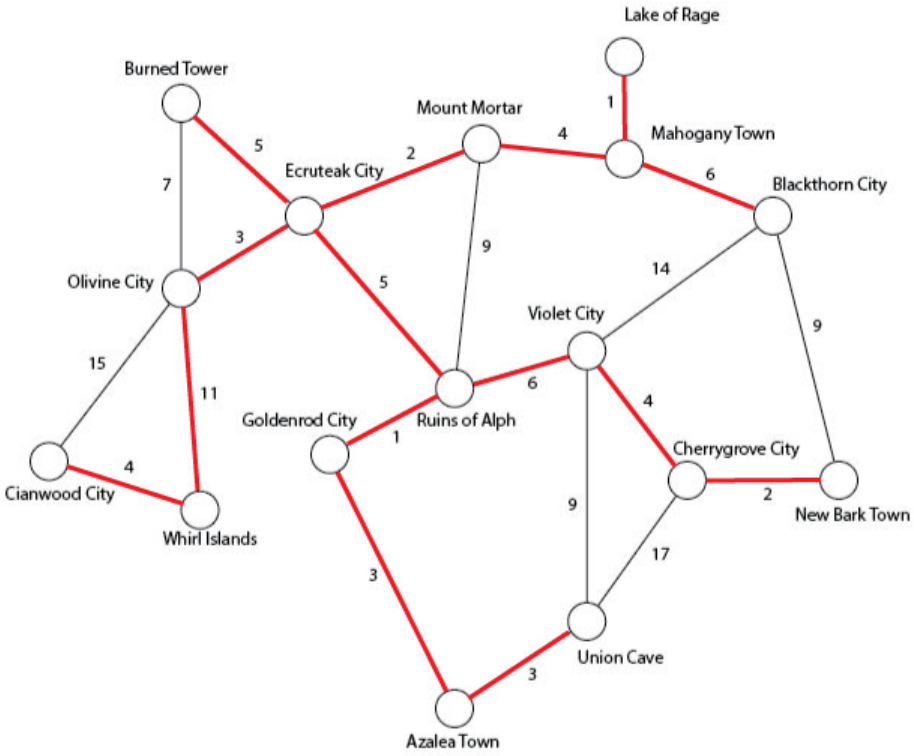
Solution:



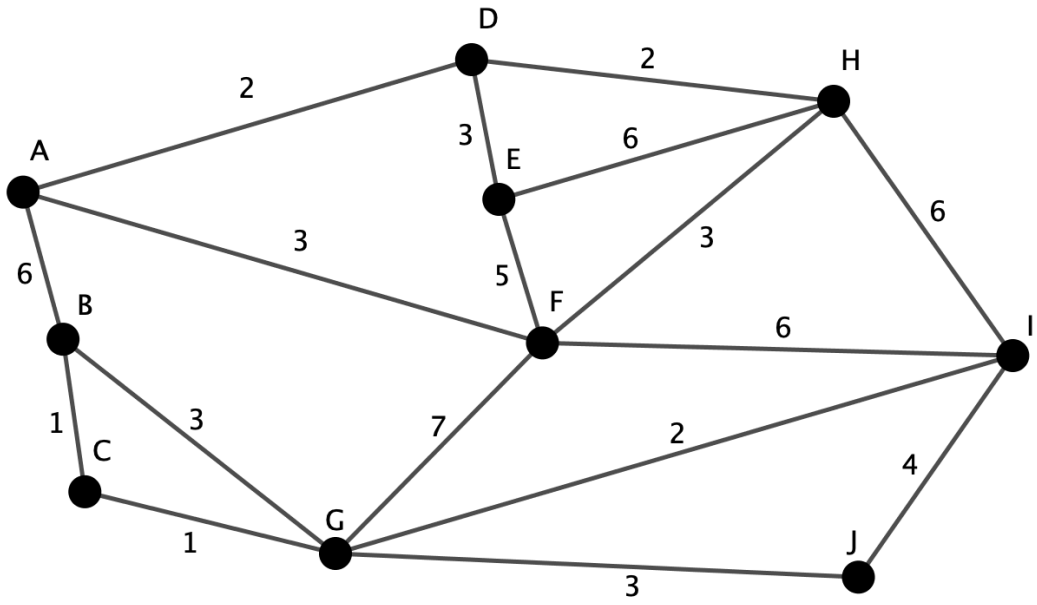
Graph 2:



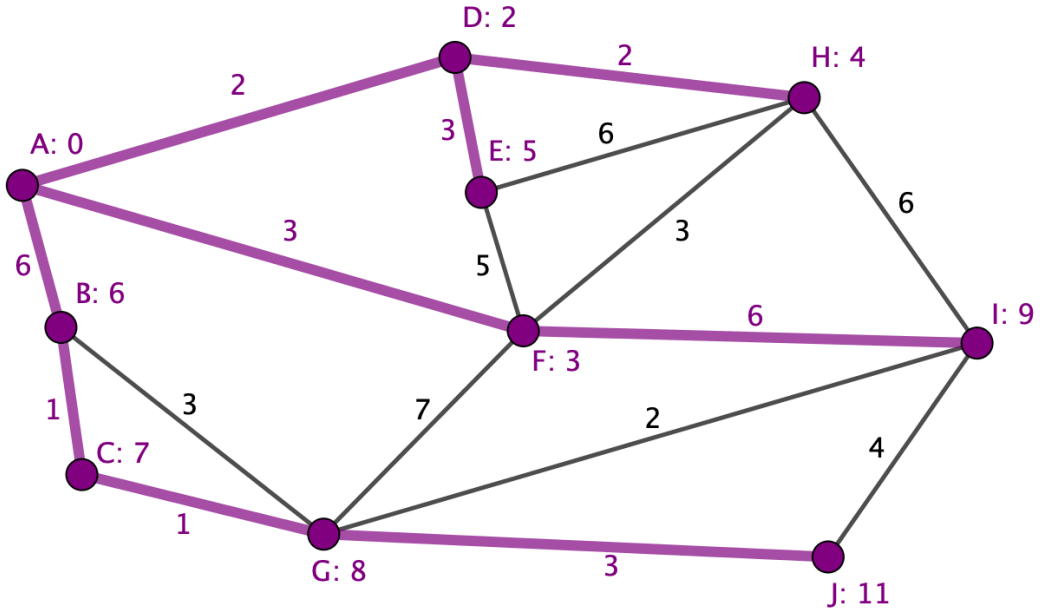
Solution:



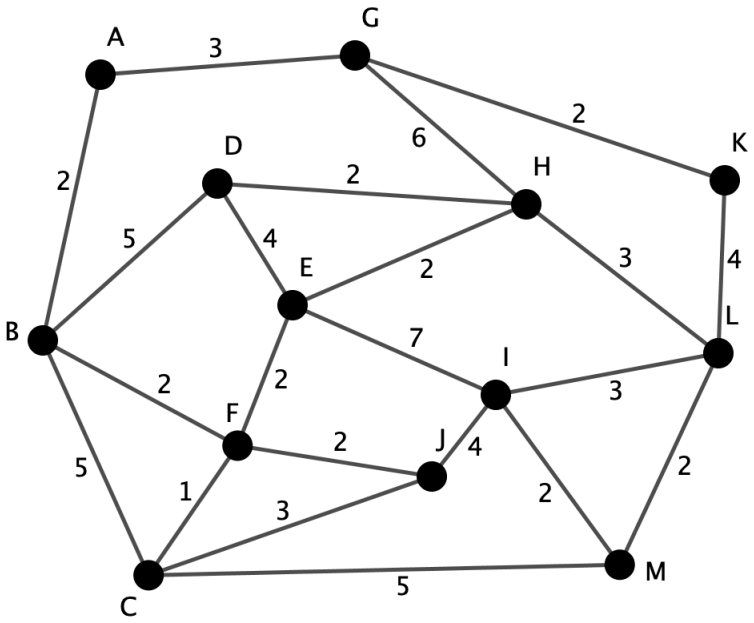
5. Use Dijkstra's Algorithm to find the shortest path from A to J.

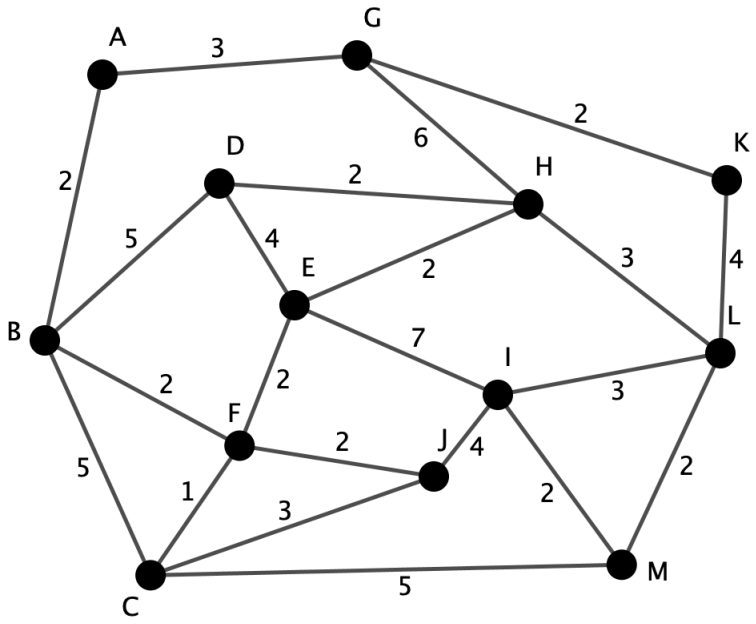


Solution:



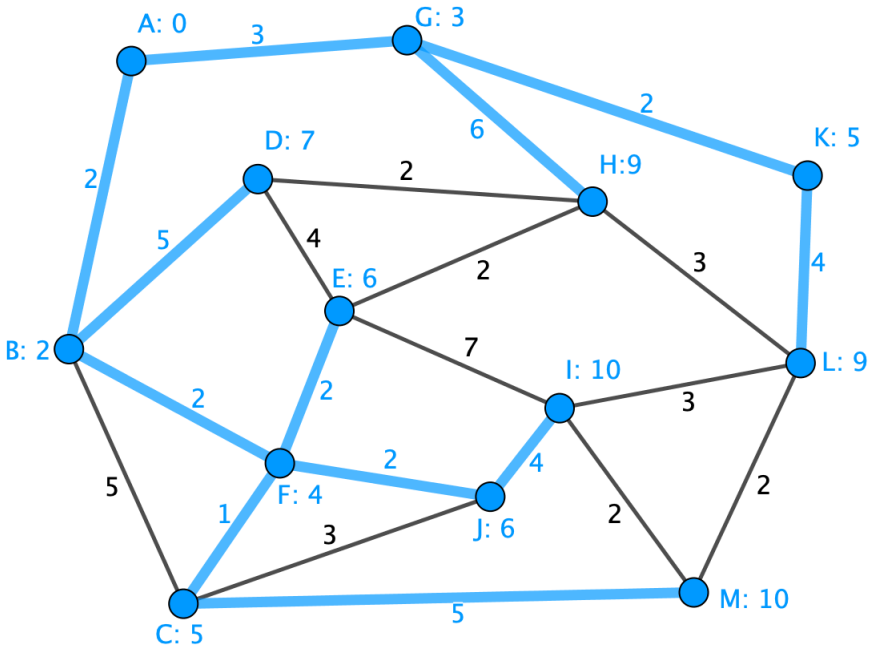
6. Use Dijkstra's Algorithm twice in the below graph. Once starting at A and once starting at M. Do we get the same path from A to M as M to A? Two copies of the graph have been given. *Hint: If you get to a point where you have multiple vertices in your unvisited list with the smallest labelled cost, you can choose any of the vertices with that smallest cost to be your next current vertex.*



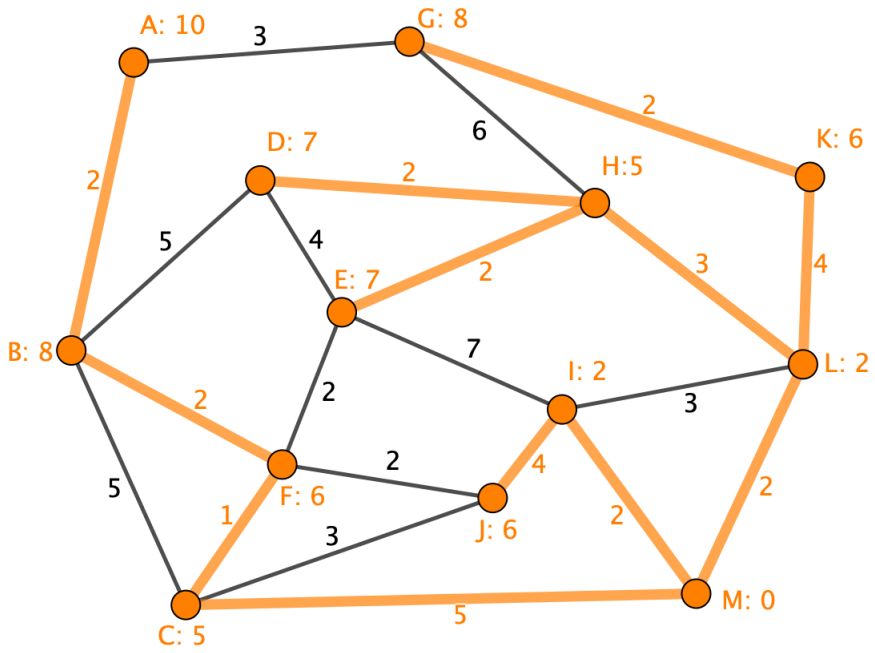


Solution:

Starting at A:



Starting at M:



In this case, we do get the same path from A to M, as seen above. That said, it is also, in general, possible to have a different path, so long as the weight of the path is still minimum (in this case 10).