

Combinatorics: The Fine Art of Counting

Week 8 Lecture Notes – Graph Theory

Introduction

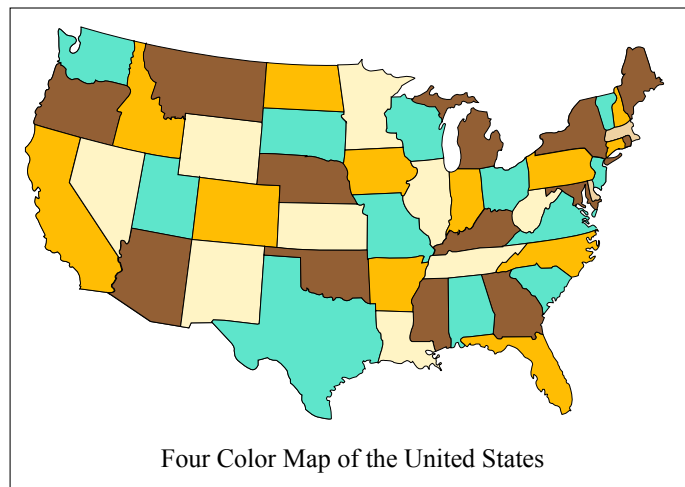
More than any other field of mathematics, graph theory poses some of the deepest and most fundamental questions in pure mathematics while at the same time offering some of the most useful results directly applicable to real world problems. There are currently five (four?) unsolved problems in mathematics which are considered so important that there is a \$1,000,000 reward for anyone who solves them. One of these problems, the “**P vs NP**” question, is directly connected to several fundamental problems in graph theory. Graph theoretic algorithms are used every day to solve problems in manufacturing, transportation, molecular biology, computer networking, finance, electrical engineering, digital imaging, ...- the list goes on and on. The many unsolved problems in graph theory and the wide range of practical applications make it a rich field of current mathematical research.

A Whirlwind Tour of Graph Theory

We saw several examples of graphs during the first lecture of this course when we looked at the regular polyhedra. Graphs provided an abstract way of capturing the essential properties of these geometric objects – the relationships between the vertices, edges, and faces. Today we will look at some other examples where graphs arise. The virtue of using a graph to represent a problem is that it captures the essential features of a problem in a concise manner that provides a lot of flexibility (e.g. looking at the polyhedra as planar graphs).

Example #1 – Maps

Graphs are an obvious way to capture the essential features of a map. Consider a geographic map where we use vertices to represent contiguous regions with edges between adjacent regions. A famous problem for this type of map is the **Four-Color Theorem**, which states that the vertices of any planar graph can always be colored with four colors in such a way that no two adjacent vertices have the same color. It is easy to see that three colors is not enough (consider the graph of the tetrahedron), and it is not too hard to prove that five colors is sufficient, but the Four-Color Theorem is extremely difficult. The current proof is far from satisfactory; it involves checking so many special cases that a computer is required just to complete the proof.



Other types of maps that give rise to graphs may be more abstract, such as a map of a transportation system (e.g. a map of interstate highways, a train system, or an airline route map) or a network (e.g. a telephone network, a power grid, the internet). In graphs of this type, important questions often relate to **paths** in the graph. Is there always a path between any two vertices in the graph? If so, the graph is said to be **connected**, if not the graph can be separated into two or more **connected components**. To ensure the reliability of a system it may be important that connectivity persists even if one or more edges or vertices is removed (due to an accident or system failure). Which subsets of edges will disconnect a connected graph if removed? Such a set of edges is called a **cut**. The reliability of a network is related to the size of the smallest cut.

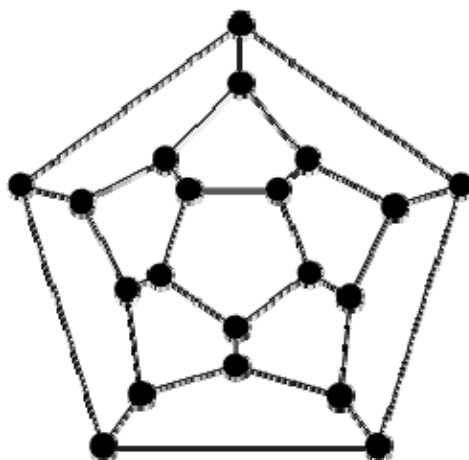
Image of Seoul Subway System removed due to copyright restrictions.

Please see: <http://www.nsubway.co.kr/korea/seoul/seoulsubwaymapen.htm>

What is the **shortest path** between two vertices? You may want to know this if you are planning a trip, routing a product shipment or sending an instant message over the internet. A generalization of this question is to consider the worst case, i.e. what is the longest shortest path between any two vertices in a connected graph. This value is called the **diameter** of the graph, and it is often desirable to have graphs that have small diameter. Small diameter graphs are important when solving routing problems, but can also play a role in social networks (see below).

A generalization of the notion of a path in a graph is a **walk**. We imagine starting at some vertex in the graph and then walking from one adjacent vertex to the next along edges. If we never use the same edge twice, the walk is called a **trail**. If we never visit the same vertex twice, then the walk is a **path**.

A famous question regarding the length of walks in a graph is the **Traveling Salesman Problem** – assuming the edges are labeled with distances, what is the shortest walk that goes through all the vertices? An apparently simpler question might be whether there is a walk which passes through all the vertices exactly once, i.e. a path containing all the vertices. Such a path is called a **Hamiltonian path**. A pastime called “Around the World” popularized by Sir William Hamilton involved finding Hamiltonian paths (or cycles) in the graph of a dodecahedron (see below). There is no general method known for efficiently determining whether a given graph contains a Hamiltonian path. However it is easy to check whether a given list of vertices is a Hamiltonian Path, thus if someone claims a graph contains a Hamiltonian path they can easily convince us by simply telling us the order of the vertices in the path. This is the essential feature of an NP-type problem. Some problems seem hard to solve, but the answer is easy to check once you have one. The “P vs NP” question asks whether NP-type problems are inherently harder to solve than problems where we have a reasonably efficient algorithm for finding the solution (these are P-type problems).



The remarkable thing about many NP-type problems (including the Hamiltonian Path problem), is that answering this question for just one of these problems would settle the question for *all* of them (problems with this property are called NP-complete). Thus to win \$1,000,000 you can pick any one of these problems and either solve it, or prove that it is unsolvable.

An apparently similar question is whether a graph contains a walk which passes through all the edges exactly once, i.e. a trail containing all the edges. Such a trail is called an **Eulerian tour** in honor of the prodigious Swiss mathematician Leonhard Euler. Among Euler's numerous mathematical achievements, he was the person most responsible for the creation of graph theory as a formal field of mathematical study. In contrast to the Hamiltonian Paths, determining whether a graph contains an Eulerian tour is straight-forward. The question which motivated Euler to pursue this topic was whether it was possible to take a walk which crossed all the bridges in Königsberg exactly once.

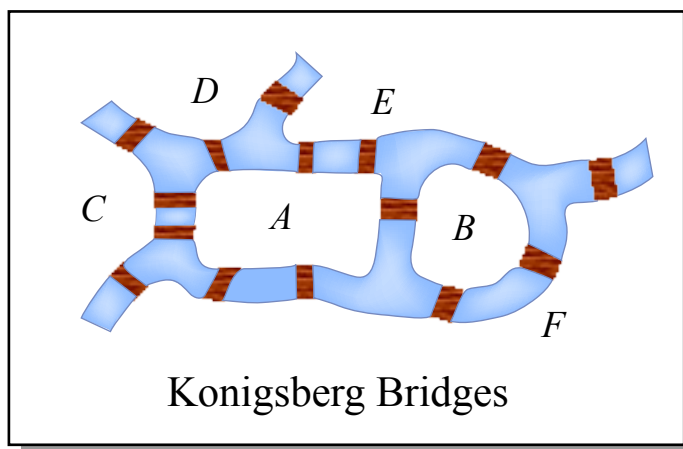


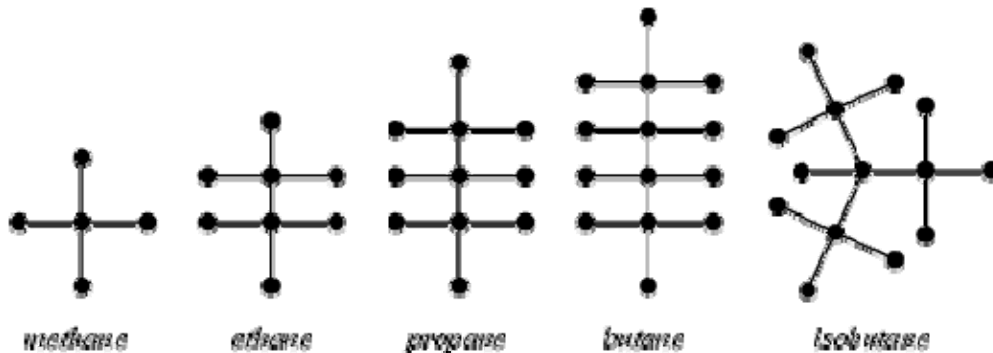
Image by MIT OCW.

Example #2 – Physical Models

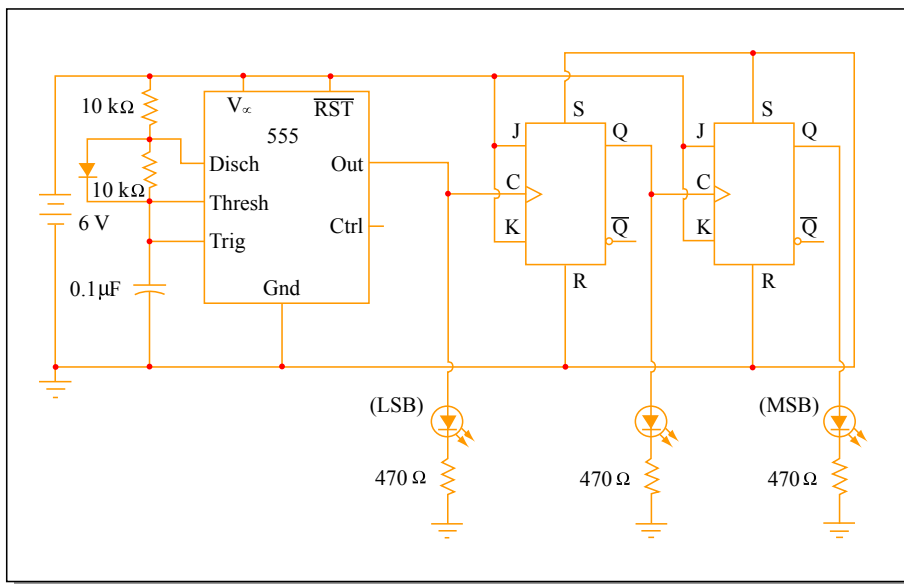
Many of the most important molecules in biology and chemistry have complex structures that can be described with graphs. The vertices in such a graph represent atoms, and the edges represent bonds between atoms. An important feature of such graphs may be the **degree** of the vertices - i.e. the number of adjacent vertices. This corresponds to the number of bonds a particular atom has in the molecule. There are chemical properties of atoms of a given type that determine the number of bonds they typically form (e.g. 1 for hydrogen, 2 for oxygen, 4 for carbon, etc...) and it may be important that the graphs satisfy certain constraints regarding the degrees of the vertices. In order to represent double bonds between atoms, it may be appropriate to extend our notion of a graph to a **multi-graph** which permits multiple edges between vertices (note that the Königsberg Bridge diagram above corresponds to a multi-graph).

One particularly important category of molecules are hydrocarbons (molecules made of carbon and hydrogen), and in such graphs an important feature is whether there are any **cycles** present,

and if so, how many and of what size. A connected graph without any cycles is called a **tree**, and many saturated hydrocarbons have this structure (some small examples are shown below). In the field of nano-technology, complex carbon molecules play an important role, e.g. the molecule C_{60} , also known as Buckminsterfullerene or a “bucky ball”. This molecule contains many cycles composed of 5 or 6 carbon atoms (it is a truncated icosahedron). We will learn a number of important properties related to characterizing graphs which are trees, and analyzing the cycles in graphs which are not trees.



Graphs can be used to model problems in physics as well. The problem of computing the current flow through the wires of an electrical circuit is critical for designing complex electronic devices. This problem was greatly simplified by the German physicist and mathematician Kirchhoff who found a way to analyze the graph representing any electrical circuit in terms of trees and cycles.



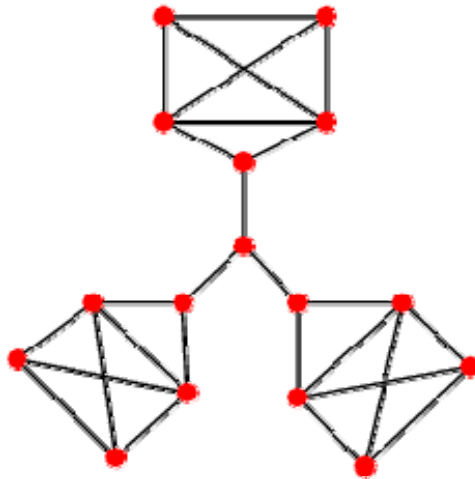
Example #3 – Relations

Image by MIT OCW.

The last and most important example we will look at are graphs which represent relationships between the elements of in a set of objects. Using social networks as an example, we can represent the various acquaintances within a group of people by having a vertex for each person and placing an edge between people who know each other. An interesting fact that can be proven using such graphs is that among any group of six people, there are either three people who all know each other, or three complete strangers. Another aspect of social networks that can be analyzed with graph theory is the “small-world” phenomenon. It has been conjectured that the graph of acquaintances among all the earth’s population has a diameter of six. Pick any person on earth, and you know someone that knows someone that ... knows this person and you only have to go through five people in between.

Rather than using edges to represent acquaintances between people, we might instead use them to represent compatibility – suppose we want to divide people up into pairs by matching compatible people (e.g. in a dating service). A set of edges in a graph which do not have any vertices in common is called a **matching**. If we are able to pair up all the vertices in the graph, we have a **perfect matching**. When does a perfect matching exist (can you find one in the graph

below)? If so how many are there? If not what is the largest matching possible? In a trading system it may be important to match buyers with sellers, in a chess tournament between teams from two different schools we may want to match opponents from opposing schools. In both of these examples we can separate the vertices of the graph into two groups (e.g. buyers and sellers) and we are only interested in edges between vertices in opposite groups. Such a graph is called a **bipartite graph**. Another example of a matching problem is tiling a region of the floor with identical dominos. How many ways can you do this? How many ways can you put the dominos back in the box? Both of these questions can be represented as matching problems.

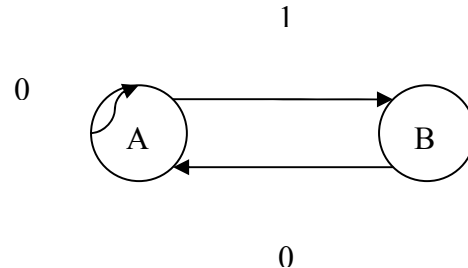


In a tournament, we may want to not only represent the opponents which are matched up, but the outcome of the game. In this situation we can generalize our notion of a graph to include the concept of **directed** edges with an arrow indicating the order of the two ends of the edge. We can place a directed edge pointing from the winner to the loser, and then use such a graph not only to declare an overall winner, but also to assign a ranking among the participants. An interesting question about such graphs is whether they include a **cycle** (e.g. A beat B who beat C who beat D who then beat A) and if so, how should players be fairly ranked in such cases?

Another example of relationships between objects are dependencies. A large project or complex assembly process may consist of many different steps, some of which may be done in parallel, but some may depend upon the completion of earlier steps. These dependencies can be represented in a graph using directed edges (see example below). In order to optimally plan such a project, it may be important to know both how long the longest path in such a graph is (often called the **critical path**), and also the maximum number of “parallel” edges, i.e. a set of edges no two of which lie on a path (this is a special type of cut). Another example of dependencies are logical implications between propositions in a formal proof system, i.e. statements of the form “A implies B”. Proving that a given theorem can be derived from a set of axioms may amount to finding a directed path in a graph in which the vertices represent propositions and the edges represent logical implications.

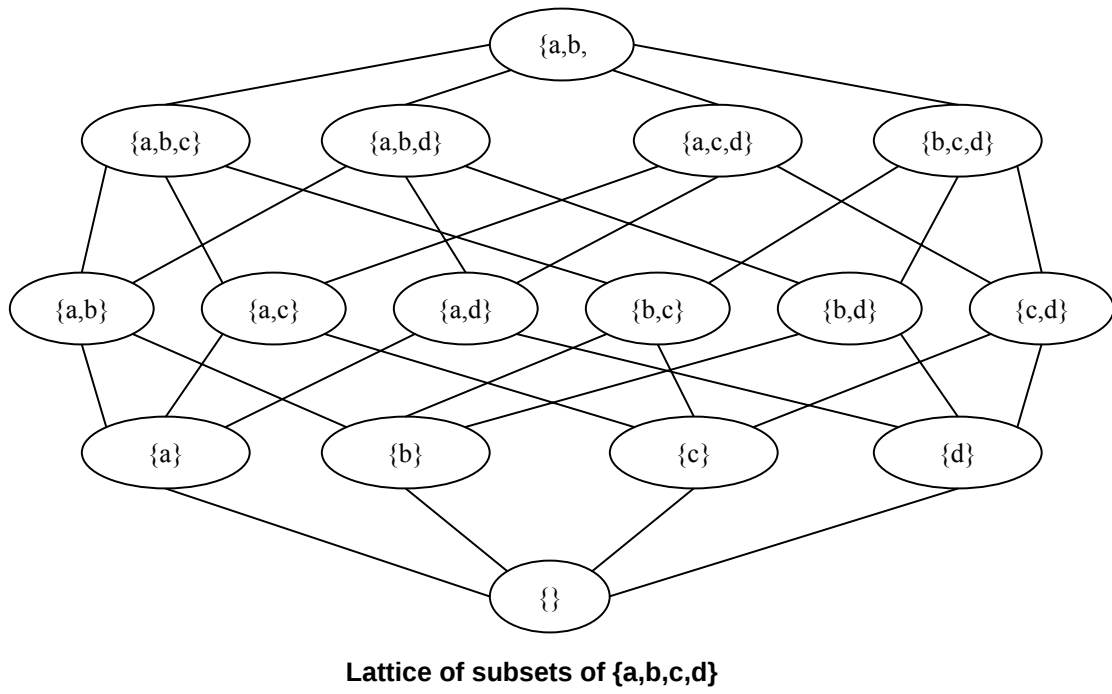
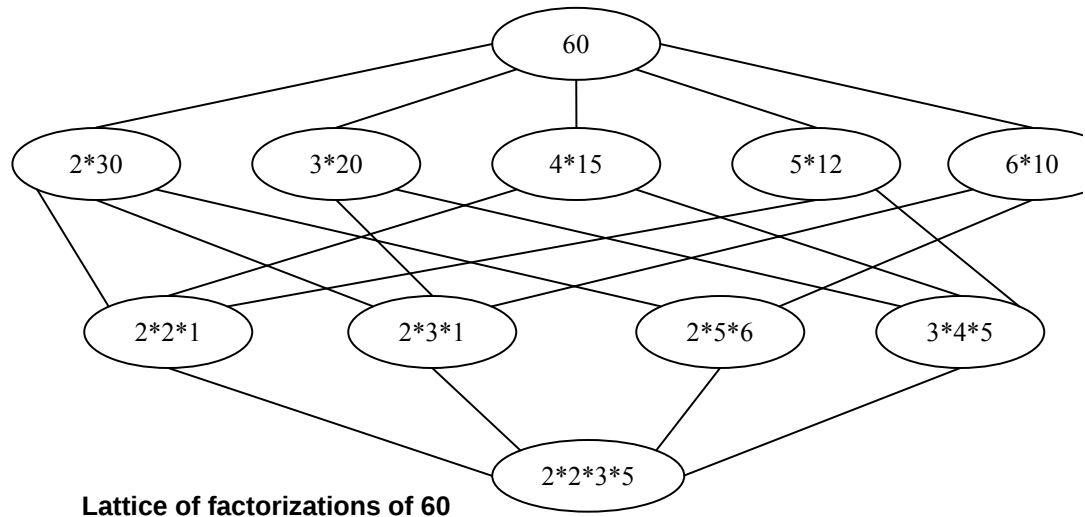
Another type of dependency graph is a state transition diagram. Consider a machine or computer program which may be in one of several different states. Transitions between states can be represented by a directed edge between vertices representing the states. The execution of the machine may be modeled as a tour along the directed edges of the graph. Note that in transition diagrams we may want to allow **self-loops** as well as normal edges. One question of particular interest for such graphs are whether a directed path exists from one state to another. Another interesting question is the number of possible tours of a given length. Consider the graph of the finite state machine depicted below. This simple state transition diagram represents a machine which recognizes binary strings which do not contain two 1's in a row. The number of directed

tours of length n which start at vertex A is equal to the number of binary strings of length n which do not contain two 1's in a row. This turns out to be equal to the $(n+1)^{\text{st}}$ Fibonacci number. This fact and many more complex string recognition problems can easily be proven by analyzing the graph of an appropriate state transition diagram.



Graphs can be used to represent negative relationships as well as positive ones. Consider the problem of scheduling a set of extracurricular activities for which various students have registered. If we associate a vertex with each activity and place an edge between any two activities which have a student in common, we can use this graph to determine how the activities may be scheduled. If we can **color** the vertices of the graph in such a way that no adjacent vertices have the same color, this coloring can be used to assign time slots to the activities in such a way that no student is scheduled to do two things at the same time. It may be important to try to minimize the number of time slots (there are only so many hours in the day), so we may want to determine the minimum number of colors required. Recall that the Four-Color Theorem applies only to planar graphs, but for more general graphs, more than four colors may be required (e.g. the **complete graph** K_n requires n colors). The minimum number of colors necessary to color the vertices of a graph so that no adjacent vertices have the same color is called the graph's **chromatic number**. Some graphs require a lot of colors (e.g. the **complete graph** K_n has chromatic number n), while others don't (e.g. the **complete bipartite graph** $K_{m,n}$ has chromatic number 2). Graph colorings can be used to solve a wide variety of problems and there are many open questions related to graph coloring.

The last two examples we will mention in passing are the graphs of the factors of 60 and the subsets of the set $\{a,b,c,d\}$. These graphs are examples of mathematical objects which have a particular type of relationship defined among the elements called a **partial ordering**. In a partial ordering some pairs of elements have a well defined order (e.g. the set $\{a,b,c\}$ contains the set $\{a,b\}$), but not necessarily all pairs do (e.g. the set $\{a,b\}$ neither contains nor is contained by the set $\{b,c\}$). Graphs of partial orders are of particular interest in combinatorics, and the two examples given here are a very special type of partial ordering called a **lattice**.



Definitions

Before we can study graph theory in more detail, we need to define some of our terms a little more precisely. Graph theory uses a lot of definitions, but fortunately they are generally very straight-forward and intuitive (unlike probability).

Graph: A graph $G=(V,E)$ consists of an arbitrary set of objects V called **vertices** and a set E which contains (unordered) pairs of (distinct) elements of V called **edges**.

The vertices contained in an edge are said to be **adjacent** and form the **end-points** of the edge. Two edges are said to be adjacent if they have a vertex in common. Vertices are sometimes called *nodes* or *points* but vertex is the most common term.

Notice that the set V can be whatever we like as long as we have a way to distinguish the vertices. When speaking in general terms we will usually use an abstract set $V = \{v_1, v_2, \dots, v_n\}$, but in practical problems it will be convenient to let the elements of V correspond to objects in the problem (e.g. people, places, events, etc...). You can think of V as a set of labels or names for the vertices.

Directed Graph: We extend the definition above by requiring the edge set E to contain *ordered* pairs of (distinct) elements of V .

By convention, the pair of vertices in an edge are distinct unless **self-loops** are explicitly permitted (a graph with self-loops is sometimes called a pseudo-graph).

Degree: For any vertex v in a graph, the degree of the vertex equal to the number of edges which contain the vertex. The degree of v is denoted by $d(v)$.

Theorem 1: The sum of the degrees of all the vertices in a graph is equal to twice the number of edges, i.e. $\sum d(v) = 2|E|$

Proof: We saw this proof in the very first lecture. If we add up the degrees of all the vertices in a graph, we are counting every edge twice, once for each end-point.

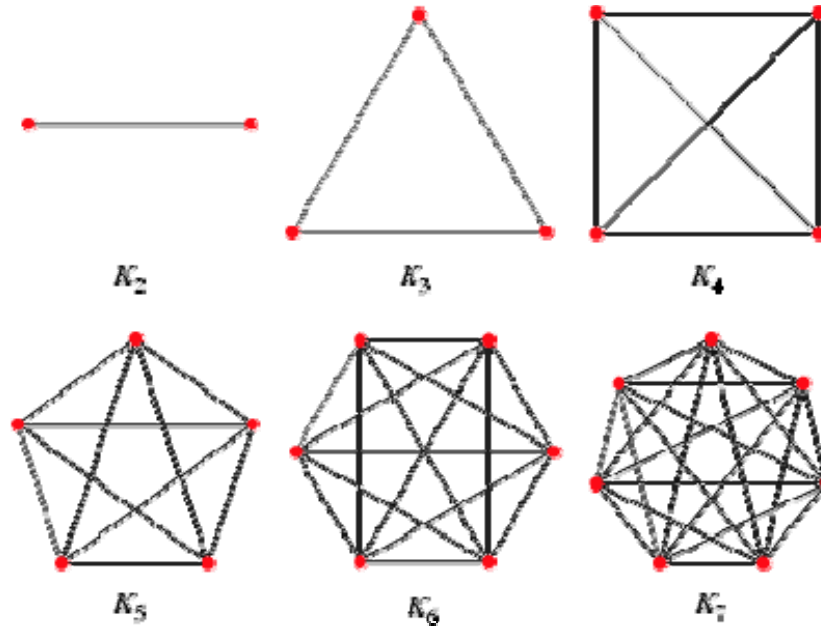
Note that this implies that the sum of the degrees of a graph must always be an even number. This is a useful fact to remember.

Regular Graph: A graph in which every vertex has the same degree is called a regular graph. If all vertices have degree n , the graph is said to be **n-regular** or simply to have degree n .

For example the cube is a 3-regular graph. A cycle (defined below) is a 2-regular graph. Note that since the sum of the degrees must be an even number, a regular graph with odd degree (e.g. the cube) must have an even number of vertices.

Complete Graph: The complete graph on n vertices K_n consists of vertex set $V = \{v_1, v_2, \dots, v_n\}$ and the edge set E containing all pairs of vertices in V .

K_n is a regular graph of degree $n-1$.



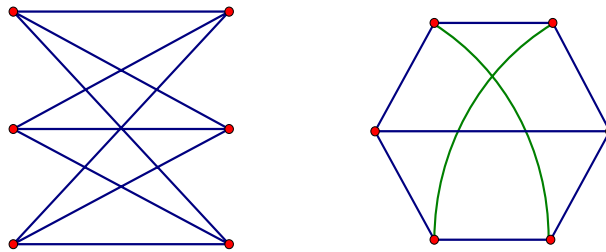
A graph may be specified by listing the set of vertices and the set of edges. Alternatively, a graph may be specified via a table called an **adjacency matrix** which contains a row and a column for each vertex. If the vertex set is $\{v_1, v_2, v_3, \dots\}$, the entry in the i^{th} row and j^{th} column of the adjacency matrix is 1 if the edge (v_i, v_j) is in E and 0 otherwise. Note that the diagonal entries of the adjacency matrix will be 0 unless self-loops are permitted, and the adjacency matrix will always be symmetric if the graph is undirected. Below is an adjacency matrix for the graph C_4 .

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

Adjacency Matrix of C_4

The third and most common way to specify a graph is to draw a diagram. However an important issue arises when drawing a graph on a piece of paper. Consider the graph with the set of vertices $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ and edges (v_1, v_2) , (v_1, v_5) , (v_1, v_6) , (v_2, v_3) , (v_2, v_4) , (v_3, v_5) , (v_3, v_6) , (v_4, v_5) , and (v_4, v_6) . Try sketching this graph in the space below before reading on.

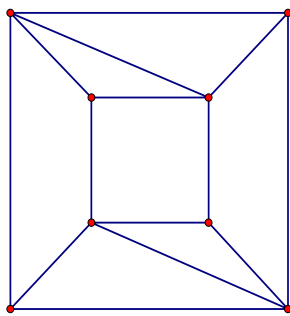
Two drawings of this graph are shown on the next page. It is likely that your drawing is not the same as either one. There are many ways to draw the same graph, and for a graph of any complexity, it is not always obvious when two drawings represent the same edge relationships.



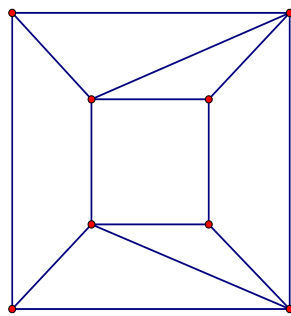
Two Drawings of $K_{3,3}$

Conversely, a given drawing may have the vertices labeled in different ways which may result in different edge sets. If we swapped the labels of the vertices v_1 and v_2 in the example above, we would have to change all of the edges which contained these vertices to interchange the roles of v_1 and v_2 in order to keep the graph the “same”. A re-labeling of a graph along with the corresponding changes to the edge set is called an **isomorphism** (a fancy word that means “identical shape”). The concept of an isomorphism is analogous to the geometric concept of congruence, except that edges don’t have a particular size or shape, just end-points, so the only thing that matters is which vertices are adjacent and which aren’t.

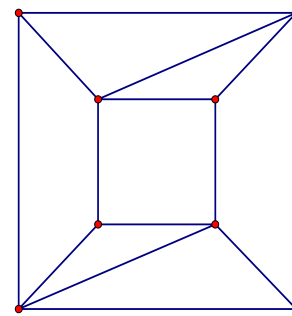
The question of when two graphs are isomorphic is a difficult one in general. As an example, consider the graphs labeled G_1 , G_2 , and G_3 . Which of these graphs are isomorphic? How can you prove this? Graphs G_1 and G_3 are isomorphic (in this case we can simply rotate the page to see the isomorphism, but in general it is not so easy). Graph G_2 is not isomorphic to the other 2. To see this, note that each node of degree 4 in G_2 has two neighbors which have degree 4, while in G_1 and G_3 this is not true.



G_1



G_2



G_3

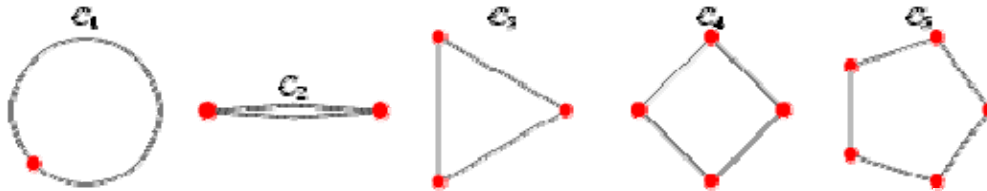
The problem of determining graph isomorphism is an unsolved problem. If someone claims two graphs are isomorphic they can easily prove this to you by giving you a re-labeling of the vertices of one of the graphs which you can then easily check. However without a re-labeling in hand, it is very hard to know one way or the other. If two graphs are not isomorphic, we may be able to find some simple feature which differentiates the two graphs (e.g. the degrees of vertices, or the existence of certain paths or cycles), but this is not always possible in general.

Sub-graph: A graph $G_1 = (V_1, E_1)$ is a sub-graph of $G_2 = (V_2, E_2)$ whenever $V_1 \not\subset V_2$ and $E_1 \not\subset E_2$.

If G_2 is a sub-graph of G_1 we say that G_1 contains G_2 . Note that E_2 does not necessarily need to contain all the edges in E_1 which connect two vertices in V_2 . When it does, we say that G_2 is the sub-graph of G_1 **induced** by V_2 .

Path: A path of length n is the graph P_n on $n+1$ vertices $\{v_0, v_1, v_2, \dots, v_n\}$ with n edges $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)$.

Cycle: A cycle of length n is the graph C_n on n vertices $\{v_0, v_1, \dots, v_{n-1}\}$ with n edges $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_0)$.



Note that C_1 only arises when self-loops are permitted and C_2 only arises in multi-graphs.

We say that a given graph **contains** a path (or cycle) of length n if it contains a sub-graph which is isomorphic to P_n (or C_n).

Note that under this definition the vertices of a path (or cycle) are always distinct. If we want to talk about an arbitrary sequence of adjacent edges in a graph which may pass through the same vertex or edge more than once, we will use the word **walk**. A walk in which all the vertices are distinct is a path. A walk in which all the edges are distinct is called a **trail**. A trail which begins and ends at the same vertex is called a **circuit**. A circuit which doesn't pass through the same vertex twice is a cycle.

For a given sub-graph which is isomorphic to a path or cycle, there will be more than one re-labeling which demonstrates this isomorphism because of the symmetry of paths and cycles (2 for a path, $2n$ for a cycle of length n). When counting paths or cycles in a graph we only count distinct sub-graphs.

Having said this, it is important to note that a subset of the vertices is not enough to fully specify a path or a cycle. The edges must also be specified and there may be multiple distinct sub-graphs which have the same vertex set but different edges. As an example, the graph K_4 contains **12 distinct paths of length 3** (each containing 4 vertices), and **3 distinct cycles of length 4**. Note that $2 \cdot 12 = 4!$ and $3 \cdot (2 \cdot 4) = 4!$

Connected: A graph contains a path between every pair of vertices is connected (note that the graph consisting of a single vertex is connected). In general, every graph is the union of one or more disjoint connected sub-graphs called the **connected components** – a connected graph has just one component, while a disconnected graph has more than one.

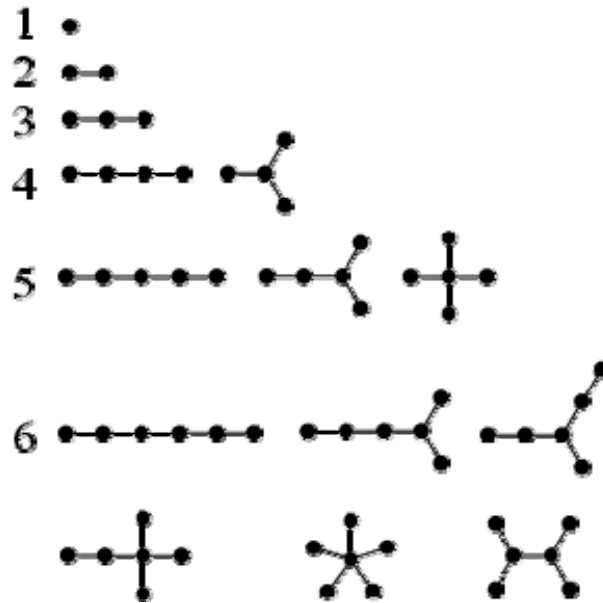
Two vertices in a graph are said to be connected if the graph contains a path which includes both vertices - note that in this situation there is a path in the graph which has the two vertices as end-points. In a connected graph, all pairs of vertices are connected.

Distance: The distance between two connected vertices is the length of the shortest path between the vertices.

Diameter: The diameter of a connected graph is the maximum distance between any two vertices in the graph.

Examples: K_n has diameter 1. The cube has diameter 3. P_n has diameter n , and C_n has diameter $n/2$ when n is even and $(n+1)/2$ when n is odd.

Forests and Trees: A graph which does not contain a cycle is called a forest. If it is a connected graph, it is called a tree. In general, the connected components of a forest are trees.



A graph with a single vertex (and no edges) is a tree. For any other tree, the degree of every vertex must be non-zero since the graph is connected. Vertices with degree 1 are called **end-points** or **leaves** of the tree. Vertices with degree greater than 1 are called internal vertices.

Note that if any edge of a tree is removed, the result is a forest with two trees, so in graph theory, cutting down a tree creates a forest.

We are now ready to use our definitions to prove a number of useful theorems. We will begin with an easy one.

Theorem 2: Every tree with at least one edge contains two end-points

Proof: Every vertex in a tree with at least one edge must have non-zero degree. Pick any vertex, and if its degree is not 1, walk along one of the edges containing the vertex and keep walking until you reach a vertex with degree 1. A tree contains no cycles, so we will never visit the same vertex twice. The tree is finite so we must eventually be forced to stop by a vertex with degree 1. This is one of the two end-points. Now turn around and start walking again until you reach another end-point. This is the second one.

QED

Our next theorem uses a proof by induction. Proofs by induction are often the simplest and clearest way to prove theorems in graph theory, and are a very useful tool in mathematics in general. Rather than avoid proofs by induction as some introductory texts do, we shall embrace them with open arms. The basic idea of an inductive proof is to start small (the base case), and then show you could in theory work your way up from this base case to any particular case. To do this we assume that the theorem is true for all cases smaller than the particular one we are trying to prove (this is called the inductive hypothesis) and use this to prove the particular case.

Key things to remember about proofs by induction:

- Make sure you are clear about what you are inducting on – i.e. what “n” is. It could be the number of vertices, the number of edges, the number of cycles, etc... Making the right choice may simplify the proof considerably.

- Be careful about the base case - don't just assume it is trivial (even though it usually is). Make sure you know where the base case starts – it won't always be at $n = 0$, and note that your proof won't apply when n is less than your base case.
- Remember that your inductive hypothesis applies to **all** values of n which are greater than or equal to the base case and less than n , not just to $n-1$. This is especially important in graph theory proofs because often we are breaking up graphs into smaller pieces, but the pieces aren't always just 1 smaller.

Theorem 3: A graph with n vertices is a tree if and only if it is connected and has $n-1$ edges.

Proof: First note that we have two things to prove, the “if” and the “only if”. Well actually we have infinitely many things to prove, since we want our theorem to be true for every n , but for each n we have two things to prove. In many situations it will be easier to prove these cases separately, i.e. prove the “if” for all n , and then prove the “only if” for all n , but here we will prove them both using a single induction on n , the number of vertices in the graph. The theorem is clearly true when n is equal to 1, since a single vertex is both a tree and a connected graph with 0 edges and there are no other graphs with 1 vertex. Now suppose that n is greater than 1 and that the theorem is true for all graphs with fewer than n vertices. We will now prove both parts of the theorem for a graph with $n > 1$ vertices.

Suppose the graph is a tree. It is clearly connected (by definition) and it must have at least one edge since $n > 1$. By Theorem 2, it has two end-points, so pick one of them. If we remove this end-point and the single edge containing it, the resulting graph is a tree with $n-1$ vertices, so by our inductive hypothesis it has $n-2$ edges. This plus the single edge we removed means the graph has $n-1$ edges in total.

Now suppose the graph is connected and has $n-1$ edges. By Theorem 1, the sum of the degrees of the vertices is equal to twice the number of edges or $2n-2$. The average degree is less than 2, so there must be at least one vertex with degree 1 (there must be two in fact). If we remove this vertex and the single edge containing it, we get a graph with $n-1$ vertices which is connected and has $n-2$ edges. By our inductive hypothesis, this graph is a tree. If we now add our vertex and edge back in, we have not created a cycle and the graph is connected, so it is also a tree.

QED

This is a simple proof, but it is worth understanding in detail, as it is a good model for more complex proofs.

There are many other important properties of trees, some of which are explored in the homework problems. We now want to turn our attention to some graphs which do contain cycles. There are two particular types of graphs which deserve special consideration.

Hamiltonian Graph: A graph which contains a **Hamiltonian cycle**, i.e. a cycle which includes all the vertices, is said to be **Hamiltonian**.

There are some special categories of graphs which are known to be Hamiltonian (e.g. the regular polyhedral graphs are all Hamiltonian) but as mentioned earlier, the problem of determining whether an arbitrary graph is Hamiltonian is a difficult and no efficient solution is currently known. Things are a bit different for the analogously defined Eulerian graphs however.

Eulerian Graph: A trail which includes all of the edges of a graph and visits every vertex is called an **Eulerian Tour**. If a graph contains an Eulerian tour which is a circuit, i.e. an **Eulerian circuit**, the graph is simply said to be **Eulerian** (this is just a fancy word that means “the graphs that guy Euler was always going on about” – mathematics abounds with such terminology. It would sound much less impressive if his name had been Fred).

The first major theorem we will prove was the first problem in graph theory that Euler solved (in one direction, the converse was proven much later) and relates to the Königsberg bridge problem mentioned above. We will prove this theorem for plain vanilla graphs, but the proof works just as well for multi-graphs and graphs with self-loops. Note that an ample supply of jelly beans is required before embarking on this proof.

Theorem: A graph is Eulerian if and only if it is a connected graph in which every vertex has even degree.

Proof Part 1) Suppose a graph is Eulerian, then it is connected and it contains an Eulerian circuit which begins and ends some vertex v . Place a jelly bean on v as you leave it and begin walking along the Eulerian circuit, placing one jelly bean on each vertex when you reach it and another when you leave it. Continue until you have walked the entire tour at which point you will have arrived back at v . At this point every vertex in the graph has an even number of jelly beans on it, since it was visited at least once during the tour and each visit resulted in a jelly bean being placed both on arrival and departure (including the starting vertex v where we departed first and arrived last). Each arrival and departure occurred along an edge incident to the vertex, and every edge in the graph was traversed exactly once, so the number of jelly beans on each vertex is equal to its degree. Therefore every vertex has even degree.

Proof Part 2) We will prove the second half by induction on the number of edges in the graph. First note that C_3 is Eulerian, this will be our base case. Now suppose G is a connected graph in which every vertex has even degree with $n > 3$ edges, and assume that every connected graph in which every vertex has even degree and which has less than n edges is Eulerian (note that such a graph will have at least 3 edges. Pick any vertex in the graph. It must have positive degree since the graph is connected, so pick an edge and walk along a tour through the graph. Note that since every vertex has even degree, each new vertex you reached has a new edge which you have not yet traversed which you can leave by until you reach a vertex that you have already visited (the graph is finite so this must eventually happen). Once you reach a vertex you have visited before you will have walked a cycle C . Removing this cycle from the graph will leave a possibly disconnected graph, but every vertex will still have even degree. Every component of this graph will have fewer edges than the original graph and will be a connected graph in which every vertex has even degree. By the inductive hypothesis, all the components are Eulerian, so each contains an Eulerian circuit. These circuits can be combined with the cycle C to form an Eulerian circuit of the original graph. First walk the Eulerian circuit for the component containing the starting vertex, then proceed along the cycle and each time you come to a vertex in a new component (if any) walk the Eulerian circuit for that component before proceeding to the next vertex in the cycle.

QED

Corollary: A graph contains an Eulerian tour if and only if it is a connected graph with at most two vertices of odd degree.

Proof: First note that a graph cannot have only one vertex with odd degree since the sum of the degrees must be even, and that if it has no vertices with odd degree, the theorem above applies, so the only case to consider is two vertices with odd degree. Suppose the graph contains an Eulerian tour, then applying the same argument as above, every vertex other than the starting and ending vertex must have even degree so these must be the two vertices with odd degree. Conversely, If the graph has two vertices with odd degree, we can add one new vertex and two new edges connecting this vertex to the two vertices with odd degree, to create a new graph which is connected and has every vertex with even degree. By the theorem above this new graph is Eulerian so it contains an Eulerian circuit. If we now remove the vertex and edges we added, the remaining part of the Eulerian circuit is an Eulerian tour.

QED

This theorem has a surprising number of applications. We will give one example here, a number of others are explored on the homework problems.

Problem: In a set of 21 dominoes where each end of a domino is labeled with 1, 2, 3, 4, 5, or 6 dots and every combination is represented exactly one, is it possible to construct a cycle containing all 21 dominos where the ends of every pair of adjacent dominoes match?

Solution: Consider the graph K_6 with the vertices labeled 1, 2, 3, 4, 5, 6. Each edge in this graph corresponds to a domino with different ends. If we ignore the doubles for the moment, any line of dominos with matching ends corresponds to a trail in K_6 . Every vertex in K_6 has degree 5, so there are 6 nodes in this graph with odd degree, hence it is not Eulerian. This means that no cycle of the dominos with different ends can be constructed. Adding in the doubles doesn't help since if we could make a valid cycle which included the doubles, we could remove the doubles and still have a valid cycle since the dominos on either side of a double must have matching ends. Therefore no cycle which uses all the dominoes exists.

In terms of the graph, adding in the double is equivalent to adding a self-loop to each node which increases the degree of all the nodes to 7 (self-loops add two to the degree) which is still odd. This fact is true in general, and makes it clear that Theorem 3 applies to graphs with self-loops as well.