1 Graph Theory: An Introduction

One of the fundamental ideas in computer science is the notion of *abstraction*: capturing the essence or the core of some complex situation by a simple model. Some of the largest and most complex entities we might deal with include the internet, the brain, maps, and social networks. In each case, there is an underlying "network" or *graph* that captures the important features that help us understand these entities more deeply. In the case of the internet, this network or graph specifies how web pages link to one another. In the case of the brain, it is the interconnection network between neurons. We can describe these ideas in the beautiful framework of *graph theory*, which is the subject of this lecture.

Remarkably, graph theory has its origins in a simple evening pastime of the residents of Königsberg, Prussia (nowadays Kaliningrad, Russia) a few centuries ago. Through their city ran the Pregel river, depicted on the left in Figure 1 below, separating Königsberg into two banks A and D and islands B and C. Connecting the islands to the mainland were seven bridges. As the residents of the city took their evening walks, many would try to solve the challenge of picking a route that would cross each of the seven bridges precisely once and return to the starting point.

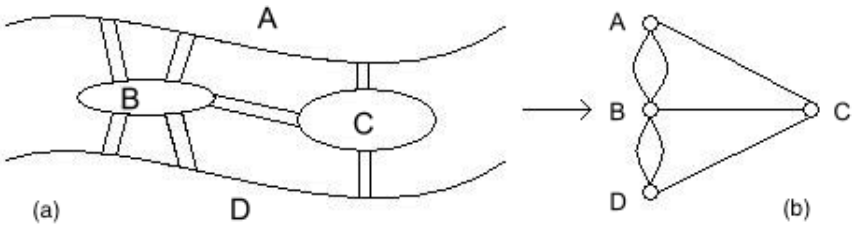


Figure 1: (Left) The city of Königsberg. (Right) The (multi-)graph modeling the bridge connections in Königsberg.

In 1736, the brilliant mathematician Leonhard Euler proved this task to be impossible. How did he do it? The key is to realize that for the purpose of choosing such a route, Figure 1a can be replaced with Figure 1b, where each land mass A, B, C, and D is replaced by a small circle, and each bridge by a line segment. With this abstraction in place, the task of choosing a route can be restated as follows: trace through all the line segments and return to the starting point without lifting the pen, and without traversing any line segment more than once. The proof of impossibility is simple. Under these tracing rules, the pen must enter each small circle as many times as it exits it, and therefore the number of line segments incident to that circle must be even. But in Figure 1b, each circle has an odd number of line segments incident to it, so it is impossible to carry out such a tracing. Actually Euler did more. He gave a precise condition under which the tracing can be carried out. For this reason, Euler is generally hailed as the inventor of graph theory.

1.1 Formal definitions

Formally, a (undirected) graph is defined by a set of vertices V and a set of edges E. The vertices correspond to the little circles in Figure 1 above, and the edges correspond to the line segments between the vertices.

In Figure 1, $V = \{A,B,C,D\}$ and $E = \{\{A,B\},\{A,B\},\{A,C\},\{B,C\},\{B,D\},\{B,D\},\{C,D\}\}$. However, note that here E is a multiset (a set where an element can appear multiple times). This is because in the Königsberg example there are multiple bridges between a pair of banks. We will generally not consider such a situation of multiple edges between a single pair of vertices, so in our definition, we require E to be a set, not a multi-set. What this means is that between any pair of vertices there is either 0 or 1 edge. If there are multiple edges between a pair of vertices, then we collapse them into a single edge.

More generally, we can also define a directed graph. If an edge in an undirected graph represents a street, then an edge in a directed graph represents a one-way street. To make this formal, let V be a set denoting the vertices of a graph G. For example, we can have $V = \{1,2,3,4\}$. Then, the set of (directed) edges E is a subset of $V \times V$, i.e. $E \subseteq V \times V$. (Recall here that $U \times V$ denotes the Cartesian product of sets U and V, defined as $U \times V = \{(u,v) : u \in U \text{ and } v \in V\}$.) Continuing with our example, let $E = \{(1,2),(1,3),(1,4)\}$. Then, the corresponding graph is given by G_1 below.

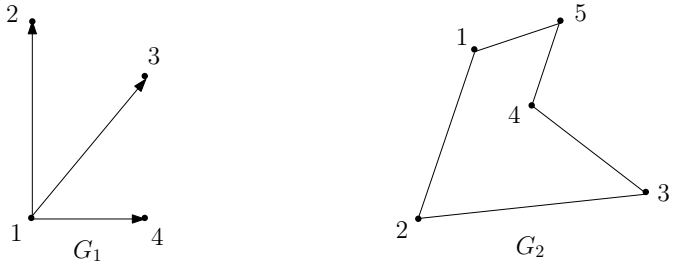


Figure 2: Examples of directed and undirected graphs, respectively.

Note that each edge in G_1 has a *direction* specified by an arrow; thus, for example, $(1,2) \in E$ but $(2,1) \notin E$. Such graphs are useful in modeling one-way relationships, such as one-way streets between two locations, and are called *directed*. On the other hand, if each edge goes in both directions, i.e., $(u,v) \in E$ iff $(v,u) \in E$, then we call the graph *undirected*. For undirected graphs we drop the ordered pair notation for edges, and simply denote the edge between u and v by the set $\{u,v\}$. Undirected graphs model relationships such as two-way streets between locations naturally, and an example is given by G_2 above. For simplicity, we omit the arrowheads when drawing edges in undirected graphs. We conclude that a graph is thus formally specified as an ordered pair $G = (V,E)$, where V is the vertex set and E is the edge set.

Sanity check! What are the vertex and edge sets V and E for graph G_2?

Let us continue our discussion with a working example from *social networks*, an area in which graph theory plays a fundamental role. Suppose you wish to model a social network in which vertices correspond to people, and edges correspond to the following relationship between people: We say Alex *recognizes* Bridget if Alex knows who Bridget is, but Bridget does not know who Alex is. If, on the other hand, Alex knows Bridget and Bridget knows Alex, then we say they *know each other*.

Sanity check! Suppose first that an edge between two people (say) Alex and Bridget means that Alex recognizes Bridget; would you use a directed or undirected graph for this? How about if an edge instead means Alex and Bridget know each other? (Answer: directed and undirected, respectively.)

Moving on with our example, we say that edge $e = \{u,v\}$ (or $e = (u,v)$) is *incident* on vertices u and v, and that u and v are *neighbors* or *adjacent*. If G is undirected, then the *degree* of vertex $u \in V$ is the number of edges incident to u, i.e., $degree(u) = |\{v \in V : \{u,v\} \in E\}|$. A vertex u whose degree is 0 is called an

isolated vertex, since there is no edge which connects u to the rest of the graph.

Sanity check! What does the degree of a vertex represent in our *undirected* social network in which an edge $\{u,v\}$ means u and v know each other? How should we interpret an isolated vertex?

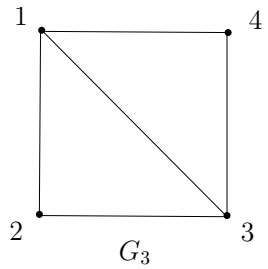
A directed graph, on the other hand, has two different notions of degree due to the directions on the edges. Specifically, the *in-degree* of a vertex u is the number of edges from other vertices to u, and the *out-degree* of u is the number of edges from u to other vertices.

Sanity check! What do the in-degree and out-degree of a vertex represent in our *directed* social network in which an edge (u,v) means u recognizes v?

Finally, our definition of a graph thus far allows edges of the form $\{u,u\}$ (or (u,u)), i.e., a *self-loop*. In our social network, however, this gives us no interesting information (it means that person A recognizes him/herself!). Thus, here and in general in these notes, we shall assume that our graphs have no self-loops, unless stated otherwise. We shall also not allow multiple edges between a pair of vertices (unlike the case of the Seven Bridges of Königsberg).

1.2 Paths, walks, and cycles

Let $G = (V,E)$ be an undirected graph. A *path* in G is a sequence of edges $\{v_1,v_2\},\{v_2,v_3\},...,\{v_{n-1},v_n\}$. In this case we say that there is a path *between* v_1 *and* v_n. For example, suppose the graph G_3 below models a residential neighborhood in which each vertex corresponds to a house, and two houses u and v are neighbors if there exists a direct road from u to v.



G_3

Sanity check! What is the shortest path from house 1 to house 3 in G_3? How about the longest path, assuming no house is visited twice?

Usually, we assume a path is *simple*, meaning $v_1,\ldots,v_n$ are distinct. This makes complete sense in our housing example G_3; if you wanted drive from house 1 to 3 via house 2, why would you visit house 2 more than once? A *cycle* (or *circuit*) is a sequence of edges $\{v_1,v_2\},\{v_2,v_3\},\ldots,\{v_{n-2},v_{n-1}\},\{v_{n-1},v_n\},\{v_n,v_1\}$, where $v_1,\ldots,v_n$ are distinct (i.e., a cycle is a path which starts and ends on the same vertex v_1).

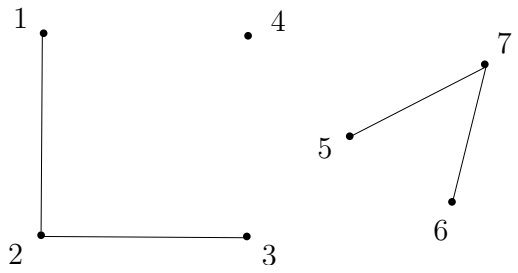
Sanity check! Give a cycle starting at house 1 in G_3.

Suppose now that your aim is not to go from 1 to 3 as quickly as possible, but to take a leisurely stroll from 1 to 3 via the sequence $\{1,2\},\{2,1\},\{1,4\},\{4,3\}$. A sequence of edges with repeated vertices, such as this

one, is called a *walk* from 1 to 3. Analogous to the relationship between paths and cycles, a *tour* is a walk which starts and ends at the same vertex. For example, $\{1,2\},\{2,3\},\{3,1\}$ is a tour.

1.3 Connectivity

Much of what we discuss in this note revolves around the notion of connectivity. A graph is said to be *connected* if there is a path between any two distinct vertices. For example, our residential network G_3 above is connected, since one can drive from any house to any other house via *some* sequence of direct roads. On the other hand, the network below is *disconnected*.



Sanity check! What would a disconnected vertex represent in our residential network? Why would you not want to design a neighborhood this way?

Note that *any* graph (even a disconnected one) always consists of a collection of *connected components*, i.e., sets $V_1,\ldots,V_k$ of vertices, such that all vertices in a set V_i are connected. For example, the graph above is not connected, but nevertheless consists of three connected components: $V_1 = \{1,2,3\}$, $V_2 = \{4\}$, and $V_3 = \{5,6,7\}$.

2 Revisiting the Seven Bridges of Koenigsberg: Eulerian Tours

With a formal underpinning in graph theory under our belts, we are ready to revisit the Seven Bridges of Königsberg. What exactly is this problem asking? It says: Given a graph G (in this case, G is an abstraction of Königsberg), is there a walk in G that uses each edge exactly once? We call any such walk in a graph an *Eulerian walk*. (In contrast, by definition a walk can normally visit each edge or vertex as many times as desired.) Moreover, if an Eulerian walk is closed, i.e., it ends at its starting point, then it is called an *Eulerian tour*. Thus, the Seven Bridges of Königsberg asks: Given a graph G, does it have an Eulerian tour? We now give a precise characterization of this in terms of simpler properties of the graph G. For this, define an *even degree* graph as a graph in which all vertices have even degree.

Theorem 5.1 (Euler's Theorem (1736))**.** *An undirected graph $G = (V,E)$ has an Eulerian tour iff G is even degree, and connected (except possibly for isolated vertices).*

Proof. To prove this, we must establish two directions: if, and only if.

Only if. We give a direct proof for the forward direction, i.e., if G has an Eulerian tour, then it is connected and has even degree. Assume that G has an Eulerian tour. This means every vertex that has an edge adjacent to it (i.e., every non-isolated vertex) must lie on the tour, and is therefore connected with all other vertices on the tour. This proves that the graph is connected (except for isolated vertices).

Next, we prove that each vertex has even degree by showing that all edges incident to a vertex can be paired up. Notice that every time the tour enters a vertex along an edge it exits along a different edge. We can pair these two edges up (they are never again traversed in the tour). The only exception is the start vertex, where the first edge leaving it cannot be paired in this way. But notice that by definition, the tour necessarily ends at the start vertex. Therefore, we can pair the first edge with the last edge entering the start vertex. So all edges adjacent to any vertex of the tour can be paired up, and therefore each vertex has even degree.

If. We give a recursive algorithm for finding an Eulerian tour, and prove by induction that it correctly outputs an Eulerian tour.

We start with a useful subroutine, FINDTOUR(G,s), which finds a tour (not necessarily Eulerian) in G. FINDTOUR is very simple: it just starts walking from a vertex $s \in V$, at each step choosing any untraversed edge incident to the current vertex, until it gets stuck because there is no more adjacent untraversed edge. We now prove that FINDTOUR must in fact get stuck at the original vertex s.

Claim: FINDTOUR(G,s) must get stuck at s.

Proof of claim: An easy proof by induction on the length of the walk shows that when FINDTOUR enters any vertex $v \neq s$, it will have traversed an odd number of edges incident to v, while when it enters s it will have traversed an even number of edges incident to s. Since every vertex in G has even degree, this means every time it enters $v \neq s$, there is at least one untraversed edge incident to v, and therefore the walk cannot get stuck. So the only vertex it can get stuck at is s. The formal proof is left as an exercise. □

The algorithm FINDTOUR(G,s) returns the tour it has traveled when it gets stuck at s. Note that while FINDTOUR(G,s) always succeeds in finding a tour, it does not always return an Eulerian tour.

We now give a recursive algorithm EULER(G,s) that outputs an Eulerian tour starting and ending at s. EULER(G,s) invokes another subroutine SPLICE$(T,T_1,\ldots,T_k)$ which takes as input a number of edge disjoint tours $T,T_1,\ldots,T_k$ ($k \geq 1$), with the condition that the tour T intersects each of the tours $T_1,\ldots,T_k$ (i.e., T shares a vertex with each of the T_i's). The procedure SPLICE$(T,T_1,\ldots,T_k)$ outputs a single tour T' that traverses all the edges in $T,T_1,\ldots,T_k$, i.e., it splices together all the tours. The combined tour T' is obtained by traversing the edges of T, and whenever it reaches a vertex s_i that intersects another tour T_i, it takes a detour to traverse T_i from s_i back to s_i again, and only then it continues traversing T.

The algorithm EULER(G,s) is given as follows:

Function EULER(G,s)
$T =$ FINDTOUR(G,s)
Let $G_1,\ldots,G_k$ be the connected components when the edges in T are removed from G, and let s_i be the first vertex in T that intersects G_i
Output SPLICE$(T,$EULER$(G_1,s_1),\ldots,$EULER$(G_k,s_k))$
end EULER

We prove by induction on the size of G that EULER(G,s) outputs an Eulerian Tour in G. The same proof works regardless of whether we think of size as number of vertices or number of edges. For concreteness, here we use number of edges m of G.

Base case: $m = 0$, which means G is empty (it has no edges), so there is no tour to find.

Induction hypothesis: EULER(G,s) outputs an Eulerian Tour in G for any even degree, connected graph with at most $m \geq 0$ edges.

Induction step: Suppose G has $m+1$ edges. Recall that $T =$ FINDTOUR(G,s) is a tour, and therefore has even degree at every vertex. When we remove the edges of T from G, we are therefore left with an even degree graph with less than m edges, but it might be disconnected. Let $G_1,\ldots,G_k$ be the connected

components. Each such connected component has even degree and is connected (up to isolated vertices). Moreover, T intersects each of the G_i, and as we traverse T there is a first vertex where it intersects G_i. Call it s_i. By the induction hypothesis $\text{EULER}(G_i, s)$ outputs an Eulerian tour of G_i. Now by the definition of SPLICE, it splices the individual tours together into one large tour whose union is all the edges of G, hence an Eulerian tour. □

Sanity check! Why does Theorem 5.1 imply the answer to the Seven Bridges of Königsberg is no?

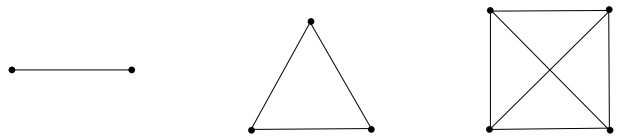
---

3 Special Kinds of Graphs

In this section, we consider a few kinds of graphs important enough to be given their own names.

3.1 Complete graphs

We start with the simplest class of graphs, the *complete* graphs. Why are such graphs called complete? Because they contain the *maximum* number of edges possible. In other words, in an undirected complete graph, every pair of (distinct) vertices u and v are connected by an edge $\{u, v\}$. For example, below we have complete graphs on $n = 2, 3, 4$ vertices, respectively.



K_2 K_3 K_4

Here, the notation K_n denotes the *unique* complete graph on n vertices. Formally, we can write $K_n = (V, E)$ for $|V| = n$ and $E = \big\{\{v_i, v_j\} \mid v_i \neq v_j \text{ and } v_i, v_j \in V\big\}$.

Sanity check!

1. Can you draw K_5, the complete graph on $n = 5$ vertices?
2. What is the degree of every vertex in K_n?

Exercise. How many edges are there in K_n? (Answer: $n(n-1)/2$.) Verify that the K_5 you drew above has this many edges.

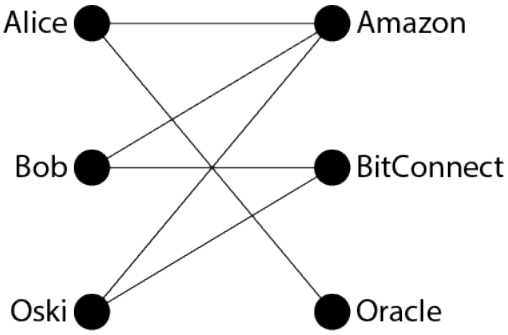
A complete graph is special in that each vertex is neighbors with every other vertex. Thus, such a graph is very "strongly connected" in that a large number of edges must be removed before we disconnect the graph into two components. Why might this be a good property to have (say) in a communications network, where vertices correspond to mainframes, and edges correspond to communications channels?

Sanity check! What is the minimum number of edges which must be removed from K_n to obtain an isolated vertex?

Finally, we can also discuss complete graphs for *directed* graphs, which are defined as you might expect: For any pair of vertices u and v, both $(u,v),(v,u) \in E$.

3.2 Bipartite Graphs

The next class of graphs looks at a very different kind of structure. In a *bipartite graph*, our vertex set is divided into two halves. Each half is allowed to connect to the other, but cannot connect to itself. For example, if we have a set of CS 70 students and a set of companies, we might model which companies each student would be willing to work for as a bipartite graph:



Formally, we have that a graph $G = (V,E)$ is bipartite if V can be split into L and R (with $L \cap R = \emptyset$ and $L \cup R = V$) such that every edge in E has one endpoint in L and one endpoint in R.

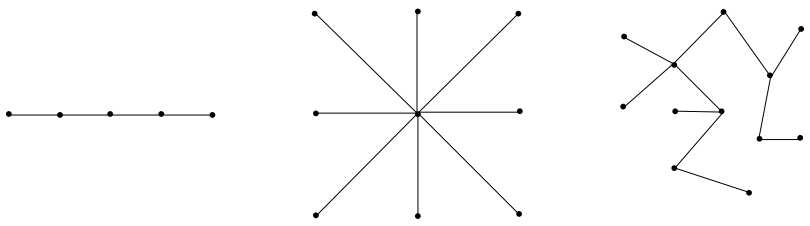
Sanity check! Suppose we have a bipartite graph G with $|L| = |R| = n$. What is the largest number of edges G can have? What is the smallest number?

3.3 Trees

If complete graphs are "maximally connected," then trees are the opposite: Removing just a single edge disconnects the graph! Formally, there are a number of equivalent definitions of when a graph $G = (V,E)$ is a tree, including:

1. G is connected and contains no cycles.
2. G is connected and has $n-1$ edges (where $n = |V|$).
3. G is connected, and the removal of any single edge disconnects G.
4. G has no cycles, and the addition of any single edge creates a cycle.

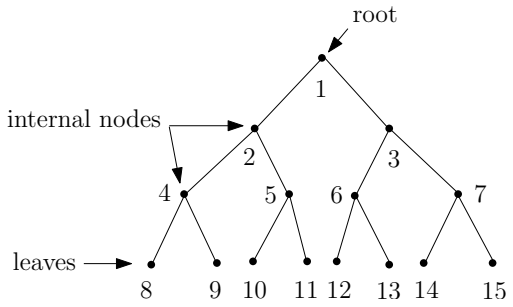
Here are three examples of trees:



Sanity check!

1. Convince yourself that the three graphs above satisfy all four equivalent definitions of a tree.
2. Give an example of a graph which is *not* a tree.

Why would we want to study such funny-looking graphs? One reason is that many graph-theoretical problems which are computationally intractable on arbitrary graphs, such as the Maximum Cut problem, are easy to solve on trees. Another reason is that they model many types of natural relationships between objects. To demonstrate, we now introduce the concept of a *rooted* tree, an example of which is given below.



In a rooted tree, there is a designated node called the *root*, which we think of as sitting at the top of the tree. The bottom-most nodes are called *leaves*, and the intermediate nodes are called *internal nodes*. The *depth d* of the tree is the length of the longest path from the root to a leaf. Moreover, the tree can be thought of as grouped into layers or *levels*, where the k-th level for $k \in \{0, 1, \ldots, d\}$ is the set of vertices which are connected to the root via a path consisting of precisely k edges.

Sanity check!

1. What is the depth of the tree above? (Answer: 3)
2. Which vertices are on level 0 of the tree above? How about on level 3?

Where do rooted trees come in handy? Consider, for example, the setting of bacterial cell division. In this case, the root might represent a single bacterium, and each subsequent layer corresponds to cell division in which the bacterium divides into two new bacteria. Rooted trees can also be used to allow fast searching of ordered sets of elements, such as in *binary search trees*, which you may have already encountered in your studies.

One of the nice things about trees is that induction works particularly well in proving properties of trees. Let us demonstrate with a case in point: We shall prove that the first two definitions of a tree given above are indeed equivalent.

Theorem 5.2. *A graph $G = (V, E)$ is a tree if and only if G is connected and has $|V| - 1$ edges.*

Before proving this theorem, we prove two quick lemmas which will aid us in our induction.

Definition 5.1 (Leaf)**.** *Let $T = (V, E)$ be a tree. $v \in V$ is a leaf if $deg(v) = 1$.*

Lemma 5.1. *Every tree (on at least 2 vertices) has at least one leaf.*

Proof. Consider the longest (simple) path in our tree, and let v be the starting point of that path. We know that v cannot be connected to any vertex on the path (other than the one immediately after it), as that would create a cycle. But we also know that v cannot be connected to any non-path vertex, as then we could add that vertex to the beginning of the path — which is impossible since we already are working with the longest path. Thus, v is connected only to the vertex right after it in the cycle, and so is a leaf. □

Lemma 5.2. *If we remove a leaf from a tree (on at least 2 vertices), the result is still a tree.*

Proof. We need to show that the result of removing the leaf is connected and acyclic. The acyclic part is straightforward: removing a leaf cannot create a cycle, as any cycle that exists in the new graph must also have existed in the original. To prove that the result is still connected, we note that no simple path can ever pass *through* a leaf, as it cannot enter and leave on the same edge. Thus, removing a leaf cannot break the path that already existed between any two other nodes, meaning that our new graph will still have all those paths, and hence will still be connected. □

Proof of Theorem 5.2. We proceed by showing the forward and converse directions.

Forward direction. We prove using induction on n, the number of vertices, that if G is a tree, then G is connected and has $n-1$ edges.

Base case ($n = 1$): In this case, G is a single vertex and has no edges. Thus, the claim holds.

Inductive step: Suppose that the claim holds for trees on k vertices. Now let $G = (V,E)$ be a tree on $k+1$ vertices. By lemma 5.1, we know that G has a leaf; by lemma 5.2, we know that removing that leaf will result in a tree G' on k vertices. By the inductive hypothesis, G' has $k-1$ edges. But then G has just one more edge than G', and so has $k = (k+1)-1$ edges as desired.

Converse direction. We prove using induction on n, the number of vertices, that if G is connected and has $n-1$ edges, G is a tree.

Base case ($n = 1$): In this case, G is a single isolated vertex, which is indeed a tree. Thus, the claim holds.

Inductive step: Suppose that the claim holds for graphs on k vertices. Now let $G = (V,E)$ be a graph on $k+1$ vertices with k edges. If we sum the degrees of all the vertices in G, we'll get $2k$ (can you prove this?), which is strictly less than $2|V|$. Thus, the average degree in G is less than 2 — meaning there must be some vertex v with degree less than 2, as not every vertex can be above average. Furthermore, since G is connected, every vertex (v included) has degree at least 1, so the degree of v must be exactly 1. Thus, if we remove v and its incident edge, we are left with a graph on k vertices with $k-1$ edges. By the inductive hypothesis, this graph is a tree. But now if we add v and its edge back in, v cannot be part of a cycle (as it has a degree of 1), meaning G must also be acyclic. We will also have that v is connected to the rest of the graph by its edge, while the rest of the graph was already connected by the inductive hypothesis, so G as a whole is connected. Thus, we have that G is indeed a tree. □

4 Induction on Graphs

As we saw in the previous theorem, it is possible for us to use induction to prove properties about graphs. This is a very powerful tool, but one must be careful when applying it, as the following "proof" demonstrates.

Claim 5.1. *Let $G = (V,E)$ be a connected graph with $|V| \geq 2$. If G has a vertex of degree 1, G is a tree.*

"Proof". We proceed by induction on $|V|$.

Base Case($|V| = 2$): In this case, G must be two vertices connected by an edge, which is a tree.

Inductive Step: Suppose that the statement holds for all graphs on k vertices. Take any connected graph on k vertices that has a degree 1 vertex (and hence is a tree by the inductive hypothesis) and create a graph on $k+1$ vertices by adding a new vertex v with one edge incident to it. Similar to the previous proof, we know that v cannot create a cycle or disconnect the graph, so the result will still be a tree.

By induction, all graphs are trees.

□

But this claim is definitely false, so where did our proof go wrong? The problem is that our inductive step only shows that the statement is true for graphs which can be created by adding a leaf to a k vertex graph that already had one — and this is not true for all graphs on $k+1$ vertices.

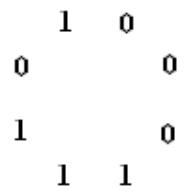
Sanity check! Give an example of a graph that has a degree 1 vertex which cannot be built in this way.

Indeed, this proof is an example of what is known as *build up error*. This type of error arises when we assume that a graph with some property can be "built up" from a smaller graph with that property, which may not always be possible, as in this case. In order to avoid this issue, proofs which induct on graphs use a *shrink down, grow back* process: we start with the larger graph, shrink it to a smaller graph by removing something, then add back what we removed after applying the inductive hypothesis. If you look back at the proof of theorem 5.2, you'll see that's exactly what we did!

Let's see what happens if we attempt to use this process to prove the inductive step of Claim 5.1. We start with a graph G on $k+1$ vertices that has a leaf. We can then shrink it to a graph G' on k vertices by removing a leaf. But now how do we know that G' has a leaf? We don't! Hence, we cannot apply the inductive hypothesis, and so we are stuck. Thus, the "shrink down, grow back" paradigm has prevented us from giving a false proof.

5 Practice Problems

1. A *de Bruijn sequence* is a 2^n-bit circular sequence such that every string of length n occurs as a contiguous substring of the sequence exactly once. For example, the following is a de Bruijn sequence for the case $n = 3$:



   Notice that there are eight substrings of length three, each of which corresponds to a binary number from 0 to 7 such as 000, 001, 010, etc. It turns out that such sequences can be generated from the *de Bruijn graph*, which is a directed graph $G = (V, E)$ on the vertex set $V = \{0,1\}^{n-1}$, i.e., the set of all $n-1$ bit strings. Each vertex $a_1a_2...a_{n-1} \in V$ has two outgoing edges:

$$(a_1a_2...a_{n-1},\ a_2a_3...a_{n-1}0) \in E \quad \text{and} \quad (a_1a_2...a_{n-1},\ a_2a_3...a_{n-1}1) \in E.$$

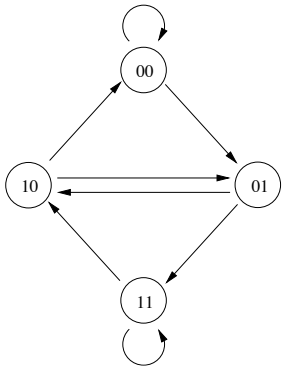
Therefore, each vertex also has two incoming edges:

$$(0a_1a_2...a_{n-2},\ a_1a_2...a_{n-1}) \in E \quad \text{and} \quad (1a_1a_2...a_{n-2},\ a_1a_2...a_{n-1}) \in E.$$

For example, for $n = 4$, the vertex 110 has two outgoing edges directed toward 100 and 101, and two incoming edges from 011 and 111. Note that these are directed edges, and self-loops are permitted.

The de Bruijn sequence is generated by an Eulerian tour in the de Bruijn graph. Euler's theorem (Theorem 5.1) can be modified to work for directed graphs — all we need to modify is the second condition, which should now say: "For every vertex v in V, the in-degree of v equals the out-degree of v." Clearly, the de Bruijn graph satisfies this condition, and therefore it has an Eulerian tour.

To actually generate the sequence, starting from any vertex, we walk along the tour and add the corresponding bit which was shifted in from the right as we traverse each edge. Here is the de Bruijn graph for $n = 3$.



Find the Eulerian tour of this graph that generates the de Bruijn sequence given above.

2. Prove using induction on the number of vertices n that any connected graph must have at least $n-1$ edges.