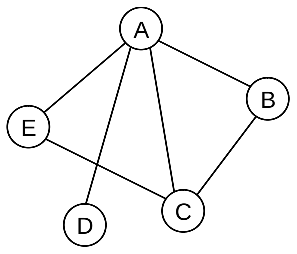
Discrete Mathematics
Graphs

Saad Mneimneh

1 Vertices, edges, and connectivity

In this section, I will introduce the preliminary language of graphs. A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, where an edge is an unordered pair of vertices (although we could extend the definition to ordered to allow for directed graphs). A graph can be visually represented by making vertices points (or small circles), with some pairs (not necessarily all pairs) connected by edges. It does not matter whether these edges are straight or curvy; all that is important is which pairs of vertices they connect.



$$V = \{A, B, C, D, E\}$$

$$E = \{(A, B), (A, C), (A, D), (A, E), (B, C), (E, C)\}$$

The above definition restricts our focus to simple graphs where we have no loops, i.e. no edges of the form (a, a), and no parallel edges, i.e. E is a set and not a multi-set. If E is a multi-set, where (u, v) can appear multiple times, we call the graph a multigraph. If the graph is directed, then (u, v) and (v, u) may both appear in E; in this case, we interpret all edges as directional edges (i.e. ordered pairs of vertices). But we will mainly consider simple undirected graphs.

We have previously seen the definition of a degree of a vertex, and related the sum of the degrees to the number of edges by the well-known Handshake Lemma. We also established simple results such as Euler formula for planar graphs and that the number of vertices with odd degree must be even for any graph. We now define paths and walks. A path consists of a sequence of

n distinct vertices $v_1, v_2, \ldots, v_n$ (except possibly for v_1 and v_n which can be equal), such that $(v_i, v_{i+1}) \in E$. We say that there is a path from v_1 to v_n. If $v_1 = v_n$, the path is called a cycle. If some vertex repeats in the middle of the sequence, the sequence is called a walk (instead of a path) or a closed walk (instead of a cycle). In the above picture, A, B, C is a path; A, B, C, A is a cycle; A, B, C, A, E, C is a walk; and A, B, C, A, E, C, A is a closed walk.

If we let the relation $u \rightsquigarrow v$ to mean there is a path from u to v, then we can show that $\rightsquigarrow$ is an equivalence relation. It is reflexive because $u \rightsquigarrow u$ (the empty path). It is symmetric because $u \rightsquigarrow v \Leftrightarrow v \rightsquigarrow u$ (simply reverse the path). It is also transitive but this is a bit tricky to show. If $u \rightsquigarrow v$ and $v \rightsquigarrow w$, then there is a walk from u to w. We say a walk because we cannot guarantee that the path from u to v and that from v to w do not share vertices (other than v that is). However, we can still construct a path from u to w (every walk can be converted to a path). Assume our walk looks like the following:

$$u \rightsquigarrow a \rightsquigarrow v \rightsquigarrow a \rightsquigarrow w$$

Then let a be the first such vertex encountered on the walk from u to w. Now construct this following path:

$$u \rightsquigarrow a \rightsquigarrow w$$

Why is the above a path? If some vertex repeats, it must occur before a. This contradicts our choice of a. We conclude that $u \rightsquigarrow w$ and that $\rightsquigarrow$ is transitive.

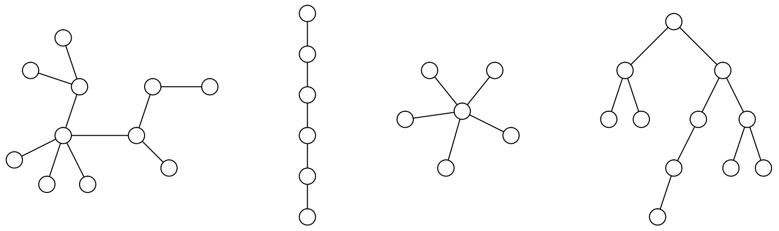
The connected components of a graph are the equivalence classes of $\rightsquigarrow$. Therefore, there is a path between every pair of vertices of a connected component. By the properties of equivalence classes, the connected components of a graph $G = (V, E)$ are disjoint and their union is V. A graph is connected if it has one equivalence class for $\rightsquigarrow$ (intuitively, there is a path between every pair of vertices).

If we have a graph G, then we can obtain a graph H by deleting some edges and/or vertices (if we delete a vertex we delete all the edges touching it of course). We say that H is a subgraph of G. If H is a connected component of G, then every subgraph of G that contains H is not connected (why?).

2 Trees

A tree is a connected graph that has no cycles. Some examples of trees are shown below. While we have used the connectedness and the cycle free properties to define a tree, a tree can be defined in many equivalent ways.

- (1) connected and cycle free
- (2) minimally connected: connected but removing any edges disconnects it
- (3) maximally cycle free: cycle free but adding any edge creates a cycle



We can further characterize the above minimal/maximal properties:

$$G = (V, E) \text{ is a tree} \Leftrightarrow G \text{ is connected and } |E| = |V| - 1$$

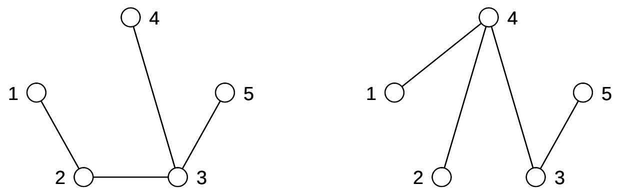
$$G = (V, E) \text{ is a tree} \Leftrightarrow G \text{ is cycle free and } |E| = |V| - 1$$

by showing that every tree with n vertices must have $n-1$ edges. We first show that every tree with at least two vertices must have a vertex with degree 1 (actually the following proof can be extended to prove the existence of two vertices with degree 1): start at any vertex and follow a path arbitrarily (there must be at least one edge). Eventually, you will have to stop because you cannot cycle. The vertex where you stop has degree 1 (to find another one, follow an arbitrary path starting from this vertex).

We now proceed by induction. Our base case is a tree with one vertex and, therefore, zero edges. Now for every $n \geq 2$, assume the property holds for a tree with $n-1$ vertices, and let us show that it remains true for a tree with n vertices. Given a tree with $n \geq 2$ vertices, there must be a vertex of degree 1. Removing this vertex with the edge connecting it produces a tree with $n-1$ vertices. By the inductive hypothesis, this tree must have $n-2$ edges. Therefore, our original tree has $n-1$ edges. Done.

3 Counting trees

How many trees are there on n vertices? To answer this question, we have to clarify when we consider two trees to be different. Consider the following two trees. Are they different or the same? There are two possibilities:



- labeled trees: If the trees are labeled, i.e. the vertices of the tree are labeled $1, 2, \ldots, n$, then two trees are the same if the have the same set of edges. The two above trees are not the same; for instance edge $(1, 2)$ exists in the first but not in the second.

- unlabeled trees: Two trees are the same if there is a one-to-one correspondence between their vertices such that an edge exists in the first if and only if the corresponding edge exists in the second. We call such trees isomorphic. The two above trees are isomorphic, and here's a one-to-one correspondence $\{(1,5),(2,3),(3,4),(4,1),(5,2)\}$.

It is easier to count labeled trees. So we will do that first, and we will then compute lower and upper bounds on the number of unlabeled trees. Let T_n be the number of labeled trees on n vertices. Let S_i be the set of trees in which vertex i has degree 1. Since every tree has a vertex of degree 1,

$$T_n = |S_1 \cup S_2 \cup \ldots \cup S_n|$$

We can use the inclusion-exclusion principle to compute the above quantity. Observe that $|S_i| = (n-1)T_{n-1}$ because there are T_{n-1} trees on vertices $1,\ldots,i-1,i+1,\ldots,n$ and vertex i can be added by connecting to any of the remaining $n-1$ vertices. Similarly, $|S_i \cup S_j| = (n-2)^2 T_{n-2}$. This expression can be generalized to $(n-k)^k T_{n-k}$. Also note that $|S_1 \cap S_2 \cap \ldots \cap S_n| = 0$ when $n \geq 3$.

$$T_n = \binom{n}{1}(n-1)T_{n-1} - \binom{n}{2}(n-2)^2 T_{n-2} + \ldots + (-1)^{n-2}\binom{n}{n-1}T_1$$

$$= \sum_{k=1}^{n-1}(-1)^{k-1}\binom{n}{k}(n-k)^k T_{n-k} = \sum_{k=1}^{n-1}(-1)^{n-k-1}\binom{n}{n-k}k^{n-k}T_k$$

$$= \sum_{k=1}^{n-1}(-1)^{n-k-1}\binom{n}{k}k^{n-k}T_k$$

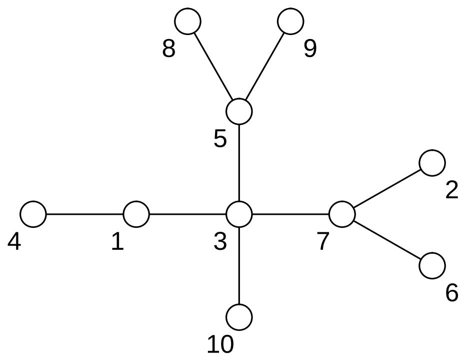
Recall that $S(m,n)n! = \sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}k^m$. Therefore, if we let $T_k = k^{k-2}$, then we retrieve this familiar expression involving the Stirling number $S(m,n)$ with $m = n-2$. For $n \geq 3$,

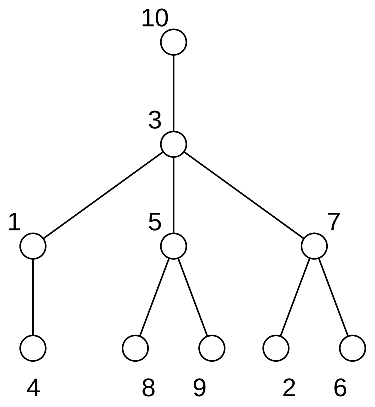
$$T_n = \sum_{k=1}^{n-1}(-1)^{n-k-1}\binom{n}{k}k^{n-2} = -\sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}k^{n-2} + n^{n-2}$$

$$= -S(n-2,n)n! + n^{n-2} = 0 + n^{n-2}$$

Therefore, if $T_k = k^{k-2}$ for $k = 3\ldots n-1$, then $T_n = n^{n-2}$. This is a proof by induction, and we only need a base case: $T_3 = 3^{3-2} = 3$, which is correct so we are done. Since $T_1 = T_2 = 1$, $T_n = n^{n-2}$ can be extended to $n \geq 1$.

There is another way of obtaining the same result using a one-to-one correspondence between labeled trees and what is known as a Prüfer code. The Prüfer code is a string of length $n-2$, where each character of the string is in $\{1,2,\ldots,n\}$. The number of such strings is obviously n^{n-2}. I will illustrate how





to generate the Prüfer code for a given tree, then prove the one-to-one correspondence. First, root the tree at vertex n. It becomes a rooted tree as shown below. Define a leaf to be a vertex of degree 1 other than the root (vertex n). Define the parent of a vertex v to be the vertex that immediately precedes it on the path from the root (vertex n) to v (conversely, v is the child). Then, start with an empty Prüfer code and repeatedly pick the smallest numbered leaf, remove it from the tree, and add its parent to the Prüfer code. The process stops when the tree has two vertices. Since every tree with at least two vertices has at least two vertices of degree 1, a leaf can always be found even if the degree of the root (vertex n) is ever 1. The Prüfer code is obtained after the removal of $n-2$ vertices; the $(n-1)^{\text{st}}$ vertex must have the root as parent and hence it is redundant to always end the Prüfer code with the number n. Here's the Prüfer code for the above example (vertices removed in order: 2, 4, 1, 6, 7, 8, 9, 5):

$$7\ 1\ 3\ 7\ 3\ 5\ 5\ 3$$

Now I will show how to reverse the process, i.e. given a Prüfer code, find the exact same tree that generated it. This will prove the one-to-one correspondence. All we have to do is guess the sequence of vertices that were removed.

Then we connect the vertices to their parents as given by the Prüfer code. Let $p[1..n-1]$ represent the Prüfer code appended with the number n, and $v[1..n-2]$ represent the $n-2$ vertices that were removed in order. We compute $v[1..n-1]$ as follows.

```
for i=1 to n-1
  v[i]=smallest number not in
       v[1..i-1] or p[i..n-1]
tree is graph ({1,...,n},{(v[1], p[1]),...,(v[n-1],p[n-1])})
```

Why does the above process succeed in retrieving the tree? When constructing the Prüfer code, we repeatedly add the smallest numbered leaf. Therefore, this leaf cannot be the parent of any future leaf. The above process repeatedly finds the smallest leaf that is not a future parent and puts it in $v[i]$. How do we prove that $1 \leq v[i] \leq n-1$? Observe that $v[1..i-1]$ and $p[i..n-1]$ contain at most $[(i-1)-1+1]+[(n-1)-i+1] = n-1$ numbers, one of them is $n = v[i-1]$. Therefore, they contain at most $n-2$ numbers in $\{1,\dots,n-1\}$. So $v[i]$ will find a number in $\{1,\dots,n-1\}$.

The significance of the Prüfer code is that it gives an efficient way of storing the tree (although in most applications of data structures a tree is not stored in this way).

Finally, a proof based on counting in two different ways can also establish that $T_n = n^{n-2}$. The proof counts in two different ways the number of different sequences of directed edges that can be added to an empty graph on n vertices to form a rooted tree. One way to form such a sequence is to start with one of the T_n possible unrooted trees, choose one of its n vertices as root, and choose one of the $(n-1)!$ possible sequences in which to add its $n-1$ edges. Therefore, the total number of sequences that can be formed in this way is $T_n n(n-1)! = T_n n!$.

Another way to count these edge sequences is to consider an algorithm that generates an edge sequence by adding the edges one by one to an empty graph in $n-1$ phases. If one has added $k-1$ edges already, so that the graph formed by these edges consists of $n-k+1$ trees, there are $n(n-k)$ choices for the next edge to add in the k^{th} phase: its starting vertex can be any one of the n vertices of the graph, and its ending vertex can be any one of the $n-k$ roots other than the root of the tree containing the starting vertex. Therefore, by the multiplication rule, the total number of choices is

$$\prod_{k=1}^{n-1} n(n-k) = n^{n-1}(n-1)! = n^{n-2}n!$$

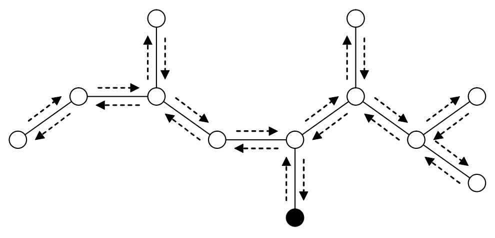
Equating these two formulas for the number of edge sequences results in $T_n = n^{n-2}$. The advantage of this proof is that it can be adapted to answer questions like: how many rooted trees on n vertices are there which have a specific set of k edges? Try to answer this question using a similar counting argument (Hint: the answer is n^{n-k-1}). You might also consider the following questions: how many unrooted trees on n vertices are there which have a specific edge, or a specific pair of edges (Hint: it will make a difference whether the edges share a vertex or not, the answers are $2n^{n-3}$ for one edge, $3n^{n-4}$ for two edges that share a vertex, and $4n^{n-4}$ for two edges that don't).

What about unlabeled trees. Every tree can be labeled in $n!$ ways. But different labelings may produce the same tree (e.g. a star tree). Therefore, if we let T_n be the number of labeled trees, then

$$U_n n! \geq T_n = n^{n-2}$$

$$U_n \geq \frac{n^{n-2}}{n!}$$

Consider the tree rooted at some vertex v. Start a walk at v that visits all the vertices and comes back to v. This walk traverses every edge twice: once in the direction parent to child, and once in the direction child to parent. If we encode these directions by 0 and 1 respectively, we obtain a binary string of length $2(n-1)$. But not every binary string of length $2(n-1)$ defines a walk on a tree.



000001101110010010111

Therefore,

$$U_n \leq 2^{2(n-1)} = 4^{n-1}$$

and we conclude that

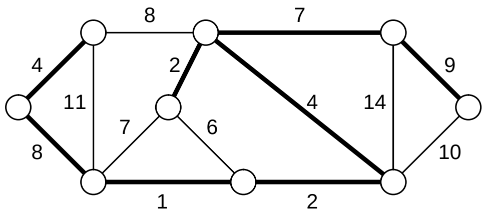
$$\frac{n^{n-2}}{n!} \leq U_n \leq 4^{n-1}$$

4 Finding the best tree

Consider a graph $G = (V, E)$ with a function $w : E \to \mathbb{R}$ that assigns a weight for every edge in E. We call this a weighted graph. Let's say we would like to find the smallest subgraph of G that connects all the vertices. Obviously, such a subgraph cannot contain cycles. So it must be a tree by definition. But which tree? It is often desirable to find the tree with the smallest total weight, i.e. a tree that minimizes the sum of the weights of its edges. Since the number of labeled trees on n vertices is n^{n-2}, an algorithm that explores all trees to pick the best one is very inefficient.

The problem just described above is known as the minimum spanning tree (MST). As it turns out, there are efficient ways to find an MST. Here's one:

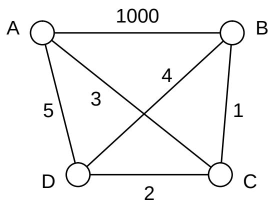
sort the edges by weight from smallest to largest. Start with an empty tree, and repeatedly pick the smallest edge and add it to the tree if it does not create a cycle (alternatively, one could repeatedly delete from the graph the largest edge that does not disconnect the graph). We stop when we have added $|V|-1$ edges. This is a greedy algorithm in the sense that we make what seems to be the best choice (smallest weighted edge) at the moment. The following figure shows an example run of this algorithm where the edges in bold end up being in the tree (see if you can reproduce this tree by manually running the algorithm).



We have seen before that sorting will require $O(|E|\log|E|)$ time. In addition, checking for cycles can be done in a total of $O(|V|\log V)$ time but the analysis will not be covered here. Since $|E|$ is $O(|V|^2)$, the total running time for the greedy algorithm is

$$O(|E|\log|V|^2+|V|\log|V|)=O(|E|\log|V|^2)=O(2|E|\log|V|)=O(|E|\log|V|)$$

Greedy algorithms, despite being efficient, do not always guarantee the optimal. Consider for example the traveling salesman problem.



The goal is to visit every vertex exactly once and return to the starting vertex in the cheapest way. Technically, we need a minimum weight Hamiltonian cycle (a cycle that goes through every vertex exactly once), i.e. a Hamiltonian cycle that minimizes the sum of the weights of its edges. A greedy algorithm similar to the above can be envisioned. For instance, we can examine edges in order of their weights and add an edge if it is not the third edge to touch a vertex and if it does not create a cycle, unless the cycle contains all vertices. On the above graph, this greedy algorithm will produce the cycle $ABCDA$ with total weight 1008. A much better cycle exists: $ACBDA$ has weight 13. We will revisit the traveling salesman problem. For now, let's prove that the greedy algorithm for MST works.

Assume that the greedy algorithm makes a mistake for the first time by adding edge $e = (u, v)$. Let T' be a minimum spanning tree that contains all edges in T prior to adding e (T' exists by the choice of e). In T', there must be a path from u to v, and on that path, there must be an edge e' that is not in T (otherwise, T would contain a cycle formed by the path $v \rightsquigarrow u$ and edge $e = (u, v)$). If $w(e') < w(e)$, then the greedy algorithm must have examined e' before e. But since it decided not to include e' in T, then adding e' would have created a cycle. But all edges in T up to that point (including e') are also in T', so T' contains a cycle, a contradiction. Therefore, $w(e') \geq w(e)$, and by replacing e' by e in T' we obtain a minimum spanning that does contain e, contradicting that the greedy algorithm made a mistake.

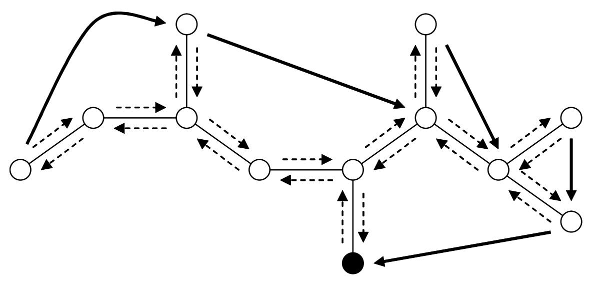
5 The traveling salesman

The traveling salesman is a very hard problem, and no efficient algorithm is known to exists that solves it optimally. We are going to consider a special version of the traveling salesman problem that satisfies the triangular inequality. For every three vertices i, j, and k:

$$w(i, j) \leq w(i, k) + w(k, j)$$

In this case, we can find a Hamiltonian cycle that is at most twice the weight of the optimal cycle. Here's how:

1. Find a minimum spanning tree (let's denote its weight by MST)
2. Construct a closed walk that traverses the entire tree (as we did before, it goes through every edge twice)
3. Transform the closed walk into a cycle by applying shortcuts (illustrated below in bold)

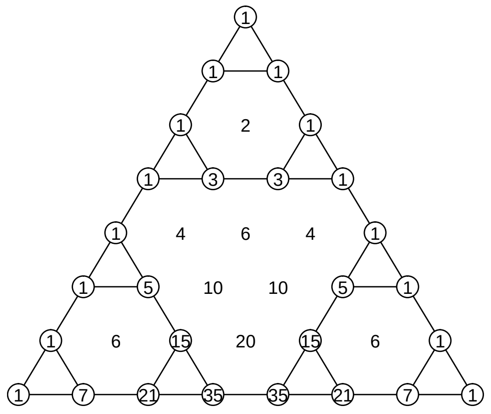


If we denote the weight of the minimum weight Hamiltonian cycle by OPT, then $MST \leq OPT$ because breaking the Hamiltonian cycle by removing any edge gives a tree. Moreover, since the triangular inequality holds, applying shortcuts can only decrease the weight and, therefore, the weight of the cycle is at most that of the closed walk, which is $2MST \leq 2OPT$.

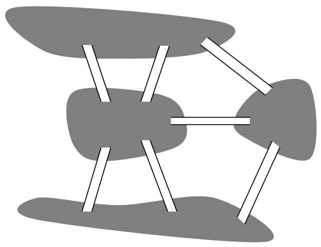
6 Hamiltonian cycles and Eulerian walks

In the traveling salesman problem, we assumed that a Hamiltonian cycle exists and wanted to find the one with minimum weight. In general, however, deciding whether a graph has a Hamiltonian cycle is very hard. Such a graph is called Hamiltonian.

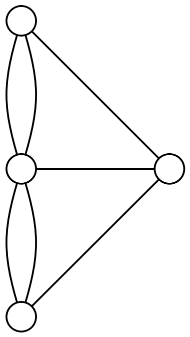
An example of a Hamiltonian graph is the graph obtained from the configurations of the towers of Hanoi. A tower of Hanoi with n discs has 3^n possible configurations of discs (why?). If we make each configuration a vertex in a graph, and connect two vertices if their corresponding configurations are separated by one move, then we obtain a Hamiltonian graph. This means that we can go through all 3^n configurations and come back to the starting point in exactly 3^n moves, i.e. in an optimal way. What does the graph look like? It turns out there is a nice connection to the Pascal triangle. Below is the tower of Hanoi graph for $n = 3$ (with $3^3 = 27$ vertices), it is obtained by connecting the odd neighboring entries in the Pascal triangle. Can you find a Hamiltonian cycle? The three corners of the outer triangle correspond to all discs on peg 1, all discs on peg 2, and all discs on peg 3. Can you find a path that corresponds to an optimal solution (the length of the path must be $2^3 - 1 = 7$)?



Surprisingly, if instead of a Hamiltonian cycle, we seek a cycle that goes through every edge exactly once, called Eulerian cycle, then finding whether such a cycle exists or not becomes easy. This was inspired by the bridges of Königsberg (now Kaliningrad). Is it possible to take a walk (and maybe come back to the starting point) that crosses every bridge exactly once?



The situation can be modeled as a multigraph (with possibly multiple edges connecting a given pair of vertices). The question becomes whether the multigraph a Eulerian walk (the walk may or may not be closed).



We can make the following two observations:

- if a vertex v has odd degree, then every Eulerian walk must either start or end at v.
- if a vertex v has even degree, then an Eulerian walk that starts at v must end at v.

In fact, Euler proved the following: A connected multigraph has a closed walk if and only if every vertex has even degree.

($\Rightarrow$): If a closed walk exists, then each time a vertex v is visited by using some edge (u, v), another edge (v, w) must be used to leave the vertex. Therefore, the vertex must have an even number of edges touching it (that's its degree).

($\Leftarrow$): we prove this by constructing the Eulerian walk. We start at some vertex u and follow edges arbitrarily using every edge at most once until we are stuck and cannot leave some vertex v. It must be that $u = v$; otherwise, v has odd degree (why?). So we have a closed walk that uses every edge at most once. If we have used all the edges, then we are done and our walk is in deed Eulerian. Otherwise, there must be some edge (w, q) that is not on the walk. Actually, there must be such an edge where w has been visited already (because the graph is connected). Starting from w, we repeat the same process to obtain another walk and merge the two walks. We continue this way until we have used all the edges, i.e. our walk is Eulerian.

We can use the above result to show that if a connected multigraph graph has exactly two vertices u and v with odd degrees, then it has a non-closed Eulerian walk. Simply connect u and v with an extra edge. Now all vertices have even degrees. Construct a closed Eulerian walk then break it by removing edge (u, v). What if we have more than two vertices with odd degree? Then a Eulerian walk cannot exist (why?).

7 Graph coloring and planarity

A coloring of a graph $G = (V, E)$ is an assignment of colors to all the vertices in V such that if $(u, v) \in E \Rightarrow u$ and v have different colors. The chromatic number of a graph G, $\chi(G)$, is the smallest number of colors needed to color the graph. Let Δ be the largest degree of a vertex in G, then

$$\chi(G) \leq \Delta + 1$$

We prove this by induction. The base case is any $|V| \leq \Delta + 1$, which obviously does not require more than $\Delta + 1$ colors. For $|V| > \Delta + 1$, consider a vertex $v \in V$ and the subgraph H obtained by deleting v (and the edges touching it) from G. Now H has $|V| - 1$ vertices and the largest degree in H is still at most Δ. By the inductive hypothesis, H can be colored using $\Delta + 1$ colors. But since v has a degree of at most Δ, there must be a color among these $\Delta + 1$ colors that can be used for v. This inductive proof can be converted into a greedy algorithm for coloring G using at most $\Delta + 1$ colors.

A graph $G = (V, E)$ is bipartite iff V can be partitioned into two (disjoint) sets A and B such that

$$(u, v) \in E \Rightarrow (u \in A \wedge v \in B) \vee (u \in B \wedge v \in A)$$

Another characterization of a bipartite graph is that it is a graph with no odd length cycles. In fact, these two definitions are equivalent. It is obvious that a bipartite graph cannot have odd length cycles. Let's explore the other direction. Choose an arbitrary vertex v and label it EVEN. Then label all its neighbors ODD (those vertices that are connected to v by an edge). Repeatedly label neighbors of vertices in this way by alternating EVEN and ODD. The process ends when all vertices are labeled (this is guaranteed if the graph is connected; otherwise, perform the process on each connected component). During this process, a vertex might have been labeled more than once, but any re-labeling must be consistent with an existing label since, otherwise, we discover an odd length cycle (because the is at an EVEN distance from v and at an ODD distance from v). Now place all vertices labeled EVEN in A, and all vertices labeled ODD in B. This proves that the graph is bipartite. In fact, A and B can be thought of as colors. So we have the following results:

$$G \text{ is bipartite} \Leftrightarrow \chi(G) = 2$$

Coloring has been a famous topic with planar graphs. Recall that a planar graph is a graph which can be drawn in the plane with no edges crossing (again edges need not be straight lines). The famous four color theorem establishes that every planar graph is 4-colorable. We can prove a more modest result: every planar graph can be colored with 6 colors (and with a little more effort we can prove that 5 colors are enough). Recall Euler's formula (where v, e, and e refer to vertices, edges, and faces respectively):

$$v - e + f = 2$$

If we define the degree of a face to be the number of edges encountered on a closed walk around its boundary, then in the sum of degrees of faces every edge is counted twice (it is on the border of two faces or within the same face). In an analogous way to the Handshake lemma, the sum of degrees of faces is equal to $2e$. Since every face must have degree at least three (exclude graphs with less than 3 vertices), then

$$3f \le 2e$$

$$e = v + f - 2 \le v + \frac{2}{3}e - 2 \Rightarrow e \le 3v - 6$$

But the sum of degrees of vertices is $2e$ (Handshake lemma), so the average vertex degree is

$$\frac{2e}{v} \le \frac{6v - 12}{v} < 6$$

Therefore, there must be a vertex with degree at most 5.

Now if all vertices have degree at most 5, we can color the graph using 6 colors as illustrated by the first result of this section. But since every subgraph of a planar graph obtained by deleting a vertex must be planar, every such subgraph must have a vertex with degree at most 5. Therefore, the same inductive proof can be adapted to show that 6 colors are enough. In fact, the first result can be generalized as follows: If every subgraph of G has a vertex of degree at most Δ, then $\chi(G) \le \Delta + 1$.

Euler's formula is very useful. It can be used to show, for instance, that the number of planar graphs in which every vertex has degree d and every face has degree b is exactly 5. These are the solutions corresponding to (d, b): (3,3), (3,4), (4,3), (3,5), and (5,3). The system of equations is:

$$v - e + f = 2$$

$$dv = 2e$$

$$bf = 2e$$

Euler's formula can also be used to show that the complete bipartite graph with 3 vertices in each partition, called $K_{3,3}$, and the complete graph on 5 vertices, called K_5 are both non-planar. Here are the two corresponding systems of equations, which are non-solvable:

$\underline{K_{3,3}}$
$v - e + f = 2$
$4f \leq 2e$ (every face has degree at least 4)
$v = 6$
$e = 9$

$\underline{K_5}$
$v - e + f = 2$
$3f \leq 2e$ (every face has degree at least 3)
$v = 5$
$e = 10$

In fact, Kuratowski proved that a graph is planar if and only if it does not contain a subdivision of $K_{3,3}$ or K_5. What is a subdivision? Inserting a new vertex into an existing edge of a graph is called subdividing the edge, and one ore more subdivisions of edges creates a subdivision of the original graph.

Finally, I present a nice connection between Hamiltonian graphs (those which have a Hamiltonian cycle), bipartite graphs (2-colorable), and planar graphs. Given a graph G with a Hamiltonian cycle H,

- Draw G in the plane with H on the outside, i.e. with H as the boundary of the "outer face".
- List the edges of G not in H, $e_1, \ldots, e_n$.
- Form a new graph K in which the vertices are labeled $e_1, \ldots, e_n$, and where the vertices labeled e_i, e_j are connected by an edge if and only if e_i and e_j cross in the drawing of G, i.e. they cannot both be drawn inside H or outside H.

Then G is planar if and only iff K is bipartite. The proof is easy: If G is planar, then some of $e_1, \ldots, e_n$ can be drawn inside H and some outside H resulting in a bipartite graph K. Conversely, if K is bipartite, say with the two sets of vertices A and B, then we can draw the edges in A inside H and the edges in B outside H to obtain a planar graph G.

Since both $K_{3,3}$ and K_5 are Hamiltonian, this result can be used to prove that neither of the two is a planar graph (try it).