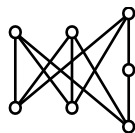


Lecture 21 – May 20

In the last lecture we saw that K_5 and $K_{3,3}$ are not planar. Clearly any graph which contains K_5 or $K_{3,3}$ as a subgraph cannot be planar (since any subgraph of a planar graph must be planar). But that is not the only possibility for ruling out planar graphs, for instance, consider the graph shown below.



This graph is found by taking $K_{3,3}$ and subdividing an edge into two edges. The resulting graph is not $K_{3,3}$ but we see that it cannot be planar, because if it were we could replace the subdivided edge by a new single edge to recover a way to draw $K_{3,3}$ in the plane. So it is not just the graph $K_{3,3}$ but rather any graph sharing similar structure (or in topology homeomorphic).

We say that a graph G contains a *topological* K_5 or $K_{3,3}$ if starting with G we can do three operations and form K_5 or $K_{3,3}$, namely:

1. Remove an edge.
2. Remove a vertex and any edge incident to the vertex.
3. Contract an edge to a single vertex. That is if $u \sim v$ then we remove the edge joining u and v and combine the two vertices u and v into a new vertex uv , this new vertex is adjacent to any vertex that was previously adjacent to either u or v .

The first two operations correspond to what it takes to form a subgraph, so it is this last operation that is of the most interest to us. This is the one that allows for us to take the graph $K_{3,3}$ with a subdivided edge and contract one of the two edges to form $K_{3,3}$.

It is not hard to see that if a graph contains a topological K_5 or $K_{3,3}$ that it cannot be planar (i.e., if it were we could use it to embed K_5 or $K_{3,3}$). Amazingly this turns out to be the only situation that we need to avoid.

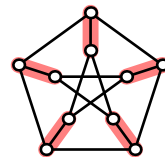
Kuratowski's Theorem:

A graph G is planar if and only if it does not contain a topological K_5 or $K_{3,3}$.

Example: Show that the Petersen graph is not planar.

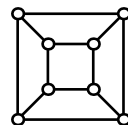
Solution: To show that the Petersen graph is not planar we must find a topological K_5 or $K_{3,3}$ inside of it. Looking at the Petersen graph it is easy to see that if we contract the edges between the outer pentagon and

the inner star (below highlighted in red) that the resulting graph is K_5 . Therefore it contains a topological K_5 and so is not planar.



We now give one final application of Euler's formula. A *Platonic* solid is a polyhedra (equivalent of a three dimensional polygon) where each face is composed of a regular n -gon (an n sided polygon with each side and each interior angle equal) and at every corner there are exactly m faces that meet. Two examples of Platonic solids are the tetrahedron (faces are triangles and at every corner three triangles meet) and cubes (faces are squares and at every corner three squares meet). Our goal is to determine how many Platonic solids there are.

The first thing we need to do is to relate this to planar graphs. Imagine that we are holding a polyhedra in our hand that is made out of a very stretchable rubber. Then we can puncture a small hole in one of the faces of the polyhedra and start pulling it apart, stretching it out until we have flattened out the rubber. Then the corners and edges of the polyhedra form a planar graph with the corners becoming vertices and the edges becoming, well, edges. The faces of the polyhedra become the faces of the planar graph. For example, if we had started with a cube we would get the following graph.



(This connection between planar graphs and polyhedra is one reason that planar graphs are interesting to study.)

So let us now translate the conditions to be a Platonic solid into conditions on the corresponding planar graph. Each face being a regular n -gon means that each face is bounded by exactly n edges. As before we can double count edges by adding up the number of edges in each face, and so we have

$$2|E| = n|F| \quad \text{or} \quad |F| = \frac{2}{n}|E|.$$

The requirement that m faces meet at each corner tells us that at every vertex we have exactly m edges coming in. So by the handshake theorem we have

$$2|E| = m|V| \quad \text{or} \quad |V| = \frac{2}{m}|E|.$$

Putting this into Euler's formula we have

$$\frac{2}{m}|E| - |E| + \frac{2}{n}|E| = 2$$

or dividing everything by $2|E|$,

$$\frac{1}{m} + \frac{1}{n} - \frac{1}{2} = \frac{1}{|E|} > 0.$$

So in order to be a Platonic solid we need to have

$$\frac{1}{m} + \frac{1}{n} > \frac{1}{2},$$

and we must also have that $m, n \geq 3$. This only gives us five possibilities.

n	m	Platonic solid
3	3	Tetrahedron
3	4	Octahedron
3	5	Icosahedron
4	3	Cube
5	3	Dodecahedron

An octahedron has eight triangular faces, an icosahedron has twenty triangular faces and a dodecahedron has twelve pentagonal faces. Therefore the number of faces on Platonic solids are 4, 6, 8, 12, 20. For people familiar with role playing you might notice that these numbers are on the most common type of dice that are available.

Finally, let us prove the following theorem.

Mantel's Theorem:

If a simple graph on $2n$ vertices contains $n^2 + 1$ edges, then G must contain a triangle.

To contain a triangle means that there are three vertices u, v, w so that $u \sim v \sim w \sim u$ (i.e., the three vertices form a triangle). Note that this result is the best possible since the graph $K_{n,n}$ has $2n$ vertices and n^2 edges but no triangle (since a triangle would give a cycle of odd length which we know cannot happen in a bipartite graph). Also not how general this is, we are saying that no matter *how* you draw the graph if it has $2n$ vertices and $n^2 + 1$ edges it must contain a triangle (and in fact contain many triangles).

To prove this we will make use of one of the fundamental principles in combinatorics.

Pigeon hole principle:

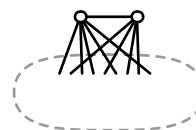
If you distribute r balls among n bins and $r > n$ then some bin has more than one ball.

The proof is by contradiction, suppose not then the total number of balls is less than the number of bins, which by assumption is impossible. This principle seems so obvious that it is amazing that it can be

used to prove anything, but in fact it is used in many beautiful proofs, including the proof we are about to give for Mantel's theorem.

We proceed by induction. For $n = 1$ it is vacuously true since there is no graph on 2 vertices with 2 edges. (This is enough to get our induction started, but suppose that we are not comfortable with this case, then for $n = 2$ it is also true since the only graph on 4 vertices with 5 edges is found by taking an edge out of a K_4 and it is easy to check that this graph has two triangles.)

Now let us suppose that we have shown that any graph on $2n$ vertices with $n^2 + 1$ edges has a triangle. Consider now a graph G on $2(n + 1) = 2n + 2$ vertices with $(n + 1)^2 + 1 = n^2 + 2n + 2$ edges. Let u and v be two vertices in G joined by an edge. Then we can draw G in the following way.



Namely as two vertices connected to an edge and then connected to the rest of the graph. Consider the graph on the vertices other than u and v , this has $2n$ vertices and so if there are $n^2 + 1$ edges in this part of the graph then by the induction hypothesis it must contain a triangle, and we are done. So we may now assume that there are $\leq n^2$ edges in the part of the graph not containing u and v . Since there is 1 edge connecting u and v that says that there are $\geq 2n + 1$ edges between the vertices u, v and the other $2n$ vertices. So by the pigeon hole principle there must be some vertex that is connected by two edges to the vertices u and v , i.e., there is a vertex w so that $w \sim u$ and $w \sim v$, but since we already have $u \sim v$ then we have our triangle and this concludes the proof.

Lecture 22 – May 22

Before we return to graph theory let us look at a nice application of the pigeon hole principle.

Let $b_1 b_2 \dots b_n$ be a rearrangement of $1, 2, \dots, n$, so for example when $n = 7$ one such rearrangement is 3521476. A subsequence of this rearrangement is $b_{i_1} b_{i_2} \dots b_{i_k}$ where $i_1 < i_2 < \dots < i_k$, i.e., we pick some elements of the rearrangement and still preserve their order. Some examples of subsequences in the arrangement for $n = 7$ given above include 3146, 5247, 35, 246 and so on. A subsequence is said to be increasing if $b_{i_1} < b_{i_2} < \dots < b_{i_k}$ (for example 146 in the above) while a subsequence is said to be decreasing if $b_{i_1} > b_{i_2} > \dots > b_{i_k}$ (for example 521 in the above).

For every rearrangement of $1, 2, \dots, n^2 + 1$ there is *always* an increasing subsequence of length $n + 1$ or a decreasing subsequence of length $n + 1$.

This result is the best possible since if we rearrange $1, 2, \dots, n^2$ it is possible to have no increasing or decreasing subsequence of length $n + 1$. For example for $n = 9$ the (only) two sequences that do this are

321654987 and 789456123.

(We divide n^2 into n blocks of n then we either reverse each block and put them in order, or reverse the order of the blocks.)

Obviously we are going to use the pigeon hole principle to prove the result. The idea behind proofs which use the pigeon hole principle is to show that there is some meaningful description of the objects (bins) so that the number of objects (balls) is more than the number of ways to describe them and so two of the objects must satisfy the same description. We then use this fact to either derive a contradiction or construct a desired object.

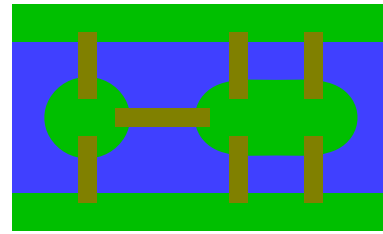
Suppose we have a rearrangement of $1, 2, \dots, n^2 + 1$. Then for each $i = 1, 2, \dots, n^2 + 1$ we associate a pair (I_i, D_i) where I_i is the length of the longest increasing subsequence that ends at b_i and D_i is the length of the longest decreasing subsequence that ends at b_i . so for example for the rearrangement 3521476 we get the following pairs.

$(1,1)$	$(2,1)$	$(1,2)$	$(1,3)$	$(2,2)$	$(3,1)$	$(3,2)$
3	5	2	1	4	7	6

Note that all of the pairs listed above are *distinct*. To see this suppose that $i < j$ then if $b_i < b_j$ we can take an increasing sequence ending at b_i and tack on b_j to get a *longer* increasing sequence ending at b_j showing $I_i < I_j$, similarly if $b_i > b_j$ then $D_i < D_j$. In any case we have that if $i \neq j$ then $(I_i, D_i) \neq (I_j, D_j)$.

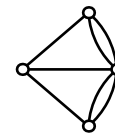
If there is no increasing and decreasing sequence of length $n + 1$ or more than we must have that $1 \leq I_i, D_i \leq n$, i.e., there are at most n^2 possible (I_i, D_i) . On the other hand there are $n^2 + 1$ points and so there are $n^2 + 1$ pairs (I_i, D_i) , so that by the pigeon hole principle two would have to be the same. This is impossible! And so we can conclude that there must be an increasing or decreasing sequence of length $n + 1$ (or more).

We now return to graph theory. We start with the first major result in graph theory given by Euler in 1736. It is based on the following story. In Königsberg (now called Kaliningrad) there was a large river that ran through the city and in the middle of the river were two large islands. Between the islands and the shore were seven bridges arranged as below.



As the story goes it was fashionable on Sunday evenings for the citizens of Königsberg to dress in their finest and walk about the city. People tried to find a route where they would return to the place they started at and cross each bridge exactly once. Try as they might people could not figure a route that would accomplish this.

Euler learned of the problem and he approached it by first translating it into a question about graph theory. He let each piece of land be a vertex and each bridge an edge connecting two vertices. This gives us the following multi-graph (a multi-graph because it has two edges between two vertices).



Translating this into graph theory we are looking for a cycle in the graph (i.e., we start at a vertex and move along edges and end at the vertex we started at) which visits each vertex at least once and uses each edge *exactly* once (the main condition is that we are using each edge exactly once). Such a cycle is called an *Euler cycle*. Euler made two observations, first if the graph is to have an Euler cycle then it must be connected because we can get from any vertex to any other vertex by using the cycle. The second observation is that the degree of each vertex must be even. To see this if we look at each vertex we will at some point come in and then we will leave via a different edge, the idea here is that we can pair up the edge we came in with the edge that we came out and we do this every time we return to the vertex.

Now looking at the graph above it is easy to see that it is impossible to have an Euler cycle because the degrees of the graph are 3, 3, 3, 5, not even one even degree!

So we now know when a graph does not have an Euler cycle (i.e., it is not connected or has odd degree vertices), but when does a graph have an Euler cycle? Amazingly the necessary conditions are also sufficient conditions.

A graph has an Euler cycle if and only if it is connected and every vertex has even degree.

We have already seen that if a graph has an Euler cycle that it is connected and every vertex has even

degree. So we need to go in the opposite direction, i.e., we need to show that if the graph is connected and every vertex has even degree that it has an Euler cycle. We do this by giving a construction that will make an Euler cycle and show that the requirements for the construction are precisely those that the graph is connected and every vertex has even degree.

We start with our graph and pick a vertex v . We start by forming a cycle by going out along an edge and then once we get to a new vertex we go along a previously unused edge. (The important part is that we will always use a previously unused edge.) We keep doing this until we get to a vertex that has no previously unused edge. Since every vertex has even degree every time we come into a vertex we can exit along a different edge (i.e., we can pair edges up) the only vertex where this is not true is the vertex that we started at and so we now have a cycle where each edge is used at most once.

If we have used all of the edges then we are done. So suppose that we have not used all of the edges. Since the graph is connected there must be some vertex w in the cycle that is incident to a previously unused edge. We start the process again at this vertex and create another cycle that uses each edge at most once. We now glue the two cycles together, i.e., go from v to w along part of the first cycle, use the second cycle to get from w to w and then finally use the remaining part of the first cycle to get from w to v . Thus we have a longer cycle where each edge is used at most once. We continue this process until we create a cycle which uses each edge exactly once, our desired Euler cycle.

Related to Euler cycles are Euler paths, the only difference being that we do not return to the vertex that we started at (i.e., we form a path, not a cycle). It is easy to see that we have the following condition.

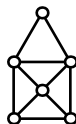
A graph has an Euler path if and only if it is connected and exactly two vertices have odd degree.

Note that the Euler path must start and end at the vertices with odd degree.

Example: Can the following figure be drawn without lifting a pencil or crossing each edge twice?



Solution: Making this a graph by putting a vertex where any edges come together we get the following.



In particular we see that the degrees of this graph are 2, 3, 3, 4, 4, 4. Since it has exactly two vertices with odd degree this graph has an Euler path which would correspond to a drawing of the graph where we draw each edge exactly once and we never lift our pencil off the paper. We can even see that if we wanted to be able to draw it we must start at either the lower left or the lower right corner (i.e., the vertices with odd degree).

Lecture 23 – May 27

There is a directed analog for Eulerian cycles. Recall that in a directed graph each edge has an orientation which we marked with an arrow. So an Eulerian cycle for a directed graph is one that uses each edge exactly once, visits each vertex and also on each edge goes in the direction indicated by the orientation. For the undirected graphs we used the fact that every time we came in we also had to leave so that we paired up edges and so the degree at each vertex had to be even. For directed graphs we have the same situation except now we have that when we pair edges up we pair an edge going *in to* the vertex with one going *away from* the vertex. This gives us the directed analog for Eulerian cycles.

A directed graph has an Euler cycle if and only if it is connected and at every vertex the number of edges directed into the vertex equals the number of edges directed away from the vertex.

We can use the directed Euler cycles to construct *de Bruijn* sequences. A de Bruijn sequence is a way to pack all 2^n binary words of length n into a single binary word of length 2^n where each binary word of length n occurs exactly once as n consecutive digits (where we also allow for wrapping around). For example for $n = 1$ we have the binary words 0 and 1 so a de Bruijn sequence in this case is 01. For $n = 2$ we have the binary words 00, 01, 10 and 11 and it can be checked that 0011 is a de Bruijn sequence. For $n = 3$ there are two possible de Bruijn sequences 00011101 and 00010111, below we check that the first one is a de Bruijn sequence (remember that we allow for wrap around!).

word	location
000	<u>000</u> 11101
001	00 <u>01</u> 1101
010	0001 <u>101</u>
011	0001 <u>110</u> 1
100	0001 <u>110</u> 1
101	0001 <u>110</u> 1
110	0001 <u>110</u> 1
111	000 <u>111</u> 01

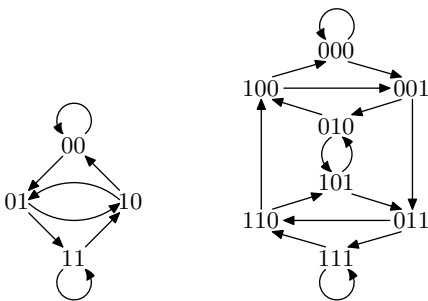
The first question that should be asked is do such sequences always exist, and if so how to construct them. They always exist, and the method to construct them involves directed Eulerian cycles (which conveniently we just discussed!).

To make a de Bruijn sequence for binary words of length n , we define a directed graph \mathcal{D}_n . The vertices of \mathcal{D}_n are binary words of length $n - 1$ (so that there are 2^{n-1} vertices) and we have directed edges going from

$$a_1 a_2 a_3 \dots a_{n-1} \rightarrow a_2 a_3 \dots a_{n-1} a_n.$$

In other words we put a directed edge if the last $n - 2$ digits of the first vertex matches the first $n - 2$ digits of the second vertex. (Note in particular that each edge is associated with a binary word on n letters and each binary word on n letters is associated with a unique edge. So de Bruijn sequences then say that we need to use all of the edges.)

The graphs \mathcal{D}_3 and \mathcal{D}_4 are shown below.



We now make some observations about this graph.

- The graph \mathcal{D}_n is connected. To see this we have to describe a way to get from any vertex to any other vertex. Suppose that we have vertices $a_1 a_2 a_3 \dots a_{n-2} a_{n-1}$ and $b_1 b_2 b_3 \dots b_{n-2} b_{n-1}$, then we can use the following edges to connect the vertices:

$$\begin{aligned} a_1 a_2 a_3 \dots a_{n-2} a_{n-1} &\rightarrow a_2 a_3 \dots a_{n-2} a_{n-1} b_1 \\ a_2 a_3 \dots a_{n-2} a_{n-1} b_1 &\rightarrow a_3 \dots a_{n-2} a_{n-1} b_1 b_2 \\ &\dots \rightarrow \dots \\ a_{n-1} b_1 b_2 b_3 \dots b_{n-2} &\rightarrow b_1 b_2 b_3 \dots b_{n-2} b_{n-1} \end{aligned}$$

Basically at each step we take off one term from the first vertex and add on a term from the second vertex. So after $n - 1$ steps we can get from any vertex to any other vertex so the graph is connected.

- The number of edges going into a vertex is equal to the number of edges going out of a vertex. More precisely there are 2 edges coming in and 2 edges going out. For a vertex $a_1 a_2 \dots a_{n-2} a_{n-1}$ the edges coming in are

$$\begin{aligned} 0 a_1 a_2 \dots a_{n-2} &\rightarrow a_1 a_2 \dots a_{n-2} a_{n-1} \\ 1 a_1 a_2 \dots a_{n-2} &\rightarrow a_1 a_2 \dots a_{n-2} a_{n-1} \end{aligned}$$

while the edges going out are

$$\begin{aligned} a_1 a_2 \dots a_{n-2} a_{n-1} &\rightarrow a_2 \dots a_{n-2} a_{n-1} 0 \\ a_1 a_2 \dots a_{n-2} a_{n-1} &\rightarrow a_2 \dots a_{n-2} a_{n-1} 1 \end{aligned}$$

By the above two items we see that the graph \mathcal{D}_n has an Euler cycle (in fact it can be shown that it has $2^{2^{n-1}-n}$ Euler cycles). The last thing to note is that every Euler cycle corresponds to a de Bruijn sequence (and vice-versa). For example in the graph \mathcal{D}_3 it is easy to find an Euler cycle

$$00 \rightarrow 00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 01 \rightarrow 10 \rightarrow 00.$$

To turn it into a de Bruijn sequence we take each edge and record the letter that it adds onto the end of the word. Doing that for the above sequence we get 01110100 which is a de Bruijn sequence (note that this is the same as the de Bruijn sequence listed earlier where we have shifted the entries, but since we are dealing with an Euler cycle shifting the entries only corresponds to changing the starting location of our Euler cycle, it is still the same cycle. The way to see that this works is that every edge is coded by a word of length n , namely the first $n - 1$ digits tells us the location of the originating vertex and the last $n - 1$ digits tells us the location of the terminal vertex. Looking at a term and the previous $n - 1$ terms (where we cycle back) tells us the edge we are currently on. Since we use each edge exactly once each binary word of length n must show up exactly once.

de Bruijn sequences have an interesting application as part of a magic trick (I learned this from Ron Graham and Persi Diaconis who are writing a book about magic and mathematics). Namely a magician is performing a magic show and he throws a deck of cards out into the back of the audience. The person who catches it is told to cut the deck (which is to split the deck in half and then put the top half on the bottom) then hand it to the person to their right, this is repeated until five people have cut the deck. (This is to show that there is no one in cahoots with the magician, i.e., the magician now has no idea what card is on top.) He now has the fifth person take off the top card, and hand it back to the fourth person who again takes off the top card and hands it back to the third person and so on until the last person.

He now looks at the five people holding cards and he tells them he is going to read their minds and tell them which card they have and asks them to each focus on the card that they have. He stares intensely for a few seconds and then declares "I am having a little trouble receiving your mental signals, I have discovered that sometimes the signals from the red cards overpower the black cards, could everyone holding a red card please sit down." At this point anyone holding a red card sits down. He continues "Ah, much better. I now can see

the cards clearly and you have a . . .” at which point he lists everyones card.

So what happened. Well one possibility is that the magician can actually read minds, but there is an easier explanation. Noticed that he asked for people to sit down, this was a way to send a signal to him about the order of the black and red cards. The magician actually planned ahead and did not toss out a random deck but instead had a deck where he had arranged 32 cards (note that $2^5 = 32$) in a specific order so that any five consecutive cards had a unique pattern of red and black cards (this is easily done using a de Bruijn sequence for $n = 5$). The fact that he had them cut cards did not change the de Bruijn sequence, the only important thing was that he had five consecutive cards and he was able to determine the order of the red and black cards. Once he knew this he knew where he was in the de Bruijn sequence and he could then say which cards they were by memorizing the locations (or having a cheat sheet to look off).

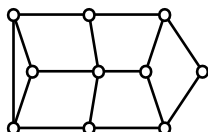
The idea behind Euler cycles is that we have a cycle in the graph which uses each *edge* exactly once. We can ask a similar question about a cycle which uses each *vertex* exactly once. Such a cycle is called a *Hamilton cycle* after a mathematician who studied these cycles (and also invented a related game about trying to find a way to visit twenty different cities so that each city is visited exactly once).

While it is easy to characterize graphs which have Euler cycles, it is difficult to characterize graphs which have Hamilton cycles (also known as hamiltonian graphs). There are some clearly necessary conditions, i.e., the graph must be connected and it cannot have any bridges, on the other hand there are also some sufficient conditions. But there are no set of conditions which are necessary and sufficient that work for all graphs. An example of a sufficient condition is Dirac’s Theorem (we will not prove it but its proof can be found in most graph theory textbooks).

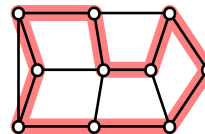
Dirac’s Theorem
 If G is a graph on $n \geq 3$ vertices and the degree of each vertex is $\geq n/2$ then G is hamiltonian.

In general to show a graph is hamiltonian we only need to find a cycle that uses each vertex exactly once. To show a graph is not hamiltonian we need to show that it is impossible to find a cycle that uses each vertex exactly once (in this case it pays to be clever, i.e., to avoid looking at every possible cycle).

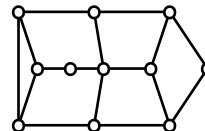
Example: Show that the graph below is hamiltonian.



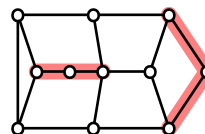
Solution: We only need to find a hamiltonian cycle. This is not too hard and one possibility is shown below.



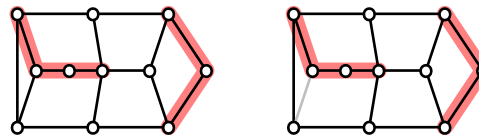
Example: Show that the following graph does not have a hamiltonian cycle.



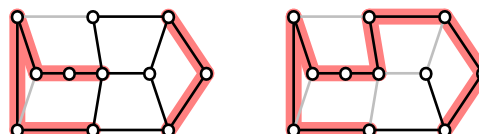
Solution: Note that this is almost the same graph as before, the only difference is that now we have taken an edge and split it into two edges, but apparently even this small change is enough to force us to not be hamiltonian. Let us make an observation, if the graph has a hamiltonian path then any vertex with degree two must have both edges in the cycle. So looking at our graph any hamiltonian cycle must have the following four edges.



By symmetry we can assume that for the vertex in the middle on the left it connects to the vertex in the top on the left, as shown below on the left.



Once two edges incident to a vertex are used in a hamiltonian path any remaining incident edges to that vertex cannot be used so we can remove it from further consideration (as shown above on the right). Using this observation we get the following graphs.



Looking at the graph on the right it is easy to see that no matter which of the remaining edges we use we will run into a problem. In particular we cannot

close up and form a hamiltonian cycle. (We can make a path that uses all the vertices exactly once, such a path is called a hamiltonian path.)

Lecture 24 – May 29

A special case of hamiltonian cycles are hamiltonian cycles on the n -dimensional hypercube Q_n . These are known as Gray codes (named after Frank Gray, an engineer at Bell Labs). Because of their connection to binary words they are studied in computer science. They also are useful if we want to look through all possible combination of using or not using some collection of objects since the edges in the hypercube only change in one digit so this corresponds to changing the state of just one object at each step.

We now show how to construct a hamiltonian cycle for Q_n . First note that this holds for $n = 1$ since we have $0 \rightarrow 1 \rightarrow 0$ (in this case we will allow ourselves to go back over the edge we used). For $n = 2$ we have essentially only one cycle (that is we can assume that we start at $00 \cdots 0$ and that we pick some direction), namely

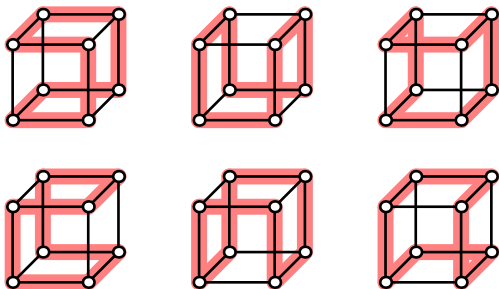
$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00.$$

In general if we have a hamiltonian cycle for Q_n , call it R , then to construct a hamiltonian cycle for Q_{n+1} we look at $0R$ and $1\bar{R}$, i.e., we put a 0 in front of everything in R (the $0R$) and then we put a 1 in front of everything in R where we run in reverse order (the $1\bar{R}$). For example, using the above Gray code for $n = 2$ we get the following Gray code for $n = 3$

$$\underbrace{000 \rightarrow 001 \rightarrow 011 \rightarrow 010}_{0R} \rightarrow \underbrace{110 \rightarrow 111 \rightarrow 101 \rightarrow 100}_{1\bar{R}} \rightarrow 000.$$

It is easy to see that this will visit all vertices and that each two consecutive entries are adjacent and so we have a desired hamiltonian cycle.

Of course this is not the only way to form a Gray code. In general there are many ways that we can go about forming a Gray code and we can choose different Gray codes which have different desired properties. One natural question to ask is how many are there, for $n = 3$ there are essentially 6 different Gray codes and these are shown below.

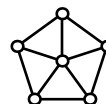


For $n = 4$ there are 1,344 different Gray codes and for $n = 5$ there are 906,545,760 different Gray codes (we will not draw them here). For $n = 6$ only an approximate number is known.

We now turn to coloring graphs. There are essentially two things that can be colored in a graph, vertices and edges. We will restrict ourselves mostly to the case of coloring the vertices (note that coloring edges is usually equivalent to coloring the vertices of the line graph). So a coloring is an assignment of colors, usually listed as $1, 2, \dots, k$, to the vertices. Equivalently, a coloring is a function $\phi : V \rightarrow \{1, 2, \dots, k\}$.

Our definition of coloring is very general so most of the time we will restrict our colorings so that they are *proper colorings*, which are colorings of a graph where no two adjacent vertices have the same color. The minimum number of colors (i.e., smallest value of k) needed to have a proper coloring for a graph G is called the *chromatic number* of the graph and is denoted $\chi(G)$.

Example: Find the chromatic number for K_n , bipartite graphs, C_n and the graph shown below.



Solution: For the complete graph since every vertex is adjacent to every other vertex they must all have different colors and so $\chi(K_n) = n$. We know a graph is bipartite if we can split the vertices into two sets U and W so that all edges go between U and W , in particular if we color all the vertices in U blue and all the vertices in W red then we have a proper coloring, so a bipartite graph only needs two colors. This gives another characterization of bipartite graphs.

A graph G is bipartite if and only if $\chi(G) = 2$.

For the graph C_n we note that if n is even that we have a bipartite graph and so we only need two colors. If n is odd then the graph is not bipartite so we need more than two colors, it is not hard to see that we can use only three, namely we go around and alternate red-blue-red-blue... and then when we get to the last vertex we see that it can neither be red or blue so we color it using a third color. In particular we have

$$\chi(C_n) = \begin{cases} 2 & \text{if } n \text{ is even,} \\ 3 & \text{if } n \text{ is odd.} \end{cases}$$

Finally for the last graph we note that it contains the graph C_5 on the outer edge. Just to color those vertices we need at least three colors, then finally the central vertex is adjacent to all the other vertices so it must

be a different color than those used on the outer cycle so we need at least four colors, and it is not hard to color the vertices of the graph using four colors so the chromatic number is 4.

The chromatic number has some interesting applications. If we look at a graph with a proper coloring then grouping the vertices together with the same color (known as the color classes) we see that there are no edges between red vertices, no edges between blue vertices, and so on. In graph terminology that says that the vertices in each color class form an independent set (a set where no two vertices are adjacent). So finding the chromatic number of a graph is the same as finding the minimum number of sets that we can group vertices into so in each set no two vertices are adjacent.

For example, suppose that we want to set up a schedule for committee meetings. We want to have as few meeting times as possible but there are some people who serve on more than one committee. In the latter case any two committees with more than one person serving on them cannot meet at the same time. Let us represent our situation using a graph where each vertex is a committee and we draw an edge between any two committees that share a member (i.e., committees that cannot meet simultaneously). Then the chromatic number of the graph is the *minimal* number of committee meeting times needed so that each committee can meet once. Further, each color class tells us what committees should be meeting simultaneously.

As another example, suppose that we want to store chemicals in warehouses. Some chemicals interact and so must be stored in separate warehouses while other chemicals do not and can safely be stored together. Again let us represent the situation using a graph where each vertex is a chemical and we draw an edge between two chemicals if they interact. Then the chromatic number is the *minimal* number of warehouses needed to safely store all the chemicals. Further, each color class tells us what chemicals should be stored together to achieve the minimum.

Given that the chromatic number is a useful property of a graph we would like to have a way to find it. This is a nontrivial problem in general, but there are some easy estimates (though generally speaking these are quite poor).

Let $\Delta(G)$ be the maximum degree of a vertex in a graph G . Then $\chi(G) \leq \Delta(G) + 1$.

The proof of this is very easy. We start coloring and we color greedily in that at every vertex we use the smallest color not used on any vertex adjacent to the current vertex. Since each vertex is adjacent to at most $\Delta(G)$ other vertices, i.e., so it is adjacent to at most $\Delta(G) + 1$ other colors, we only need to use $\Delta(G) + 1$

colors to make a proper coloring of the graph. This Bound is far from best possible and it can be shown that the only graphs for which $\chi(G) = \Delta(G) + 1$ are K_n and odd cycles (C_n with n odd).

To get a lower bound we first make the following observation.

If H is a subgraph of G then $\chi(H) \leq \chi(G)$.

This is easy to see since any proper coloring of G automatically gives a proper coloring of H . So H would never need more colors than it takes to color G (and depending on the graph can sometimes use far fewer). Clearly one type of graph which has a high chromatic number are complete graphs, so one way to look for a lower bound for the chromatic number is to look for large complete graphs inside the graph we are trying to color. This gives us the following result.

Let $\omega(G)$ denote the clique number of the graph G , i.e., the size of the largest complete graph that is a subgraph of G . Then $\chi(G) \geq \omega(G)$.

Looking at the last result it is tempting to think that the chromatic number of a graph is somehow a “local” property. In other words the main force in giving us a high chromatic number is to have a large clique. This is not the case, and in fact the opposite counter-intuitive fact is true. Namely, that there are graphs which have high chromatic number but “locally” look like trees (which are simple bipartite graphs we will talk about in the next lecture). More precisely, let the girth of a graph to be the length of the smallest cycle without repeated edges.

Theorem (Erdős):

For any k there exists graphs with $\chi(G) \geq k$ and girth at least k .

The most famous problem in graph theory (which became the driving force of graph theory for more than a century) is the four color problem. This dates back to the 1850’s when someone noticed that when coloring the counties of England that they only needed four colors so that no two adjacent counties had the same color. (Adjacent meant that they shared some positive length of distance. For instance in the United States the four corners is a point where four states meet at a point, namely Utah, Colorado, New Mexico and Arizona we would not consider Arizona and Colorado adjacent. We also make the basic assumption that counties are contiguous.) The question then became is this always possible for any map.

We can relate this to graph theory by putting a vertex for every county (or state, country, whatever) and connect two vertices when two counties share a border.

The resulting graph is planar, and so the question became can every planar graph be colored with four or fewer colors?

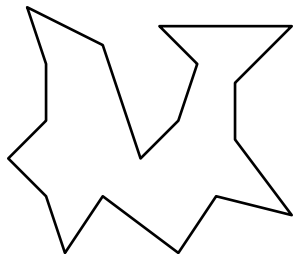
Since every planar graph has a vertex of degree at most 5 it is not hard to show by induction that the chromatic number for a planar graph is at most 6. In the late 1800's Kempe found a "proof" that the chromatic number of the planar graph was 4 but it was later shown to have a defect and was modified to show that the chromatic number was at most 5. The final proof was finally settled in the 1970's by Appel and Haken.

Theorem (Appel and Haken):
If G is a planar graph then $\chi(G) \leq 4$.

This was one of the most controversial proofs in mathematics because it heavily relied on computers and could not be verified by hand. A shorter proof (that still relied on computers) was subsequently found and most people generally accept the four color theorem as true.

Interestingly enough there are similar statements for other surfaces. For instance any graph drawn on the surface of the torus with no edges crossing can be colored in seven or fewer colors, but you can also draw K_7 on the torus so there is a graph which needs at least seven. Surprisingly this was known well before the fact that the plane (which we would expect to be the simplest case) needs at most four colors, and also the proof is much simpler and does not require computers.

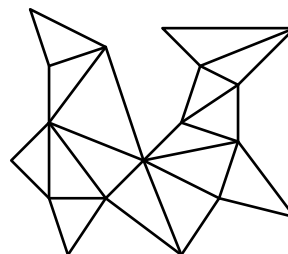
Finally, we give an application for coloring. The art gallery problem asks the following question. Suppose that we have an art gallery which is in the shape of a polygon with n sides (we will assume that there are no holes). What is the minimal number of guards which need to be positioned inside the art gallery so that the entire gallery is under surveillance. If the polygon is convex then we only need one guard who can be positioned anywhere. But the layout for the art galleries might not be arranged in a convex polygon, for example consider the gallery below which has 18 sides.



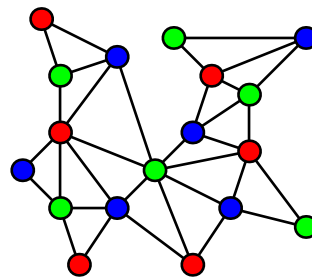
In the art gallery problem for a polygon with n sides with no holes then you need no more than $\lfloor n/3 \rfloor$ guards to protect the gallery.

The notation $\lfloor n/3 \rfloor$ means that you take $n/3$ and round down. So for instance we know that in the gallery above we need at most 6 guards. We also note that there are examples where we need at least $\lfloor n/3 \rfloor$ guards to protect the gallery (we do not give them here) so this result is the best possible.

To prove this we first take out gallery and triangulate it, i.e., add edges between the corners so that the resulting floor layout is a collection of triangles. In general there are many ways to do this (it is not immediately obvious that this can be done, it is not difficult to show, but we will omit that step). One example for the gallery shown above is the following.



We now think of this as a graph where the lines are edges and the vertices are polygons. We claim that we can color the corners in three colors so that in any triangle there is one red corner, one blue corner and one green corner. This is easily seen for the case when we start with three sided polygon. Inductively we can take an interior edge and split the gallery into two halves. By induction we know that we can color each half satisfying the requirement and then we can combine the two halves to get the desired coloring. For example for the gallery above we get (the essentially unique) desired coloring below. (Here essentially unique means that given a triangulation there is only one coloring up to permutation of colors, different triangulations lead to different colorings.)



Now we choose the color that is used least often, since there are n corners to color and we only use three colors there is some corner used at most $\lfloor n/3 \rfloor$ times. Position the guards at the vertices with the color used least often. Note that each triangle has a corner with each color and so each triangle is now under surveillance. But of course once all of the triangles are under surveillance the entire gallery is under surveillance and we are done.

In our case it turns out that based on our triangulation each color was used exactly 6 times, and so we can position guards at either the corners with the blue vertices, the corners with the red vertices, or the corners with the green vertices and protect the entire gallery.

Lecture 25 – June 1

The art gallery problem from the last lecture has a nice application.

Fary's Theorem:

Every planar graph can be drawn in the plane with no edges crossing and as all edges as straight lines.

Previously we said that a planar graph is one which can be drawn in the plane so that no two edges intersect, but we make no requirement about the shape of the edges, so they can bend and go around as needed. This says that every planar graph can be drawn so that all the edges are straight. We sketch the proof of this result here.

First we note that we can limit ourselves to the case that we are dealing with maximally planar graphs, i.e., these are graphs where the addition of any edge would make the graph nonplanar. The important thing about these graphs is that each face is a triangle (if not then we could add edges inside the face to triangulate it and still stay planar). We fix one triangle to act as the outer triangular face and we now proceed by induction to show that we can draw the rest of the graph with straight lines. Clearly, we can draw the triangle with straight lines. Suppose we have shown that we can draw a planar graph with n vertices with straight lines.

Now consider the case with $n + 1$ vertices. We know that there is some vertex of degree ≤ 5 in the graph (and in particular we can also assume it is not one of the vertices involved in the outer triangular face). Remove the vertex and the incident edges to give a graph on n vertices, and we now triangulate the face (note that the face will be either a triangle, a quadrilateral or a pentagon). By induction we can now embed the graph on n vertices in the plane with all edges as straight lines. Finally, remove the edges that we added in earlier, we now have to position the last vertex inside the face so that it can connect to all of the other edges using a straight line. This can be done since the face has at most five sides and so by the art gallery problem we know that we need at most $\lceil 5/3 \rceil = 2$ guards to keep the entire gallery under surveillance, i.e., there is some point which sees all of the corners, put the $(n+1)$ th vertex in that point and connect with a straight edge to all other vertices. We now have our desired straight line embedding for our graph on $n + 1$ vertices.

An important type of graph are *trees*. There are a few different ways to characterize a tree. A tree is a graph on n vertices which

- is connected and acyclic (no cycles); or
- is connected and has $n - 1$ edges; or
- is acyclic and has $n - 1$ edges; or
- between any two vertices there is a unique path.

There are many others as well. We note that for a graph on n vertices to be connected we need at least $n - 1$ edges so that in some sense these are the “smallest” graphs on n vertices.

Trees are frequently used to model series of decisions, where we start at a root vertex and then have several options, then for each option we might have several more options and so on until we get to the end of the process. One important example of this are binary trees where we have a root vertex and then every vertex has 0, 1 or 2 vertices coming down from that vertex (only in computer science do trees grow down), which are labeled left or right. The binary comes from the number of outcomes being at most two.

Trees get their names because they do bear a resemblance to actual trees in the sense that trees have nodes branching off into smaller branches which then branch off and (unless through some devious topiary) the tree does not come back in onto itself. An important terminology about trees are leaves. These are the vertices of degree 1 in a tree (so the end of the tree).

Every tree on $n \geq 2$ vertices has at least one leaf.

To prove this we make use of an averaging argument.

Given numbers n_1, n_2, \dots, n_k let \bar{n} denote their average. Then there is some i and j so that $n_i \leq \bar{n}$ and $n_j \geq \bar{n}$.

In other words, unlike the children in Lake Wobegon, not all the numbers can be above average. In our case let us note that since the graph is connected the degree of each vertex is at least one, i.e., $d(v) \geq 1$ for all v in the tree. By considering the average of the degrees we have that for some vertex

$$d(v) \leq \frac{\sum_{v \in V} d(v)}{n} = \frac{2(n-1)}{n} = 2 - \frac{2}{n} < 2.$$

Here we used the fact that the sum of the degrees is twice the number of edges and that in a tree there are $n - 1$ edges. So this shows that some vertex has degree less than 2, which combined with the fact that all vertices have degree at least 1 we can conclude that there is a vertex with degree exactly one, i.e., a leaf.

We can actually say a little more. In particular, if a graph has a vertex of degree m then it has at least m leaves. One way to see this is we start at the vertex of degree m we go out along each edge and we keep walking until we hit a leaf, since there are no cycles each leaf that we end up at is distinct and so we must have at least m leaves.

Alternatively, suppose that we have k leaves in the graph, so that the remaining $n - k - 1$ vertices have degree at least 2 (the k leaves and the 1 vertex of degree m). Then we have




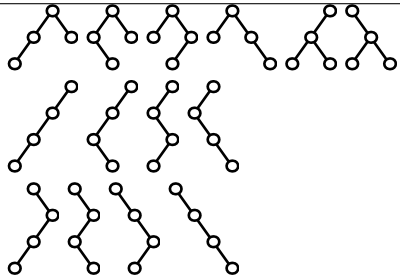
$$2(n - 1) = \sum_{v \in V} d(v) \geq k + m + 2(n - k - 1),$$

which simplifying gives $k \geq m$, the desired result.

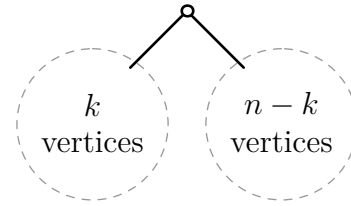
Lecture 26 – June 3

In the last lecture we discussed binary trees. These are trees with a rooted vertex and every vertex has 0, 1 or 2 children where each child is labeled left or right. (Binary here reflects that we have two options at each stage.) A natural question to ask is how many binary trees are there, before we do that we need to say that two binary trees are the same if and only if when we map the root of one tree to the other the trees and labeling (i.e., left/right) remain the same. To get an idea we can look at the first few cases (these are shown below).

In particular we see that we have 1, 1, 2, 5, 14, ... binary trees. These numbers might look familiar, and they should, they are the Catalan numbers!

n	trees	# of trees
0		1
1		1
2		2
3		5
4		14

To see why we should get the Catalan numbers we note that to make a binary tree on $n + 1$ vertices we have a root and then on the left child we have a binary tree on k vertices and on the right child we have a binary tree on $n - k$ vertices (see picture below).





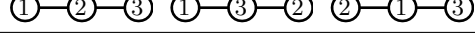
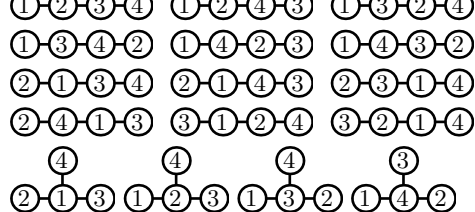
In particular if we let B_n be the number of binary trees on n vertices then we have B_k choices for the binary tree on the left and B_{n-k} choices for the binary tree on the right or $B_k B_{n-k}$ ways to make a binary tree. But then we can also let $k = 0, 1, \dots, n$ and so we add them all up to get all the possible binary trees on $n + 1$ vertices, i.e.,

$$\begin{aligned} B_{n+1} &= B_0 B_n + B_1 B_{n-1} + \dots + B_n B_0 \\ &= \sum_{k=0}^n B_k B_{n-k}. \end{aligned}$$

Note that this is an example of convolution and whenever we get a recurrence of this type it can often be related back to the Catalan numbers.

More generally we can ask how many trees on n vertices there are. We will not answer that question (since it is not trivial!), instead we will look at another problem, how many *labelled* trees on n there are. A labelled tree on n vertices is a tree where we have assigned a label to each vertex, we will use the labels $1, 2, \dots, n$. Two labeled trees are the same if and only if vertices labeled i and j are adjacent in one tree if and only if they are adjacent in the other tree for all pairs i and j .

Again we can start by listing them all for a few small values. We have the following.

n	trees	#
1		1
2		1
3		3
4		16

Certainly the first thing we notice is that there are a lot of labelled trees. For instance for $n = 5$ there would be 125 (which explains why we stopped at 4). It turns out that there is a simple expression for the number of labelled trees on n vertices.

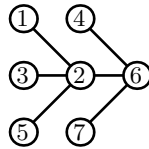
Cayley's Theorem:

There are n^{n-2} labelled trees on n vertices.

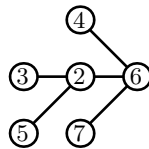
This is certainly a simple expression and the form n^{n-2} is highly suggestive. It is not hard to find something else that has the form n^{n-2} , namely consider words of length $n-2$ where each letter in the word is in $\{1, 2, \dots, n\}$. One method to prove Cayley's Theorem is to show that there is a one-to-one correspondence between labelled trees on n vertices and these words of length $n-2$. Conveniently enough there is a nice correspondence known as Prüfer Codes.

Prüfer Codes:
 Given a labelled tree on n vertices we construct a word using the following procedure. For the current tree find the *leaf* which has the lowest label, remove the leaf and write down the label of the vertex that it was adjacent to. Repeat this until there are no leaves.

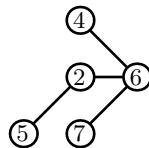
As an example, consider the following labelled tree on 7 vertices.



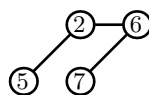
The lowest labelled leaf is 1 and it is adjacent to 2. So we remove the leaf and record 2. We now have the tree.



The lowest labelled leaf is now 3 and it is adjacent to 2. So we remove the leaf and record 2, so our word is currently 22. We now have the tree.



The lowest labelled leaf is now 4 and it is adjacent to 6. So we remove the leaf and record 6, so our word is currently 226. We now have the tree.



The lowest labelled leaf is now 5 and it is adjacent to 2. So we remove the leaf and record 2, so our word is currently 2262. We now have the tree.



The lowest labelled leaf is now 2 and it is adjacent to 6. So we remove the leaf and record 6, so our word is currently 22626. We now have the tree.



The lowest labelled leaf is now 6 and it is adjacent to 7, so we remove the leaf and record 7, so our word is currently 226267. We now have the tree.



This tree has no leaves (note that the only tree without leaves are the trees on one or fewer vertices), and so we stop. So this tree is associated with the word 226267. Now at this point we notice that this has 6 letters. In particular, this procedure produces a word of length $n-1$, and since each letter in the word is a label on a vertex each letter is one of $\{1, 2, \dots, n\}$.

We wanted to have a word of length $n-2$ not $n-1$, but let us make an observation. In the last lecture we showed that every tree on $n \geq 2$ vertices has a leaf, in fact it is easy to show that every tree on $n \geq 2$ vertices has at least two leaves. Since we will always be taking the *smallest* leaf out we will never take out n as we build our code. In particular, the last letter in the sequence of length $n-1$ must *always* be n , so we can ignore it and just work with the first $n-2$ digits.

We see that for every tree that we can make a Prüfer code, but to be complete we also need to check that there is a one-to-one correspondence. In particular given a sequence of length $n-1$ where the first $n-2$ digits are in $\{1, 2, \dots, n\}$ and the last digit is n we need to be able to construct the tree that it came from.

Deconstructing Prüfer Codes:
 Given a sequence

$$a_1 a_2 \cdots a_{n-2} a_{n-1}$$

where $a_i \in \{1, 2, \dots, n\}$ for $i \leq n-2$ and $a_{n-1} = n$ construct the sequence $b_1 b_2 \cdots b_{n-1}$ recursively by letting

$$b_i = \min \{k \mid k \notin \{b_1, \dots, b_{i-1}, a_i, \dots, a_{n-1}\}\}.$$

Then form a tree on n labeled vertices with edges joining a_i and b_i for $i = 1, 2, \dots, n-1$.

The idea is that the Prüfer code was recording the vertices adjacent to the leaf that we removed, if we knew the corresponding labels of what was removed

then we could form all the edges of the tree. So what we are doing with the b_i is determining the labels of the leaves that were removed. The idea being that b_i cannot be one of b_1, \dots, b_{i-1} since those vertices were already removed and it cannot be one of a_i, \dots, a_{n-1} since they need to stay in the graph after the current step, so then b_i must be the smallest number not on the list.

Let us work back through our example above. Applying the above procedure to the word 226267 we see that the smallest number not seen is 1 so we have $b_1 = 1$.

$$\begin{array}{c|cccccc} a_i : & 2 & 2 & 6 & 2 & 6 & 7 \\ \hline b_i : & 1 & & & & & \end{array}$$

We now cover up a_1 and see that the smallest number not seen is 3 so $b_2 = 3$.

$$\begin{array}{c|cccccc} a_i : & \cancel{2} & 2 & 6 & 2 & 6 & 7 \\ \hline b_i : & 1 & 3 & & & & \end{array}$$

We now cover up a_2 and see that the smallest number not seen is 4 so $b_3 = 4$.

$$\begin{array}{c|cccccc} a_i : & \cancel{2} & \cancel{2} & 6 & 2 & 6 & 7 \\ \hline b_i : & 1 & 3 & 4 & & & \end{array}$$

We now cover up a_3 and see that the smallest number not seen is 5 so $b_4 = 5$.

$$\begin{array}{c|cccccc} a_i : & \cancel{2} & \cancel{2} & \cancel{6} & 2 & 6 & 7 \\ \hline b_i : & 1 & 3 & 4 & 5 & & \end{array}$$

We now cover up a_4 and see that the smallest number not seen is 2 so $b_5 = 2$.

$$\begin{array}{c|cccccc} a_i : & \cancel{2} & \cancel{2} & \cancel{6} & \cancel{2} & 6 & 7 \\ \hline b_i : & 1 & 3 & 4 & 5 & 2 & \end{array}$$

We now cover up a_5 and see that the smallest number not seen is 6 so $b_6 = 6$.

$$\begin{array}{c|cccccc} a_i : & \cancel{2} & \cancel{2} & \cancel{6} & \cancel{2} & \cancel{6} & 7 \\ \hline b_i : & 1 & 3 & 4 & 5 & 2 & 6 \end{array}$$

So this says our tree should have edges 2-1, 2-3, 6-4, 2-5, 6-2 and 7-6. Comparing this to our original tree we see that it matches.

We have not checked (nor will we) all of the details but the above two procedures give a one-to-one correspondence between labelled trees on n vertices and sequences of length $n - 2$ with letters from $\{1, 2, \dots, n\}$. So we can conclude that there are n^{n-2} such trees, giving Cayley's Theorem.

Given a graph G a spanning tree of the graph is a subgraph of G which is a tree and which contains all of the vertices of G (i.e., it spans). We make the following observation

A graph has a spanning tree if and only if it is connected.

To see this we note that if the graph has a spanning tree then since a tree is connected we can get from any vertex to any other vertex using the tree. So we can get from any vertex to any other vertex in the whole graph (since the whole graph contains the tree) so the graph must be connected. On the other hand if the graph is connected we look for any edge that is contained in a cycle, if we find an edge we remove it. We continue until we cannot find such an edge, the resulting graph is a subgraph which is connected and does not have cycles, i.e., a spanning tree.

A *weighted* graph is a graph where each edge has been given a weight. For instance we might have several different routes to take and we can map all the various roads as a graph and on each edge we put the amount of time it would take to travel on that stretch of road. In this case one thing that we might be looking for are the shortest distance connecting two points (i.e., the start and end).

As another example we might be trying to hook up cities on a fiber optic network. We could draw a graph where the edges are the possible locations for us to lay fiber between cities and each edge is the cost associated with connecting the cities. A reasonable problem to ask is what is the cheapest way to connect the cities. Well we might as well assume that we remove redundancies since we can route data through the fibers we do not need to have any cycles so we are looking for a tree. In particular we want to have the tree that would give us the lowest total cost. This is an example of the minimum spanning tree problem.

Given a weighted graph G the minimum spanning tree problem asks us to find the spanning tree with the lowest total sum of edge weights. There are several different ways to do this, but they all boil down to one basic idea: *be greedy*.

A greedy algorithm is one where at each stage we make the best possible choice. For instance when you get change back at a store the cashier starts with the largest bill less than what you are owed and gives you that and then keeps repeating the procedure until the account is settled. The advantage of a greedy algorithm is that it is simple to understand and easy to implement. The disadvantage is that it is not always the best option, but in this case we are in luck! Below we will give two different algorithms that solves the minimum spanning tree problem.

Finding a minimum spanning tree (I):

Given a connected weighted graph look for the “heaviest” edge whose removal does not disconnect the graph. Continue the process until we have a tree.

Note that this is similar to the same proof showing that every connected graph has a spanning tree, the only difference is that we now make sure we remove the heaviest edge. So we start with all the edges and we trim down. Another algorithm works in the opposite direction by building up edges.

Finding a minimum spanning tree (II):

We start with no edges and we look among all the edges in G not currently in and add the “lightest” edge whose addition does not create a cycle. Continue the process until we have a tree.