Discrete Math Lecture 8[1]

1 Graphs

Networks are everywhere. Any situation where you have a bunch of objects that have connections between them is a sort of network. Some examples include a network of roads between cities, a network of connections between distributed processors, a social network of friendships between people and so on. Graphs are a mathematical model that can be used to model networks and since such networks are ubiquitous, graphs are also ubiquitous. In an algorithms class, you will see a bunch of common algorithms to perform a variety of tasks on input that consists of a network of some sort, such as traversing through a network, finding the shortest path between any two nodes in a network, or detecting if there are loops or cycles in a network. These algorithms are all phrased as acting on an input graph. In this course however, we will just introduce the mathematical formalism of graphs and study a few of the basic properties of graphs. We will also see a few special types of graphs that come up especially often in computer science and study a few special properties of these kinds of graphs.

Before we get started, it should be emphasized that one great feature of using graphs as a model is that we can use graphs to model situations that you might not immediately think of as having a graph structure. For example, suppose you have a bunch of data about birth and death dates of a large number of individuals. You want to check if this information is consistent: that is is it possible for all the data to be simultaneously correct? Of course A can't be born before B if B died before A was born. But in general there may be more complicated reasons why some data might be inconsistent. You can model this information as a directed graph where the people's birth and death dates are vertices and there are directed edges going from earlier events to later events. Now checking whether the information is consistent just boils down to checking if the resulting graph has a cycle. In general, checking whether a bunch of constraints can be simultaneously satisfied can often be reduced to some kind of cycle detection problem in a graph. And there are fast algorithms to detect if graphs have cycles, which we can now use to solve these problems.

Another example where modeling things using a graph might be useful, is to study strategies in a multi-person game. Each game state can be represented as a vertex and edges will represent legal moves that correspond to changing the game state from one state to another. Now we can talk about strategies that lead to a forced win by one of the players in terms of structures within this graph (such as paths that lead to vertices corresponding to states where a particular player has won the game.)

Graphical models are used to model the structure of dependencies between random variables. These models often involve directed graphs where the variables are the vertices and the dependencies are the edges. This is widely used in machine learning frameworks.

The whole world wide web can itself be thought of as a graph of interconnected web pages (vertices) and edges (hyperlinks) between them. Google's PageRank algorithm looks at this graph and ranks websites based on this graph structure in order to be able to list relevant website results for any given keyword. PageRank measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption, roughly speaking, is that a page is only as important as the pages that link to it. One tricky thing is that Page Rank needs to be relatively robust so that you can't just set up a bogus website with a few other sites linking to it and have your newly created site appear high on Google search results.

[1]large parts of the notes for this lecture are based on Prof. Timothy Ng's notes on this topic

Yet another example where graphs are used to model situations is in the study of social influence/ dynamics. We can imagine a social network as a graph where the people are vertices and the edges represent the relationships between people. We can even associate weights to each of the edges to represent the strength or degree of closeness of the relationship. We can model people's beliefs about any given issue as being influenced by people who are their neighbours in this graph. The closer the relationship, the stronger the influence. One simple model might be that each person's views are influenced by the weighted average of the beliefs of all their neighbours. This can be modeled by doing iterated updates on the graph that we use to model the situation. We can now ask how quickly a belief that is dominant in one cluster of people will spread throughout the group of people. The answer will depend on various properties of the underlying social network graph.

Graphs have also been used to explain various other phenomena on social networks. A famous experiment in the social sciences led to the discovery of the so called small-world phenomenon. The fact that social networks are rich in short paths is known as the small-world phenomenon, or the "six degrees of separation," and it has long been the subject of both anecdotal and scientific fascination. The first significant empirical study of the small-world phenomenon was undertaken by the social psychologist Stanley Milgram who asked randomly chosen individuals to each try forwarding a letter to a designated "target" person living in the town of Sharon, MA, a suburb of Boston. He provided the target's name, address, occupation, and some personal information, but stipulated that the participants could not mail the letter directly to the target; rather, each participant could only advance the letter by forwarding it to a single acquaintance that he or she knew on a first-name basis, with the goal of reaching the target as rapidly as possible. Roughly a third of the letters eventually arrived at the target, in a median of six steps, and this has since served as basic experimental evidence for the existence of short paths in the global friendship network, linking all (or almost all) of us together in society. This style of experiment, constructing paths through social networks to distant target people, has been repeated by a number of other groups. This has led to the popularity of the term "six degrees of separation," referring to the fact that any two people are quite likely to only separated by a short (around 6) chain of acquaintances.

Milgram's experiment really demonstrated two striking facts about large social networks: first, that short paths are there in abundance; and second, that people, acting without any sort of global "map" of the network, are effective at collectively finding these short paths. It is easy to imagine a social network where the first of these is true but the second isn't— a world where the short paths are there, but where a letter forwarded from thousands of miles away might simply wander from one acquaintance to another, lost in a maze of social connections. A large social-networking site where everyone was known only by 9-digit pseudonyms would be like this: if you were told, "Forward this letter to user number 482285204, using only people you know on a first-name basis," the task would clearly be hopeless. The real global friendship network contains enough clues about how people fit together in larger structures — both geographic and social — to allow the process of search to focus in on distant targets.

Jon Kleinberg's seminal work on the small world phenomenon explains what properties of the underlying social graph of acquaintances leads to this small world effect. This effect shows up in many other networks (including, to an extent, the world wide web) and such networks have the property that the nodes in the network quickly route information to a designated target efficiently and in a decentralized manner. You can experience a version of this phenomenon by playing the following game. Pick a Wikipedia page and a totally unrelated other Wikipedia page. For example one page might be the page for your favourite style of cuisine and the other might be the Wikipedia page for let's say graphs. You will find that you can get from one page to the other through a sequence of a relatively small number of links within Wikipedia!

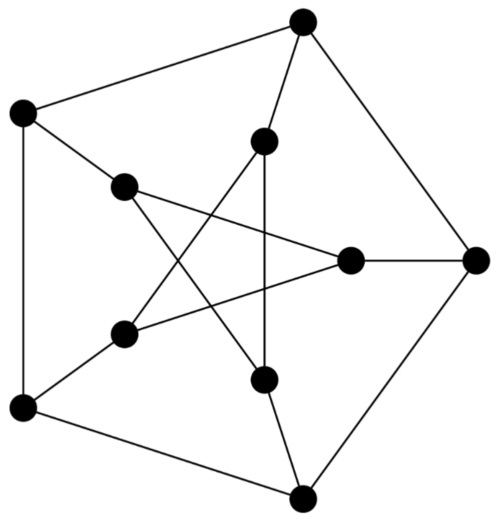

Figure 1: The Petersen Graph

It often turns out that to solve problems involving a network, what we really want to do is compute some property of the underlying graph. And often to solve such a problem, we first need to realise that we can model the situation using a graph, and then the solution involves computing some properties of that graph. We will see an example of this today in the case of graph coloring.

We've actually already worked with graphs a couple of times in this class. Our discussion of Ramsey numbers ended with us showing that if we have a small enough ($< 2^{k/2-1}$) number of people at a party, then it is possible to set up a situation where no k are mutual friends or mutual enemies. In the language of graphs, we can imagine we have a graph of n vertices and some number of edges between them. Then what our result shows is that if n is not too big, then it is always possible to construct a graph with no k vertices that are all connected to each other (which we call a k-clique) and no k vertices that are all not connected to each other (which we call a k-anti-clique).

We also saw, as part of a homework problem, that given any graph representing a roads (edges) between cities (vertices), we can always divide the vertices into two disjoint sets, such that at least half the edges are between the two sets.

Let's now formally define graphs and some basic terms associated with them.

Definition 1 ((simple) Graphs)**.** *A graph $G = (V, E)$ is a pair with a non-empty set of vertices V together with a set of edges E. An edge $e \in E$ is an unordered pair (u, v) of vertices $u, v \in V$.*

This graph is called the Petersen graph. The Petersen graph is a particularly interesting graph, not just because it looks nice, but because it happens to be a common counterexample for many results in graph theory.

Note that, since edges are unordered, we really should be writing them as $\{u, v\}$, but we don't. The usual notation is (u, v) even when the edges are not directed.

One can define graphs more generally and then restrict our consideration to our particular definition of graphs, which would usually be then called *simple* graphs. The more general definition of graphs allows for things like multiple edges between vertices or loops. Other enhancements may include asymmetric edges (which we call directed edges), equipping edges with weights, and even generalizing edges to hyperedges that relate more than two vertices. Instead, what we do is start with the most basic version of a graph, from which we can augment our definition with the more fancy features if necessary (it will not be necessary in this course).

However let's briefly see why you might sometimes want undirected edges and sometimes you might want directed edges. Consider the social network on Meta (erstwhile Facebook). The people can be modeled as vertices and then the edges are between the vertices and represent friendships. But if I am your friend on Facebook, then you are also my friend on Facebook. So we can just use undirected edges to model this. However on another popular social network, Twitter, we have a directed relationship possible between people. I might follow Obama on Twitter, but it certainly doesn't mean that he follows me! Thus the social network of Twitter might be better modelled as a directed graph, where once again the people are represented as vertices and there is directed edge from person u pointing to person v if u follows v.

Definition 2 (Adjacent vertices)**.** *Two vertices u and v of a graph G are adjacent if they are joined by an edge (u, v). We say the edge (u, v) is incident to u and v.*

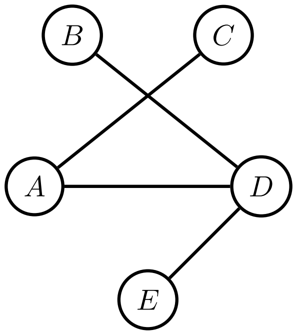

Figure 2: Graph G

In the graph in the figure, A and C are adjacent, but A and B are not. Here the set of vertices is $V = \{A, B, C, D, E\}$ and the set of edges is $E = \{(A, C); (A, D); (B, D); (D, E)\}$.

Note that a graph with n vertices has a minimum of 0 edges, and at most it can have $\binom{n}{2}$ edges, which would happen if all pairs of neighbours are adjacent.

Definition 3 (neighbours)**.** *Two vertices u and v are neighbours if they are adjacent. The neighbourhood of a vertex v is the set of neighbours of v*

$$N(v) = \{u \in V \mid (u, v) \in E\}.$$

Definition 4 (neighbouring set of a set of vertices)**.** *We can extend this definition to sets of vertices $A \subseteq V$ by*

$$N(A) = \bigcup_{v \in A} N(v)$$

In Figure 2, the set of neighbours of $\{A, B\}$ is $\{C, D\}$.

Definition 5 (degree)**.** *The degree of a vertex v in G, denoted* $\deg(v)$*, is the number of its neighbours. A graph G is k-regular if every vertex in G has degree k.*

In Figure 2, the degree of A is 2, while the degree of D is 3. Observe that every vertex in the Petersen graph has degree 3, so the Petersen graph is 3-regular, or cubic.

Definition 6 (degree sequence)**.** *The degree sequence of a graph G, is the sequence of degrees of it's vertices listed in increasing order.*

In Figure 2, the degree sequence of the graph is $(1, 1, 1, 2, 3)$.

The following result is one of the first graph theory results, proved by Leonhard Euler in 1736.

Theorem 1 (Handshaking Lemma)**.** *Let $G = (V, E)$ be an undirected graph. Then*

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|$$

Proof. Every edge (u, v) is incident to exactly two vertices: u and v. Therefore, each edge contributes 2 to the sum of the degrees of the vertices in the graph. □

Now, let's look at some examples of some well-known graphs.

Definition 7 (complete graph)**.** *The complete graph on n vertices is the graph $K_n = (V, E)$, where $|V| = n$ and*

$$E = \{(u, v) \mid u, v \in V\}.$$

That is, every pair of vertices are neighbours.

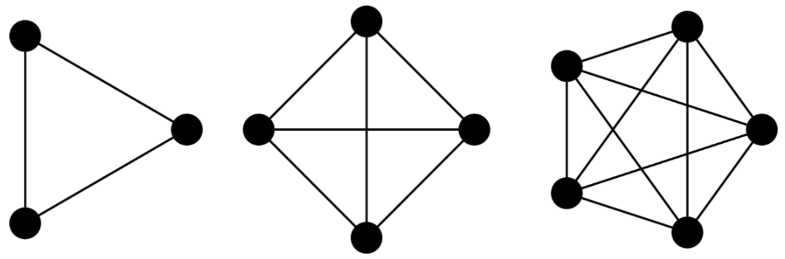

Figure 3: The complete graph on 3, 4 and 5 vertices.

Definition 8 (cycle graph)**.** *The cycle graph on n vertices is the graph $C_n = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{(v_i, v_j) \mid j = i + 1 \bmod n\}$. It's named so because the most obvious way to draw the graph is with the vertices arranged in order, on a circle.*

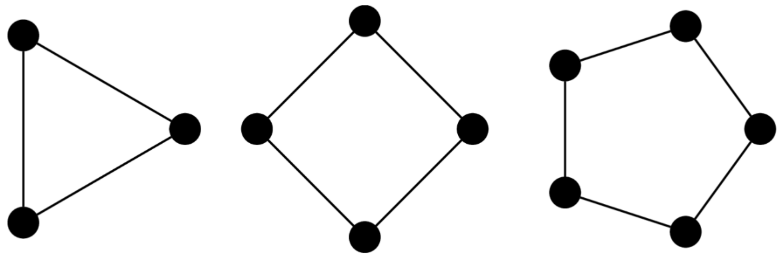

Figure 4: Some cycle graphs for n=3,4 and 5

Definition 9. *The n-(hyper)cube graph is the graph $Q_n = (V, E)$, where the vertices represent the binary strings of length n. $V = \{0, 1\}^n$ (i.e., binary strings of length n) and (u, v) is an edge if, for $u = a_1a_2 \cdots a_n$ and $v = b_1b_2 \cdots b_n$, there exists an index $i, 1 \le i \le n$ such that $a_i \neq b_i$ and $a_j = b_j$ for all $j \neq i$. In other words, u and v have an edge between them if the binary strings that they represent differ in exactly one position.*

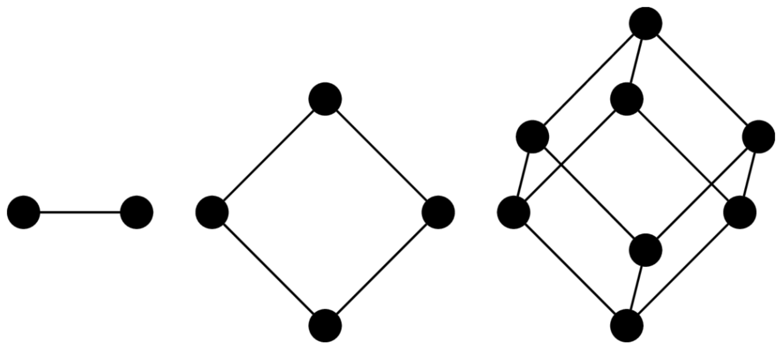

Figure 5: The n-(hyper)cube graph for $n = 1$, 2 and 3

We can look at the n hyper cube and use it to find the fewest number of bits you need to change in an n bit binary string s_1 to convert it into the string s_2. This is called the Hamming distance between the binary strings and is a useful measure in information theory. It is just the distance between the vertices representing the two strings in the n hypercube graph.

2 Graph isomorphism

Two graphs might look very different at first sight, and yet be the same graph! What does this mean? How is this possible? Well, when it comes to graphs, our intuition is that all that matters are the number of vertices and how they are all connected to each other. Just renaming the vertices should not matter. Look at the following graph from Figure 2:

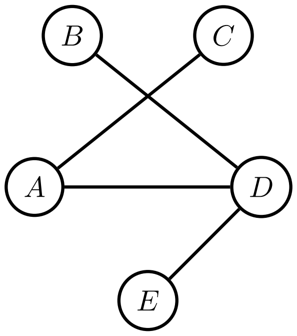

Figure 6: Graph G

If I just change the vertex names, I can instead have the graph:

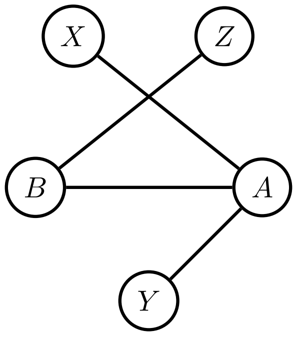

Figure 7: Graph G

But clearly the underlying network is the same! All I did was change vertex names. We can have a function f that maps the old names of the vertices to the new names. $f(A) = B, f(B) = X, f(C) = Z, f(D) = A, f(E) = Y$. Notice that vertices u and v are adjacent in the old graph if and only if vertices $f(u)$ and $f(v)$ are adjacent in the new graph. And this really is all that matters to us.

Such a function is called a graph isomorphism and the graphs in figures 6 and 7 are isomorphic. So, changing the vertex names does not really matter. But how we draw the graph, in a certain sense, also does not matter. If I have a network, the network doesn't change if I simply position the vertices differently in my drawing and draw the connections in a different looking way. A graph can have different representations if I draw it in different ways, but these representations are not really different. Such graphs are also going to be called isomorphic.

For example, consider the cube graph for $n = 3$. The way we drew it, it looks like this

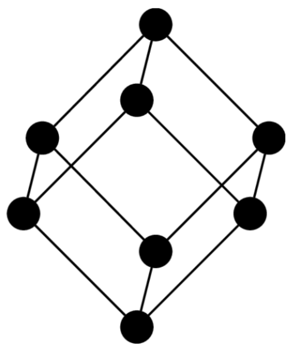

Figure 8: The 3-cube graph

It looks like a three dimensional cube, right? You might think that you can't possibly draw this graph in two dimensions without having any of the edges cross each other. But in fact you can! Both the graphs below are isomorphic to the 3-cube graph.

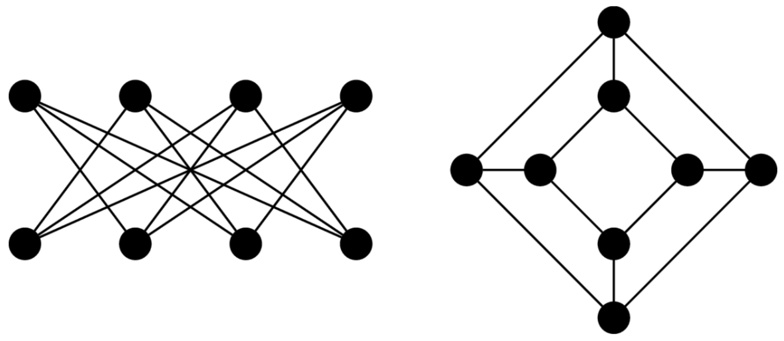

Figure 9: Two other representations of the 3-cube graph

You can think about moving the vertices in the three cube, without adding or removing any edges in order to go from the drawing in figure 8 to the drawing on the right of figure 9. You are allowed to make the edges longer or shorter. In fact it turns out that the graph in figure 8 and the two graphs in figure 9 are all isomorphic to each other. Thus the visual representation of a graph, while very useful, can be somewhat misleading. Indeed the study of which graphs can be drawn in what sorts of ways, in particular which graphs can be drawn on a 2D sheet of paper without any crossing edges, is an interesting topic of study, but we won't delve much further into it in this class.

We now intuitively understand that just because the graph isn't exactly the same (i.e., the vertices are named something differently) doesn't mean that we don't want to consider them equivalent. The notion of graph isomorphism captures the idea that two graphs are basically the same, just with different names or drawings. Now let's formalise our intuition about graph isomorphism.

Definition 10 (graph isomorphism)**.** *An isomorphism between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a bijection $f : V_1 \to V_2$ which preserves adjacency. That is, for all vertices $u, v \in V_1, u$ is adjacent to v in G if and only if $f(u)$ and $f(v)$ are adjacent in G_2. Two graphs G_1 and G_2 are isomorphic if there exists an isomorphism between them, denoted $G_1 \cong G_2$.*

In other words, we consider two graphs to be the "same" or equivalent if we can map every vertex to the other graph such that the adjacency relationship is preserved. In practice, this amounts to a "renaming" of the vertices, which is what we typically mean when we say that two objects are isomorphic. Of course, this really means that the edges are preserved.

Consider the following graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ from Figure 9. One has a huge number of crossings and the other one has 0 crossing edges. We will show that they are isomorphic to each other.

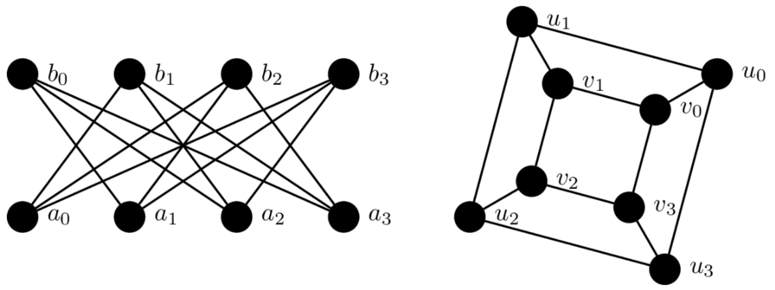

Figure 10: Two isomorphic graphs

We will show that these two graphs are isomorphic by defining a bijective function $f : V_1 \to V_2$ which preserves adjacency:

$$\begin{array}{llll} f(a_0) = u_0 & f(a_1) = v_3 & f(a_2) = u_2 & f(a_3) = v_1 \\ f(b_0) = v_2 & f(b_1) = u_1 & f(b_2) = v_0 & f(b_3) = u_3 \end{array}$$

This is a bijection, since every vertex in V_1 is paired with a vertex in V_2. We then verify that this bijection preserves adjacency by checking each edge under the bijection:

$$\begin{array}{lll} (f(a_0), f(b_1)) = (u_0, u_1) & (f(a_0), f(b_2)) = (u_0, v_0) & (f(a_0), f(b_3)) = (u_0, u_3) \\ (f(a_1), f(b_0)) = (v_3, v_2) & (f(a_1), f(b_2)) = (v_3, v_0) & (f(a_1), f(b_3)) = (v_3, u_3) \\ (f(a_2), f(b_0)) = (u_2, v_2) & (f(a_2), f(b_1)) = (u_2, u_1) & (f(a_2), f(b_3)) = (u_2, u_3) \\ (f(a_3), f(b_0)) = (v_1, v_2) & (f(a_3), f(b_1)) = (v_1, u_1) & (f(a_3), f(b_2)) = (v_1, v_0) \end{array}$$

So an obvious and fundamental problem that arises from this definition is: Given two graphs G_1 and G_2, are they isomorphic? The obvious answer is that we just need to come up with an isomorphism or show that one doesn't exist. Unfortunately, there are $n!$ possible such mappings we would need to check.

Indeed solving this problem is computationally hard. If we could solve this problem fast for any arbitrary pair of graphs, it would be really useful, because we often want to check if two networks really have the same underlying graph structure. For example you might be looking at two large molecules prepared in a lab and want to know if they are really the same molecule (in terms of having the same atoms linked by the same network of inter atomic bonds).

What we can do to help a bit with this search is to look at some graph isomorphism invariants. These are properties that remain the same between two graphs that are isomorphic. An obvious invariant is the number of edges or number of vertices of a graph. Another invariant is the number of vertices having any given degree. For example, if one graph has 17 vertices with degree 20 and another has only 15 vertices of degree exactly 20, then they simply cannot be isomorphic. Try to formally explain why this is so. You can have all sorts ofisomorphism invariants. For example you could look at all degree 7 vertices in a graph and ask how many are adjacent to each other. This too is an invariant! Again try to reason why this is so. However, several properties of a representation of a graph, such as how many of its edges cross one another when it is drawn on a 2D sheet of paper, are not isomorphism invariants, as we have just seen in Figure 10.

However, while it is true that a property that is an isomorphism invariant must be the same for two isomorphic graphs, that does not mean that just because two graphs have the same value of the invariant then they must be isomorphic! For example isomorphic graphs have the same number of vertices, but just because two graphs have the same number of vertices does not mean that the graphs are isomorphic.

Now try to solve the problems in the handout.

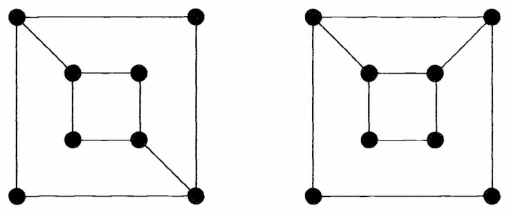

Figure 11: Two non-isomorphic graphs

These graphs are not isomorphic. You can look at the degree 3 vertices in each of them. In the right graph each of these is adjacent to two other such vertices, but on the left graph each degree 3 vertex is adjacent to only one other such vertex. But an isomorphism would have to map a degree 3 vertex to a degree 3 vertex only. Argue that if there is an isomorphism between the two graphs shown above, then this leads to a contradiction.

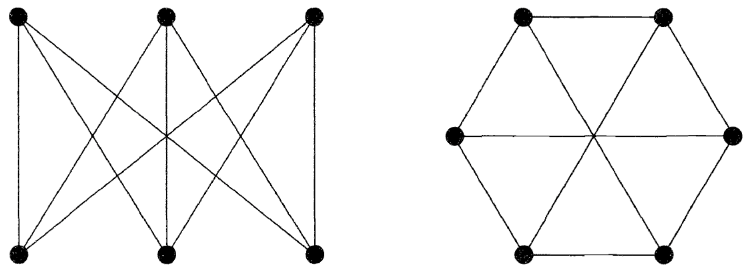

Figure 12: Two isomorphic graphs

The above graphs however are isomorphic! Try to come up with an explicit isomorphism between them. Start with one of the vertices and then map where it's neighbours go. Alternatively notice that each graph can be completely described as two sets of 3 vertices each, with every vertex in one set connected to all vertices in the other set. This is what we call the complete bipartite graph between two sets of 3 vertices. We will talk a little more about bipartite graphs in a short while.

3 Coloring graphs

Typically when we talk about coloring a graph, we are referring to coloring the vertices of the graph such that no two vertices that have the same color are adjacent to one another. Why might we care about coloring a graph using the least possible number of colors? Well, we can consider a graph that expresses constraints of some kind. For example suppose we want to schedule exams for every class offered by the MPCS. Of course we can't schedule exams for two classes at the same time if there are any students who are taking both of those classes. We can consider a graph where each vertex is a class and there are edges between classes if there are any students who take both of those classes. If you can color the vertices of the graph using as few number of colors as possible, you have found the minimum number of different exam times that are needed in your exam schedule!

Definition 11 (k-colourability)**.** *A k-colouring of a graph is a function $c : V \to \{1, 2, \ldots, k\}$ such that if $(u, v) \in E$, then $c(u) \neq c(v)$. That is, adjacent vertices receive different colours. A graph is k colourable if there exists a k-colouring for G.*

Definition 12 (chromatic number)**.** *A graph G has chromatic number k if it is k-colourable, but not $k-1$ colourable. That is, if k is the least number of colours needed to color its vertices such that adjacent vertices receive different colors. This is often denoted as $\chi(G) = k$.*

We will use k-colourability as an invariant to show that two graphs are not isomorphic. Consider the following two graphs, G_1 and G_2.

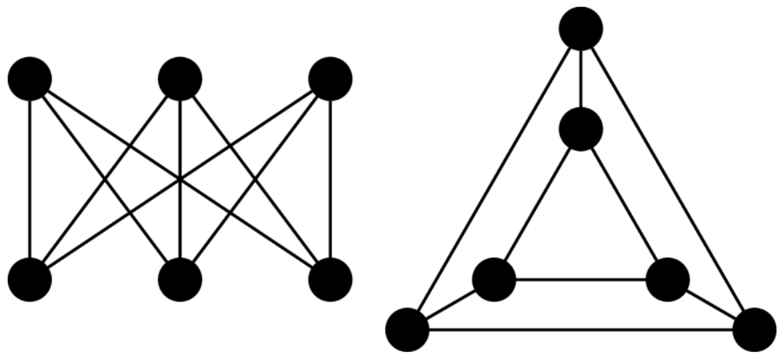

Figure 13: G_1 on the left and G_2 on the right

First, we observe that both graphs have 6 vertices and and 9 edges and both graphs are 3 regular. However, we will show that G_1 is 2 -colourable and G_2 is not. To see that G_1 is 2 colourable, we observe that the vertices along the top row can be assigned the same colour because they are not adjacent to each other. Similarly, the vertices along the bottom can be assigned a second colour.

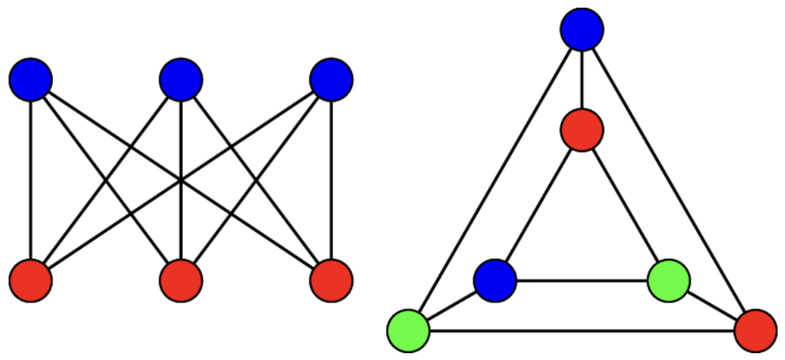

Figure 14: G_1 on the left and G_2 on the right

However, G_2 contains two triangles and by definition, the vertices belonging to the same triangle must be coloured with 3 different colours. Since G_2 is not 2-colourable, the adjacency relationship among the vertices is not preserved and so G_1 and G_2 cannot be isomorphic.

In fact G_1 in Figure 14 is another special kind of graph: a bipartite graph. A graph is called bipartite if its vertices can be partitioned into two disjoint sets A and B such that each vertex is in exactly one of the sets and all the edges are between vertices that are in different sets. The following theorem provides a handy characterisation of bipartite graphs. The proof is left as an (easy) exercise.

Theorem 2. *A graph is bipartite if and only if it is 2 colourable.*

A complete bipartite graph $K_{m,n}$ is one where you have m vertices in on set A, n vertices in another disjoint set B, and every pair of vertices (i, j) where i and j are not in the same set, are connected by an edge. There are no edges between vertices in the same set. G_1 in figure 14 is a complete bipartite graph with 3 vertices in set A and 3 vertices in set B.

4 Walks, Paths and Cycles

One basic property you can ask when you look at a graph is whether you can get from some given vertex u to another given vertex v. In other words is there a path from u to v? If so, you might be interested in the shortest path between the vertices. Indeed this sort of question is often of great relevance when trying to solve problems where we are modelling some kind of network by a graph. To discuss these sorts of questions effectively, we will define a notion of paths in a graph formally. Note that the specifics of these definitions vary slightly from one source to the next, what we call paths, might be called walks in other sources and so on. So make sure you keep the specific definitions in mind, especially if you read about this material from other sources.

Definition 13 (walk)**.** *A walk of length k is a sequence of $k + 1$ vertices $v_0v_1v_2 \ldots v_k$ where v_i and v_{i+1} are adjacent. It consists of k edges.*

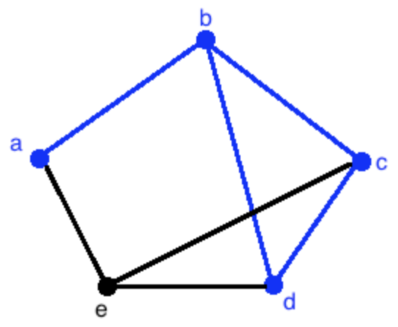

Figure 15: *abcdbc* is a walk

Definition 14 (path)**.** *A path is a walk with no repeated vertices.*

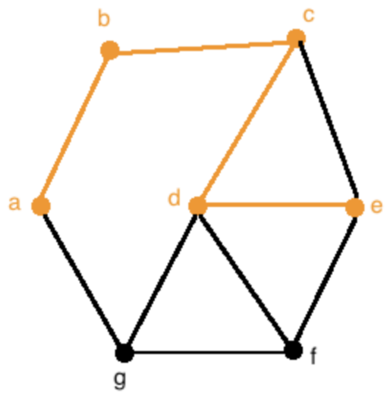

Figure 16: *abcde* is a path

Definition 15 (cycle)**.** *A cycle of length k is a path from v_0 to v_{k-1}: $v_0, v_1, v_2 \dots v_{k-1}$ along with the edge (v_{k-1}, v_0).*

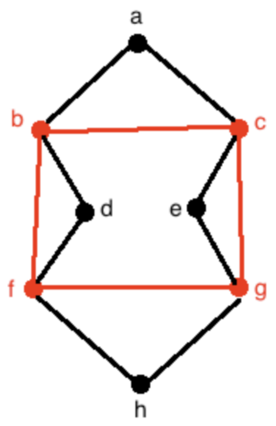

Figure 17: $bcgf$ is a cycle

Note that we don't count cycles of length 2 as cycles. That is two adjacent vertices v_0 and v_1 do not themselves form a cycle. A cycle has to have length at least 3. A graph that has no cycles is called an acyclic graph.

We have some common sense results about walks, paths and cycles. The proofs are easy enough, we go over them briefly to give some examples of how to write proofs demonstrating facts about graphs.

Theorem 3. *If there is a walk in a graph from vertex u to vertex v, then there is a path in the graph from u to v.*

Proof. Consider the walk $u = v_0 v_1 \cdots v_k = v$ from u to v of minimum length k. Suppose that a vertex is repeated. In other words, there exist, i, j with $i < j$ such that $v_i = v_j$. Then we can create a shorter walk

$$(u = v_0)v_1 \cdots v_{i-1} v_i v_{j+1} v_{j+2} \cdots (v_k = v)$$

since $(v_i, v_{j+1}) = (v_j, v_{j+1})$. Then this contradicts the minimality of the length of our original walk. So there are no repeated vertices in our walk and therefore, it is a path by definition. $\square$

Theorem 4. *If a graph has two distinct paths from a vertex u to a different vertex v then the graph has a cycle.*

Note that we are not assuming that the paths have no overlap; they are not necessarily disjoint. But they are distinct, i.e., they are not completely overlapping.

Proof. Consider two paths P_1 and P_2 between vertices u and v. We can write $P_1 = a_0 a_1 \cdots a_m$ and $P_2 = b_0 b_1 \cdots b_n$, where $a_0 = b_0 = u$ and $a_m = b_n = v$, with m not necessarily equal to n. Since the two paths are distinct, without loss of generality there is an edge that is in P_1 that is not in P_2. Let (a_i, a_{i+1}) be this edge.

Consider the graph $G - (a_i, a_{i+1})$. Now, consider the walk

$$a_i a_{i-1} \cdots (a_0 = b_0) b_1 \cdots (b_n = a_m) a_{m-1} \cdots a_{i+1}.$$

Call this walk w. Then w is a walk from a_i to a_{i+1} which does not use the edge (a_i, a_{i+1}). So there is a path p from a_i to a_{i+1} that does not involve the edge (a_i, a_{i+1}). Thus p, a_{i+1}, a_i is a cycle. So the edge (a_i, a_{i+1}) participates in a cycle. Therefore, G contains a cycle. $\square$

5 Connectedness in graphs

Some graphs have the property that you can get from anywhere to anywhere else in the graph via some path. These graphs are the ones we called connected. However graphs need not always be connected. They may consist of some k pieces such that you can get from anywhere to anywhere else within a piece, but there is no path between vertices in different pieces. We call these pieces the connected components of a graph. A graph with n vertices may have between 1 and n connected components.

To define these formally, we first introduce the notion of a subgraph.

Definition 16 (subgraph). *A subgraph $G' = (V', E')$ of $G = (V, E)$ is a graph whose vertex set V' is a subset of V and whose edge set E' is a subset of E. E' of course can only have edges between vertices in V'.*

A subgraph G' is a proper subgraph of G, if G atleast has some edge that is not in G'. Given a graph G, we can look at a subset X of its vertices and just look at the edges involving those vertices. This vertex and edge set also forms a subgraph of G, which we call the subgraph of G *induced* by the vertex set X.

Definition 17 (connected graph)**.** *A graph G is connected if for any distinct vertices u and v in G there is a path between u and v.*

Definition 18 (connected component)**.** *A connected component of a graph G is a connected subgraph G' of G that is not a proper subgraph of any another connected subgraph of G.*

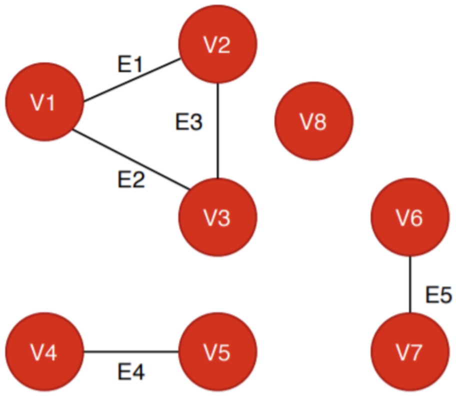

Figure 18: A graph G with 4 connected components

The above graph has 4 connected components. Notice that the subgraph v_1, v_2 with just the edge (v_1, v_2) is not itself an entire connected component since it is a proper subgraph of a larger connected subgraph of G(the one induced by $\{v_1, v_2, v_3\}$ in G).

Theorem 5. *Any graph consists of (possibly only 1) disjoint connected components.*

Proof. (Sketch) Pick a vertex u of the graph. Add all the vertices that are reachable from u into one connected component. Once you are done, pick one of the remaining vertices and start adding all the vertices reachable from it to its connected component. This process will end at some point since the graph is finite.

Notice that these components we generated are indeed the connected components since they are the largest possible connected subgraphs, in the sense that we cannot add anything to them. Why is this? Well suppose some vertex x was supposed to be in some connected component that we built starting with vertex v, because x is reachable from v. If x had not already been put into another connected component, then we would have added it to the component built starting from v. But if x was already in another connected component, let's say the one started with u, then x would be reachable from u and so x cannot possibly be reachable from v since otherwise there would be a path between u and v, and so we would have added v to the component started with u itself.

Similar reasoning tells us that all the components must be disjoint, because if two connected components share a vertex x, then you can get from any vertex in one component to any vertex in any other component by travelling through x. Thus these would not be two distinct connected components but rather the same connected component. □

6 Trees

Trees are a type of graph that you will certainly come across a good deal in computer science. As computer scientists we often think about trees as having a root vertex at the top and then they keep branching more and more as we go downwards. In the next lecture, we will see an inductive definition of one particular kind of tree, binary trees, that we see a lot of in computer science.

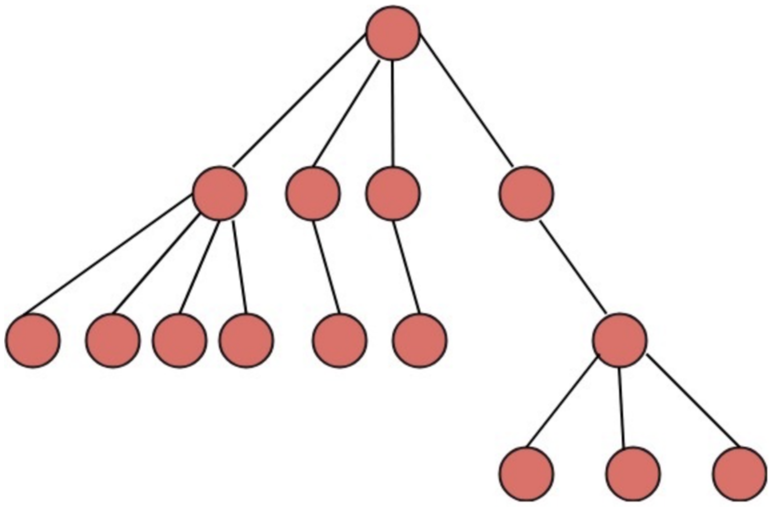

Figure 19: A typical tree in computer science.

However trees in general can be defined without any direct reference to a root or topmost node of any kind. In a sense, a tree is a minimally connected graph on n vertices that still consists of just 1 connected component. It contains exactly $n-1$ edges, and you can show that if a graph on n vertices has less than $n-1$ edges, then it cannot possibly be connected. Naturally enough, a graph that consists of a bunch of connected components, each of which is a tree, is called a forest! Formally, we can define a tree as follows.

Definition 19 (Tree.)**.** *A tree is a connected graph which does not contain any cycles.*

Here is an example of a tree:

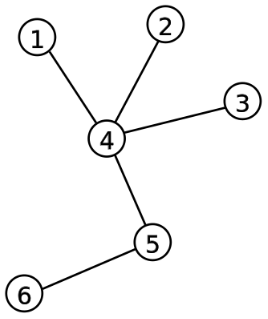

Figure 20: A tree with 6 vertices and 5 edges.

In fact, given a tree, you could consider any of its vertices to be the root, and you would then be able to draw a rooted tree with the root at the top, then the "children" of the root below that and so on. For example you can try placing node 4 as the root and then drawing this tree like Figure 19. We will see a little more of rooted trees in the next lecture.

We will now state a few facts about trees.

Theorem 6. *A graph G is a tree if and only if there is a unique path between any two of it's vertices.*

Proof. Assume that there is a unique path between any two of G's vertices. There is a path between any two vertices of G, so G is connected. Further, suppose G had a cycle $u_0, u_1, \dots u_k, u_0$. Then there would be two paths from u_0 to u_k, one would be the path $u_0, u_1 \dots u_k$ and the other would just be the edge u_0, u_k. Thus we have shown that if there is a unique path between any two of the graph's vertices then it must be connected and acyclic, i.e. it must be a tree.

Now to prove the other direction. Suppose G is a tree. Since it is connected, there must be a path between any two vertices u and v of G. G does not have any cycles. But if there were two distinct paths p_1 and p_2 between u and v then, by Theorem 4, there is a cycle. Contradiction. So if G is a tree, then there is a unique path between any two of its vertices. □

Theorem 7. *If you add an edge between any two non-adjacent vertices of a tree, you will end up with a graph which has a cycle.*

Proof. Consider any two non adjacent vertices u and v of the tree T. There is already a path from u to v, call it $p = u, a_1, a_2 \dots a_k, v$. If you add the edge (u, v) then v, p which is $v, u, a_1, a_2 \dots a_k, v$ is a cycle. □

Theorem 8. *A tree on n vertices has exactly $n - 1$ edges.*

In fact a connected graph on n vertices is a tree precisely when it has exactly $n - 1$ edges. This is a rather useful characterization, whose proof we will discuss in the next lecture.

Remember, we can think of a tree as a minimal connected graph. Any other connected graph can be thought of as a tree with some more edges added to it. This leads to the notion of a spanning tree.

Definition 20 (Spanning tree)**.** *A spanning tree of a graph G is a subgraph of G that has all the vertices of G and is a tree.*

Here is an example of a graph (that looks like a grid) with one of the spanning trees highlighted. Every connected graph has to have a spanning tree.

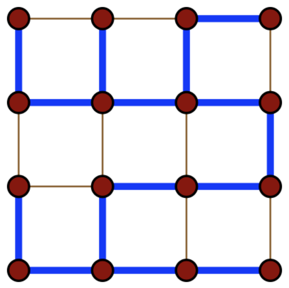

Figure 21: A graph and one of its spanning trees

Theorem 9. *Every connected graph has a spanning tree.*

Sketch. The proof is by induction on the number of edges. If G is connected and has zero edges, it is a single vertex, so G is already a tree and is its own spanning tree.

Now suppose G has $m \geq 1$ edges. If G is a tree, it is its own spanning tree. Otherwise, G contains a cycle; remove one edge of this cycle. The resulting graph G' is still connected (prove this: i.e. prove that if you have a connected graph with a cycle and you remove one edge from the cycle, it still remains connected) and has fewer edges, so it has a spanning tree (by inductive hypothesis); this is also a spanning tree for G (since G' has the same vertices as G).

□

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

Of course a graph could in general have a very large number of spanning trees. One important computational question, for which you will see some algorithms in an algorithms course, is how to find a low cost spanning tree of a given graph (assuming that the edges of the graph have some associated costs.) This has natural applications if we want to build a cheap network of connections in order to be able to connect all the vertices to each other in a way that minimizes total cost.