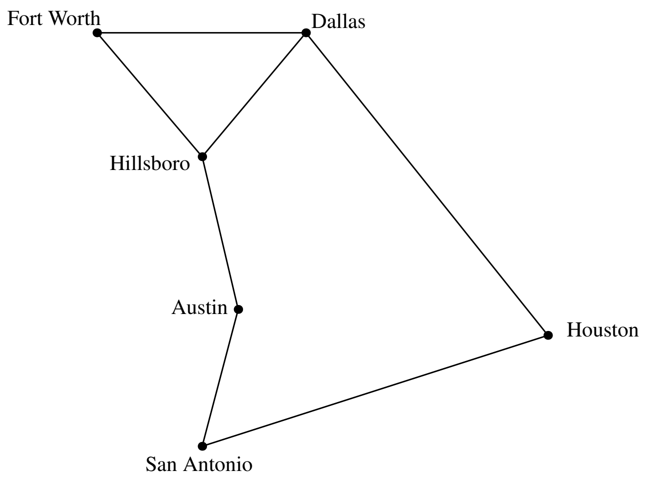
New Topics for Junior High Mathematics (2014-2015)

Graph Theory [1]

Graph theory is the mathematical study of things and how these things are connected. These "things" can be cities which are connected by roads, people on Facebook who are connected when they are friends, or actors who are connected when they starred in the same movie or TV show together. Today's society is full of networks (another name for graphs). Exploring these networks is the primary reason for studying graph theory.

A graph consists of **vertices** (plural of *vertex*), which represent the items that are connected and **edges**, which represent when two vertices are connected.

For example, suppose you have a graph with selected cities in Texas where two cities are connected by an edge if they are connected by an interstate highway. Here, Dallas and Fort Worth are connected by IH-30 and Austin and San Antonio are connected by IH-35. These connections are represented by edges. However, there is no interstate that runs from Austin to Houston, so there is no edge between Austin and Houston.



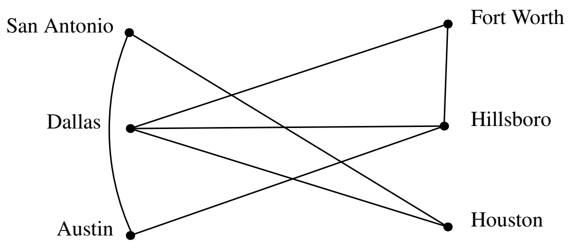
Terms and Definitions – The following terms are used throughout graph theory.

- **degree of a vertex** – the number of edges incident with the vertex
- **regular graph** – a graph where every vertex has the same degree
- **order** – the number of vertices of a graph
- **size** – the number of edges of a graph
- **adjacent** – two vertices are adjacent if there is an edge between them

In the graph above, the degree of the vertex Dallas is 3. This graph is not regular since the degree of Austin is 2 and the degree of Dallas is 3. The order of this graph is 6. The size of this graph is 7. Vertices San Antonio and Houston are adjacent. Adjacency can be represented by the following notation: San Antonio $\leftrightarrow$ Houston.

[1]This document was prepared by Doug Ray for competition in years 2014 and 2015. If you have any questions about the material presented, please email `doug@academicmeet.com`.

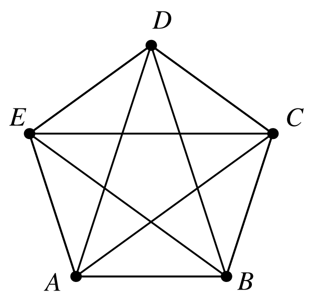
The way a graph is drawn is not important. Only the vertices and their connections are important in describing the graph. The graph shown here is exactly the same as the one above. However, the graph above has the advantage of the familiar look of the map of Texas.



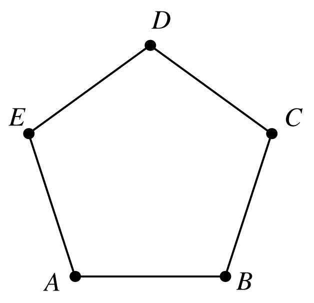
Types of Graphs

- **complete graph** – K_n – a graph with n vertices where every vertex is connected to every other vertex
- **path** – P_n – a list of n vertices that are connected in sequence with no repeated vertices
- **cycle** – C_n – a series of connected vertices where the last vertex is connected to the first vertex
- **tree** – a graph that is connected (there is a path between any two vertices) and acyclic (no cycles)
- **bipartite** – a graph where the vertex set can be split into two groups where edges must connect vertices from one set only to the vertices in the other set
- **complete bipartite** – $K_{m,n}$ – a bipartite graph with all possible edges having m vertices in one part and n vertices in the other part

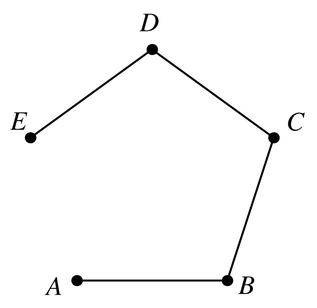
Examples:



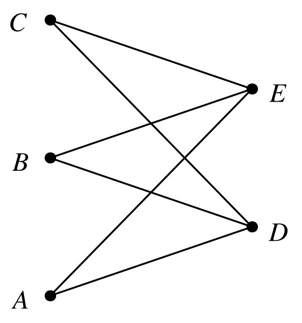
Complete graph on 5 vertices – K_5



Cycle with 5 vertices – C_5



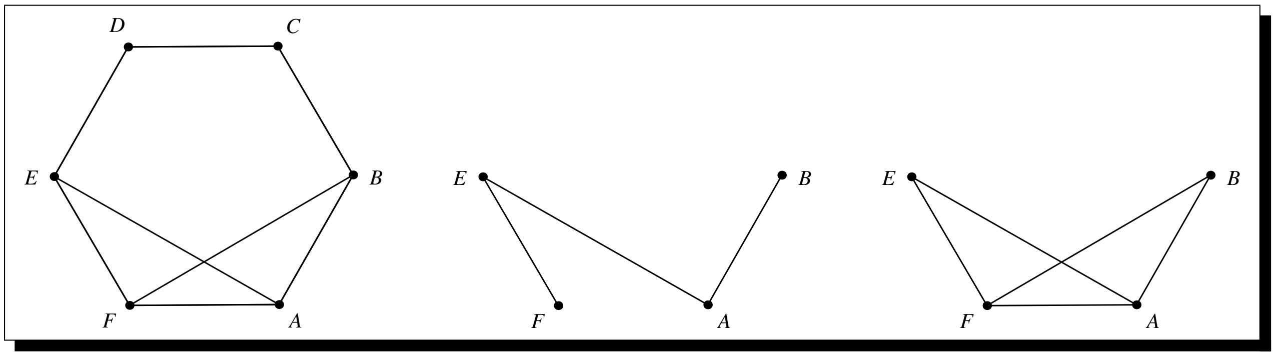
Path on 5 vertices – P_5



Complete bipartite graph – $K_{3,2}$

A **subgraph** of a graph is a graph composed of any subset of the vertex set and any subset of the edge set (with vertices from the subset). An **induced subgraph** is a subgraph that contains all the edges from the original graph between any two vertices of the new subgraph.

For example, suppose G is the graph shown below on the left. One subgraph of G is the middle graph; let's call it H. It contains the vertices A, B, E, and F. Notice that G contains the edges BF and AF, but graph H is missing these edges. Therefore, H is not an induced subgraph. An induced subgraph must contain all of the edges among the vertices of the subgraph that are part of the original graph. The induced subgraph with vertices A, B, E, and F is shown on the right.



Suppose we had a system of seven computers, named A through G. We need to connect all six computers together in a network. We do not need to connect every computer directly to every other computer. Instead, we just need is ensure that there is a path from any computer to any other computer (perhaps through other computers). What happens if computer A is connected to computer B, computer B is connected to computer C, and computer C is connected to A? This is redundant since if any two of these connections are present, all three computers are connected. This suggests that any optimal network should be acyclic. Since we also require all computers to be connected, the optimal graph for this situation must be a tree (acyclic and connected).

Further consider this situation. What happens if it costs less to connect computer A to computer B than it costs to connect computer B to computer C? This suggests that some connections might be better than others.

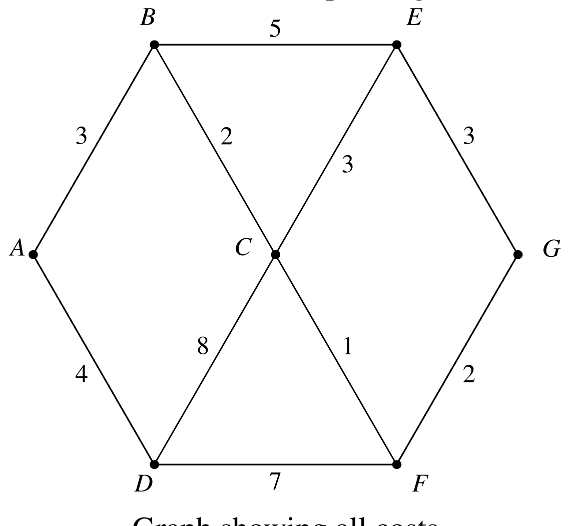
A **weighted graph** is a graph where the edges have been assigned numerical values. The cost to connect any two computers in the above situation is the weight of the edge between those two computers. Another application of weighted graphs is when graphs represent maps and the weights are distances between two cities that are connected by a road. The **weighted distance** between two vertices is the smallest possible sum of weights along edges that connect the two vertices.

The **minimum spanning tree** of graph is the subgraph that is a tree (connected and acyclic) and the sum of the weights of the edges is as small as possible.

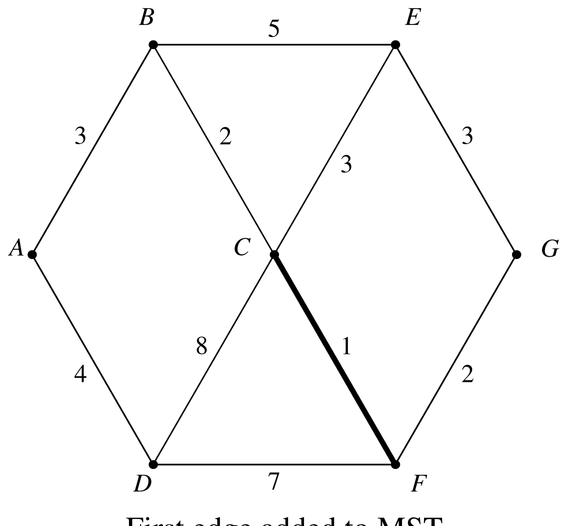
The graph below shows the seven computers that we need to connect together in a network. The weights on the edges give the cost that is required in order to connect the two computers incident with that edge. Not all vertices are connected with an edge, meaning that there is no way to directly connect those two computers. The smallest cost required to connect all the computers is found by creating the minimum spanning tree. Minimum spanning trees (MST) are found as follows:

- Start by selecting the smallest weighted edge and add it to the MST.
- Continue picking the smallest available weighted edge to add to the MST as long as adding this edge does not produce a cycle.
- If there are two or more edges with the same smallest weighted edge value, you can pick any of them to add to the MST.
- The process is complete when all vertices are connected.

As an example, suppose we know the costs of connecting the seven computers together. These costs are represented as weights on the graph shown below. The smallest weight is 1, on edge CF. We add CF to our minimum spanning tree, as shown.



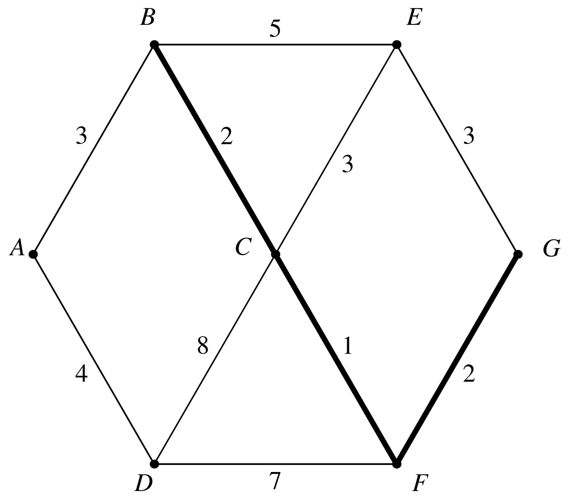
Graph showing all costs



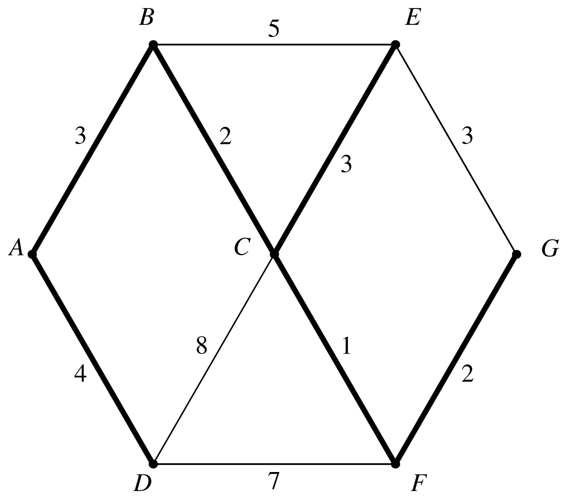
First edge added to MST

The smallest weight available now is 2. Adding both edges with weights of 2 maintains the tree and we do it. Next, there are edges with weight 3. We can easily add edge AB to the MST.

However, if we add both edges CE and EG, we will have formed a cycle $CEGF$. We are not allowed to form a cycle, since the goal is just to ensure that all vertices are connected to the other vertices without redundancy. Once one of these edges is added, there is no need to add the other one. Moreover, it does not matter which edge we add. We will add edge CE. The only vertex not part of the MST at this point is D. The three edges incident with vertex D have weights of 4, 7, and 8 and we add the edge with weight 4. This completes the minimum spanning tree. The total cost of connecting all seven computers is 15.



Add edges with weight 2



Completed MST

Consider the same graph as above. Suppose now we want to know the smallest distance from vertex A to every other vertex in the graph. The **distance** between two vertices A and B, denoted by $d(A, B)$, is defined to be the smallest sum of the weights along a path between the two vertices. (If the graph is unweighted, then the distance is defined to be the number of edges between the two vertices.)

To find the distance from a fixed vertex, in this case A, to all of the other vertices, we will use the following idea. The distance between two vertices is built up using the edge weights to extend already known shortest paths from vertex to vertex. A chart is helpful in keeping track of the distances already found.

To start, we will keep track of the tentative distances from vertex A to vertex x and label them $t(x)$, where x represents any of the other vertices. If A is adjacent to a vertex x, assign $t(x)$ the value of the weight of the edge between Ax; if they are not adjacent, let $t(x)$ blank. The initial setup chart should look like this:

In our first step, add the tentative distances from A to each vertex adjacent to A. This would be $t(B) = 3$ and $t(D) = 4$.
Next, choose the smallest possible tentative distance available. This must be the shortest distance from that vertex to A. In this case, it is vertex B and the distance is 3. Update the chart be declaring the distance from A to B, $d(A, B)$, to be 3.

x	$d(A, x)$	$t(x)$
B	**3**	3
C		
D		4
E		
F		
G		

x	$d(A,x)$	$t(x)$
B	3	3
C		**5**
D		4
E		**8**
F		
G		

Next, we need to update the list of tentative distances. The only distances that could possibly have changed are the ones to vertices that are adjacent to vertex B, the vertex we have just establish a shortest path for. Only vertices C and E are adjacent to B. Since both of these currently do not have a tentative distance assigned, the distances from vertex A to each of these is the distance from A to B added to the distance from B to each of them. Update the table with these sums.

x	$d(A,x)$	$t(x)$
B	3	3
C		5
D	**4**	4
E		8
F		**11**
G		

Next, choose the smallest available number in the tentative distance list (from among the vertices not yet chosen). This is vertex D with a distance of 4. Update the table of tentative distances based on the new vertices that are adjacent to D. The tentative distance from A to F is the sum of the distance from A to D and the weight of the edge DF, $4+7 = 11$. Notice, that we must reconsider the tentative distance to vertex C as well. The current tentative distance is $t(C) = 5$. The edge DC has weight 8, so the distance from A to C by going through D would be $4 + 8 = 12$. Since $12 > 5$, we do not update $t(C)$ since it is already smaller.

x	$d(A,x)$	$t(x)$
B	3	3
C	5	5
D	4	4
E		8
F		~~11~~ **6**
G		

The smallest available distance in the chart is $d(A,C) = 5$. Update the tentative distances: $t(F)$ improves to 6 by going through vertex C, but $t(E)$ is unchanged.

Next, F is the closest available vertex, with distance 6. Update the tentative distances.

x	$d(A,x)$	$t(x)$
B	3	3
C	5	5
D	4	4
E		**8**
F	**6**	6
G		**8**

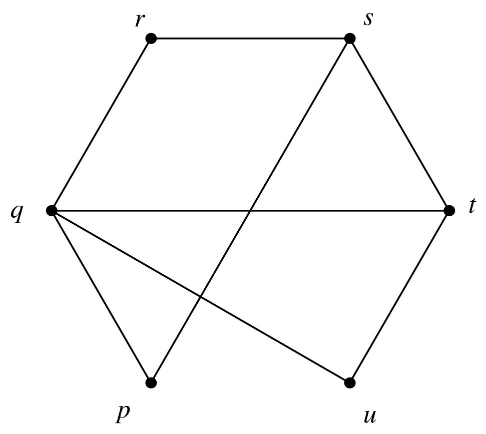
Finally, vertices E and G remain and both are a distance of 8 away from A. Both vertices are chosen and the distances are found.

x	$d(A,x)$
B	3
C	5
D	4
E	8
F	6
G	8

Other topics for research: Hand-shaking lemma, (vertex) coloring, independent set, eulerian graph, hamiltonian graph

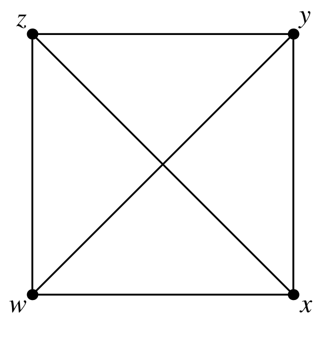
Exercises:

1. Given graph G, answer each of the following:



   (a) What is the order of G?
   (b) What is the size of G?
   (c) Is G regular?
   (d) Which vertices are adjacent to r?
   (e) What is the degree of t?
   (f) Which vertex has the largest degree?

2. Given graph G, answer each of the following:



   (a) What is the order of G?
   (b) What is the size of G?
   (c) Is G regular?
   (d) Which vertices are adjacent to w?
   (e) What type of graph is G? Give the symbol for G.

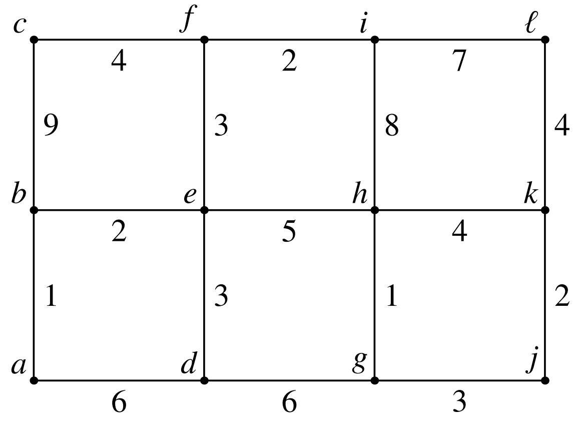
3. Determine the order and size of each of the following graphs. Draw each graph.

   (a) K_3
   (b) K_5
   (c) P_6
   (d) C_4
   (e) $K_{4,2}$

4. Determine the order and size for each of the following graphs in terms of n and m.

   (a) P_n
   (b) C_n
   (c) K_n
   (d) $K_{n,m}$
   (e) a tree with n vertices

5. In the graph shown, find the minimum spanning tree. What is the sum of the weights on the MST?



6. Using the graph above, find the distance from vertex a to vertex ℓ.

7. Let $G = K_5$. How many different 3-cycles are subgraphs of G?