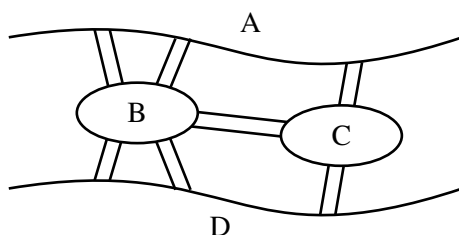


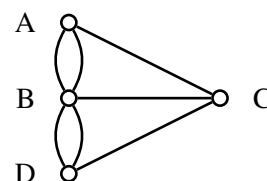
## 1 Graph Theory: An Introduction

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 note.

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.



(a) The city of Königsberg.



(b) The (multi-)graph modeling the bridge connections in Königsberg.

Figure 1: Two equivalent representations of the city of 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 point (shown as 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 point as many times as it exits it, and therefore the number of line segments incident to that point must be even. But in Figure 1b, each point 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, an (undirected) graph is defined by a set of vertices  $V$  and a set of edges  $E$ . The vertices correspond to the points 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. Unless otherwise stated, we will generally not allow multiple edges between a single pair of vertices, i.e., we will require  $E$  to be a set, not a multi-set. Thus between any pair of vertices there is either 0 or 1 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 the vertex set of a graph  $G$ . 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  $U \times V = \{(u, v) : u \in U \text{ and } v \in V\}$ .) For example, if  $V = \{1, 2, 3, 4\}$  and  $E = \{(1, 2), (1, 3), (1, 4)\}$ , then the corresponding graph is  $G_1$  as shown in Figure 2.

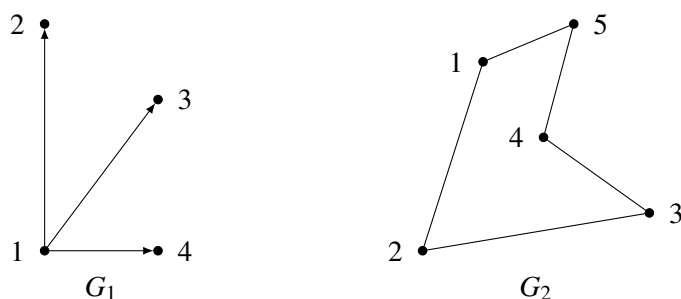


Figure 2: Examples of directed and undirected graphs, respectively.

Note that each edge in  $G_1$  has a *direction* specified by an arrow; thus, for example,  $(1, 2) \in E$  but  $(2, 1) \notin E$ . Directed graphs are useful in modeling one-way relationships, such as one-way streets between two locations. 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 naturally model relationships such as two-way streets between locations; an example is given by  $G_2$  in Figure 2. For simplicity, we omit the arrowheads when drawing edges in undirected graphs. In either case, a graph is formally specified as an ordered pair  $G = (V, E)$ , where  $V$  is the vertex set and  $E$  is the edge set (directed or undirected).

---

*Concept check!* What are the vertex and edge sets  $V$  and  $E$  for graph  $G_2$ ?

---

Let us continue our discussion with an 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 necessarily know who Alex is). We would use a directed graph to model this situation, since the relationship is not symmetrical. On the other hand, if we instead wanted to model the relationship that Alex and Bridget *know each other*, then we would use an undirected graph.

Introducing some further useful terminology, 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 on  $u$ , i.e.,  $\text{degree}(u) = |\{v \in V : \{u, v\} \in E\}|$ . 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.

---

*Concept 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.

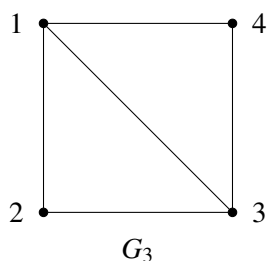
---

*Concept 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.

**Paths, walks, and cycles.** Let  $G = (V, E)$  be an undirected graph. A *path* in  $G$  is a sequence of edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-2}, v_{n-1}\}, \{v_{n-1}, v_n\}$ . 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  $v$ .



---

*Concept check!* What is a shortest path from house 2 to house 4 in  $G_3$ ? How about a longest path, assuming no house is visited twice?

---

In this class, we assume a path is *simple*, meaning  $v_1, \dots, v_n$  are distinct. This makes complete sense in our housing example  $G_3$ ; if you wanted drive from house 2 to 4 via house 1, why would you visit house 1 more than once? A *cycle* (or *circuit*) is a sequence of edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-2}, v_{n-1}\}, \{v_{n-1}, v_n\}, \{v_n, v_1\}$ , where  $v_1, \dots, v_n$  are distinct. Thus a cycle is a path from  $v_1$  to  $v_n$ , plus the edge  $\{v_n, v_1\}$  taking us back to  $v_1$  again.

---

*Concept check!* Give a cycle starting at house 1 in  $G_3$ .

---

Suppose now that your aim is not to go from 2 to 4 as quickly as possible, but to take a leisurely stroll from 2 to 4 via the sequence  $\{2, 1\}, \{1, 2\}, \{2, 3\}, \{3, 4\}$ . A sequence of edges with possibly repeated vertices, such as this one, is called a *walk* from 2 to 4. Analogous to the relationship between paths and cycles, a *tour* is a walk which starts and ends at the same vertex. For example,  $\{2, 1\}, \{1, 3\}, \{3, 4\}, \{4, 1\}, \{1, 2\}$  is a tour.

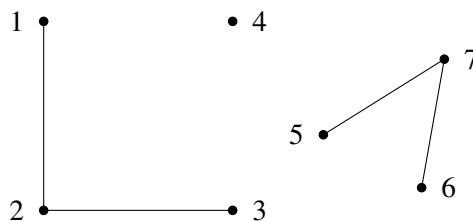
Here's a quick summary of all of these terms:

|       | no repeated vertices | no repeated edges | start = end |
|-------|----------------------|-------------------|-------------|
| Walk  |                      |                   |             |
| Path  | ✓                    | ✓                 |             |
| Tour  |                      |                   | ✓           |
| Cycle | ✓*                   | ✓                 | ✓           |

(\*except for start and end vertices)

*Remark:* In this class, we will do our best to remind you of the subtleties in the definitions of the terms; don't worry too much about the formal definitions, as long as you understand the ideas.

**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*.



*Concept 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  $V_1, \dots, V_k$  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 Königsberg: 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,  $\text{FINDTOUR}(G, s)$ , which finds a tour (not necessarily Eulerian) in  $G$ .  $\text{FINDTOUR}$  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  $\text{FINDTOUR}$  must in fact get stuck at the original vertex  $s$ .

**Claim 5.1.**  $\text{FINDTOUR}(G, s)$  must get stuck at  $s$ .

*Proof of claim.* An easy proof by induction on the length of the walk shows that when  $\text{FINDTOUR}$  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.  $\square$

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

We now give a recursive algorithm  $\text{EULER}(G, s)$  that outputs an Eulerian tour starting and ending at  $s$ .  $\text{EULER}(G, s)$  invokes another subroutine  $\text{SPLICE}(T, T_1, \dots, T_k)$  which takes as input a number of edge disjoint tours  $T, T_1, \dots, T_k$  ( $k \geq 1$ ), with the condition that the tour  $T$  intersects each of the tours  $T_1, \dots, T_k$  (i.e.,  $T$  shares a vertex with each of the  $T_i$ 's). The procedure  $\text{SPLICE}(T, T_1, \dots, T_k)$  outputs a single tour  $T'$  that traverses all the edges in  $T, T_1, \dots, T_k$ , 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 does it continue traversing  $T$ .

The algorithm  $\text{EULER}(G, s)$  is given as follows:

```
Function Euler(G, s)
  T = FindTour(G, s)
  Let  $G_1, \dots, G_k$  be the connected components
    when the edges in T are removed from G
  Let  $s_i$  be the first vertex in T that intersects  $G_i$ 
  Output Splice(T, Euler( $G_1, s_1$ ), ..., Euler( $G_k, s_k$ ))
end Euler
```

We prove by induction on the size of  $G$  that  $\text{EULER}(G, s)$  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:  $\text{EULER}(G, s)$  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  $T = \text{FINDTOUR}(G, s)$  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  $G_1, \dots, G_k$  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  $\text{EULER}(G_i, s_i)$  outputs an Eulerian tour of  $G_i$ . Now by the definition of  $\text{SPLICE}$ , it splices the individual tours together into one large tour whose union is all the edges of  $G$ , hence an Eulerian tour.  $\square$

---

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

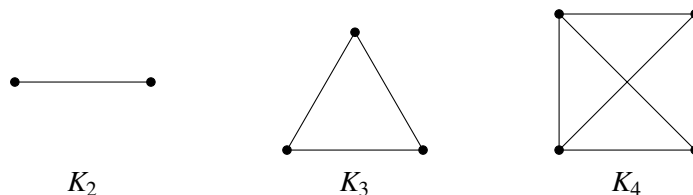
---

## 3 Trees and Complete Graphs

In this section we introduce two very important special classes of graphs, namely complete graphs and trees. These graphs are maximal and minimal, respectively, among all connected graphs, in a sense that we will discuss in more detail in Section 5.

### 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  $\{u, v\}$ . The following diagrams show the complete graphs on  $n = 2, 3, 4$  vertices, respectively.



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  $E = \{\{v_i, v_j\} \mid v_i \neq v_j \text{ and } v_i, v_j \in V\}$ .

---

*Concept 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.

---

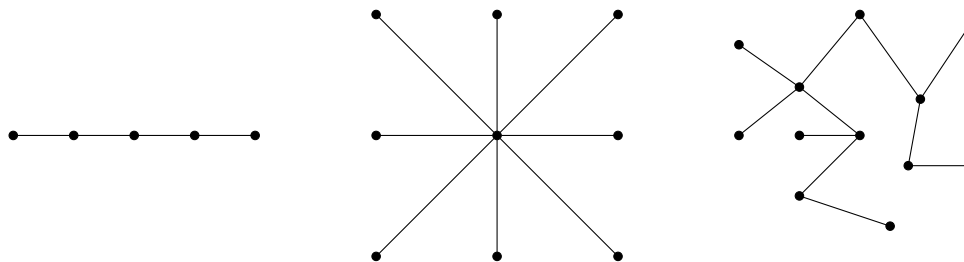
Finally, we can also introduce complete *directed* graphs, which are defined as you might expect: For any pair of vertices  $u$  and  $v$ , both  $(u, v)$  and  $(v, u)$  belong to  $E$ .

## 3.2 Trees

A *tree* is a connected graph that contains no cycles. In fact, there are several equivalent definitions for when a graph  $G = (V, E)$  is a tree, including the following:

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:



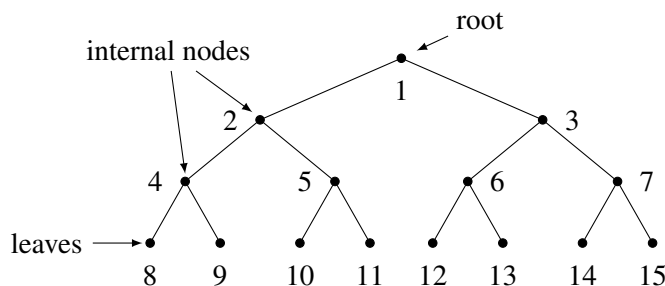
---

*Concept 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.
- 

Note that, for any given number of vertices  $n$ , there are a very large number of possible trees ( $n^{n-2}$  to be precise). This is in sharp contrast to complete graphs, which are unique for any given  $n$ .

Why are trees important? 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 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*. Note that in a rooted tree, a root is never a leaf. In particular, this differs from an unrooted tree; in an unrooted tree, *leaves* are any vertex of degree 1. 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  $k \in \{0, 1, \dots, d\}$  is the set of vertices which are connected to the root via a path consisting of precisely  $k$  edges.

---

*Concept 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 them. 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.** *The statements “ $G$  is connected and contains no cycles” and “ $G$  is connected and has  $n - 1$  edges” are equivalent.*

*Proof.* We proceed by showing the forward and converse directions.

*Forward direction.* We prove using strong induction on  $n$  that if  $G$  is connected and contains no cycles, then  $G$  is connected and has  $n - 1$  edges. Assume  $G = (V, E)$  is connected and contains no cycles.

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

*Inductive hypothesis:* Assume the claim is true for  $1 \leq n \leq k$ .

*Inductive proof:* We show the claim for  $n = k + 1$ . Remove an arbitrary vertex  $v \in V$  from  $G$  along with its incident edges, and call the resulting graph  $G'$ . Clearly, *removing* a vertex cannot create a cycle; thus,  $G'$  contains no cycles. However, removing  $v$  may result in a *disconnected* graph  $G'$ , in which case the induction hypothesis cannot be applied to  $G'$  as a whole. Thus, we have two cases to examine — either  $G'$  is connected, or  $G'$  is disconnected. Here, we show the former case, as it is simpler and captures the essential proof ideas. The latter case is left as an exercise below.

So, assume  $G'$  is connected. But now  $G'$  is a connected graph with no cycles on  $k$  vertices, so we can apply the induction hypothesis to  $G'$  to conclude that  $G'$  is connected and has  $k - 1$  edges. Let us now add  $v$  back to  $G'$  to obtain  $G$ . How many edges can be incident on  $v$ ? Well, since  $G'$  is connected, then if  $v$  is incident on more than one edge,  $G$  will contain a cycle. But by assumption  $G$  has no cycles! Thus,  $v$  must be incident on one edge, implying  $G$  has  $(k - 1) + 1 = k$  edges, as desired.

*Converse direction.* We prove using contradiction that if  $G$  is connected and has  $n - 1$  edges, then  $G$  is connected and contains no cycles. Assume  $G$  is connected, has  $n - 1$  edges, and contains a cycle. Then, by definition of a cycle, removing any edge in the cycle does not disconnect the graph  $G$ . In other words, we can remove an edge in the cycle to obtain a new connected graph  $G'$  consisting of  $n - 2$  edges. However, we claim that  $G'$  must be disconnected, which will yield our desired contradiction. This is because in order for a graph to be connected, it must have at least  $n - 1$  edges. This is a fact that you have to prove in the exercise below. This completes the proof of the converse direction.  $\square$

## 4 Planar Graphs and Euler's Formula

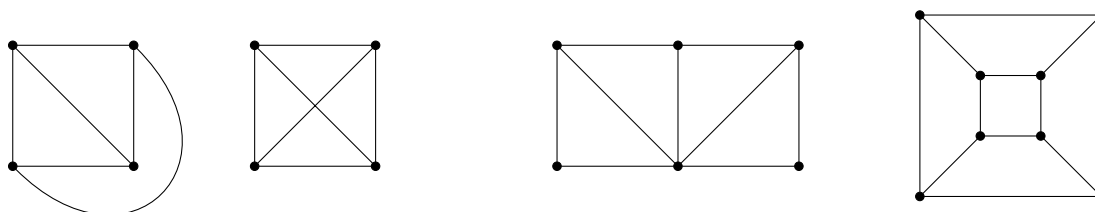
A graph is *planar* if it can be drawn on the plane without crossings. For example, the first four graphs shown below are planar. Notice that the first and second graphs are the same, but drawn differently. Even though



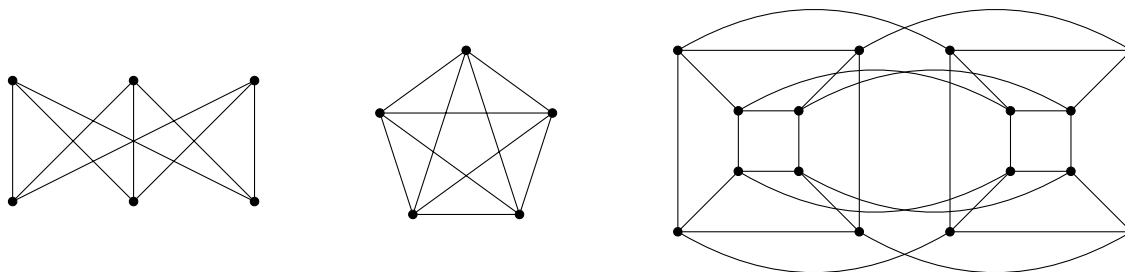
the second drawing has crossings, the graph is still considered planar since it is possible to draw it without crossings.

The other three graphs are not planar. The first one of them is the infamous “three houses-three wells graph,” also called  $K_{3,3}$ , (This notation says there are two sets of vertices, each of size three and all edges between the two sets of vertices are present.) The second is the “complete” graph (every edge is present) with five nodes, or  $K_5$ . The third is the four-dimensional cube. We shall soon see how to prove that all three graphs are non-planar.

A useful concept is the notion of a *bipartite* graph,  $G = (V, E)$  which is a graph where the vertices are split into two groups and edges only go between groups. More formally, we have  $V = L \cup R$  and  $E \subseteq L \times R$ . Clearly,  $K_{3,3}$  is bipartite. Moreover, the four dimensional cube is bipartite: this follows from the fact that every cycle in the cube has even length. We will show the equivalence a bit later.



Planar graphs



Non-planar graphs

When a planar graph is drawn on the plane, one can distinguish, besides its vertices (their number will be denoted  $v$  here) and edges (their number is  $e$ ), the *faces* of the graph (more precisely, of the drawing). The faces are the regions into which the graph subdivides the plane. One of them is infinite (the “outer” face), and the others are finite. The number of faces is denoted  $f$ . For example, for the first graph shown  $f = 4$ , and for the fourth (the cube)  $f = 6$ .

One basic and important fact about planar graphs is *Euler’s formula*,  $v + f = e + 2$  (check it for the graphs above). It has an interesting story. The ancient Greeks knew that this formula held for all polyhedra (check it for the cube, the tetrahedron, and the octahedron, for example), but could not prove it. How do you do induction on polyhedra? How do you apply the induction hypothesis? What is a polyhedron minus a vertex, or an edge? In the 18th century Euler realized that this is an instance of the inability to prove a theorem by induction *because it is too weak*, something that we saw time and again when we were studying induction. To prove the theorem, one has to generalize polyhedra. And the right generalization is *planar graphs*.

---

*Exercise.* Can you see why planar graphs generalize polyhedra? Why are all (convex) polyhedra planar graphs?

---

**Theorem 5.3.** (*Euler’s formula*) For every connected planar graph,  $v + f = e + 2$

*Proof.* By induction on  $e$ . It certainly holds when  $e = 0$ , and  $v = f = 1$ . Now take any connected planar graph  $G$ . There are two cases to consider:

- If  $G$  is a tree, then  $f = 1$  (drawing a tree on the plane does not subdivide the plane), and  $e = v - 1$ . So the formula holds.
- If  $G$  is not a tree, find a cycle and delete any edge of the cycle. This amounts to reducing both  $e$  and  $f$  by one. By induction the formula is true in the smaller graph, and so it must be true in the original one.

□

---

*Exercise.* What happens when the graph is not connected? How does the number of connected components enter the formula?

---

Take a planar graph with  $f$  faces, and consider one face. It has a number of *sides*, that is, edges that bound it clockwise. Note that an edge may be counted twice, if it has the same face on both sides, as happens for example in a tree (such edges are called “bridges”). Denote by  $s_i$  the number of sides of face  $i$ . Now, if we add up all the  $s_i$ ’s we must get  $2e$ , because each edge is counted twice, once for the face on its right and once for the face on its left (these may coincide if the edge is a bridge). We conclude that, in any planar graph,

$$\sum_{i=1}^f s_i = 2e \quad (1)$$

Now notice that, since we don’t allow parallel edges between the same two nodes, and if we assume that there are at least two edges (so there are at least three vertices), every face has at least three sides, or  $s_i \geq 3$  for all  $i$ . It follows that  $3f \leq 2e$ . Solving for  $f$  and plugging into Euler’s formula we get

$$e \leq 3v - 6.$$

This is an important fact. First it tells us that planar graphs are *sparse*: i.e., they cannot have too many edges. A 1,000-vertex connected graph can have anywhere between a thousand and half a million edges. This inequality tells us that for planar graphs the range is much smaller, between 999 and 2,994.

It also tells us that  $K_5$  is not planar: Just notice that it has five vertices and ten edges.

$K_{3,3}$  has  $v = 6, e = 9$  so it passes the planarity test with flying colors. We must think a little harder to show that  $K_{3,3}$  is non-planar. Notice that, if we were able to draw it on the plane, there would be no triangles: this is because a triangle would mean that either two wells or two houses are connected to each other, which is not the case. So, we can now strengthen what we get from (1) from  $3f \leq 2e$  to  $4f \leq 2e$  Solving for  $f$  and plugging into Euler’s formula as before now yields the stronger conclusion  $e \leq 2v - 4$ , which shows that  $K_{3,3}$  is indeed non-planar!

So, we have established that  $K_5$  and  $K_{3,3}$  are both non-planar. There is something deeper going on: In some sense, these are *the only non-planar graphs*. This is made precise in the following famous result, due to the Polish mathematician Kuratowski (this is what “K” stands for).

**Theorem 5.4.** *A graph is non-planar if and only if it contains  $K_5$  or  $K_{3,3}$ .*

“Contains” here means that one can identify nodes in the graph (five in the case of  $K_5$ , six in the case of  $K_{3,3}$ ) which are connected as the corresponding graph through paths (possibly single edges), and such that no two of these paths share no vertex. For example, the 4-cube shown in Figure 3 is non-planar, because it contains  $K_{3,3}$ , as shown.

Figure 3:  $K_{3,3}$  in a 4-cube (to view the animation, use Adobe Acrobat or another PDF reader supporting PDF animations)

---

*Exercise.* Can you find  $K_5$  in the same graph?

---

One direction of Kuratowski's theorem is obvious: If a graph contains one of these two non-planar graphs, then of course it is itself non-planar. The other direction, namely that in the absence of these graphs we can draw any graph on the plane, is difficult. If you are interested in a proof, you may want to Google "proof of Kuratowski's theorem."

## Duality and Coloring

There is an interesting *duality* between planar graphs. For example, the Greeks knew that the octahedron and the cube are "dual" to each other, in the sense that the faces of one can be put in correspondence with the vertices of the other (think about it!). The tetrahedron is self-dual, i.e., it is its own dual. And the dodecahedron and the icosahedron (look for images on the web if you don't know these polyhedra) are also dual to one another.

What does this mean? Take a planar graph  $G$ , and assume it has no bridges and no degree-2 vertices. Draw a new graph  $G^*$  as follows: Start by placing a vertex on each face of  $G$ . Then draw an edge between two faces if they touch at an edge — draw the new edge so that it crosses that shared edge. The result is  $G^*$ , also a planar graph. Notice now that, if you construct the dual of  $G^*$ , it is the original graph: i.e.,  $(G^*)^* = G$ .

Duality is a convenient consideration when thinking about planar graphs. Also, it tells us that "coloring a political map so that no two countries who share a border have the same color" is the same problem as "coloring the vertices of a planar graph (the dual of the political map) so that no two adjacent vertices have the same color." We call such a coloring "legal" or "proper."

Returning for a moment to the concept of a bipartite graph in the context of coloring, one sees that a two colorable graph is equivalent to a bipartite graph as the coloring splits the vertices into two sets where edges only connect vertices in the two sets(colors). Here, with this observation, we can see that a graph that does not contain any odd length cycles is bipartite as we can greedily color any such graph with two colors as follows for a connected graph. Color one vertex, say  $x$  red, color its neighbors blue, and then continue by coloring their uncolored neighbors red, and continue alternatively assigning colors to uncolored neighbors.

If this does not work, then some edge  $(u, v)$  must connect two vertices of the same color, in which case we can identify an odd cycle by using the path  $p_1$  from  $x$  to  $u$ , the edge  $(u, v)$  and the path  $p_2$  from  $v$  to  $x$  and removing common portions of  $p_1$  and  $p_2$ . The lengths of  $p_1$  and  $p_2$  have the same parity (even or odd) as  $u$  and  $v$  have the same color and the total length of the cycle must therefore be odd as it includes the additional edge  $(u, v)$ .

A famous theorem in graph theory states that four colors are always enough to legally color any planar graph! (Search for “The Four Color Theorem” to find out more about this result, which was proved in 1976 by Appel and Haken in one of the earliest examples of a computer-assisted proof.)

We shall prove something slightly weaker (which, however, is *much* easier to prove!).

**Theorem 5.5.** *Every planar graph can be colored with five colors.*

Before proceeding with the proof, we consider a “switch” operation on a legal coloring of a graph. We first take the subset of vertices of two colors, say 1 and 2, and compute its connected components. Then if we *switch* these two colors throughout a single connected component, we get another legal coloring: this is because any edge in the connected component has the colors at both its endpoints switched, while edges in other connected components are unaffected by these switches.

*Proof.* Induction on the number of vertices  $v$ . The base case  $v = 1$  is trivial, so we go directly to the inductive step. Let  $G$  be a planar graph. Note first that  $G$  must have a vertex of degree five or less. To prove this, consider the inequality  $e \leq 3v - 6$ , which is a corollary of Euler’s Theorem. If all vertices had degree six or more, then  $e$  would be at least  $3v$ , a contradiction.

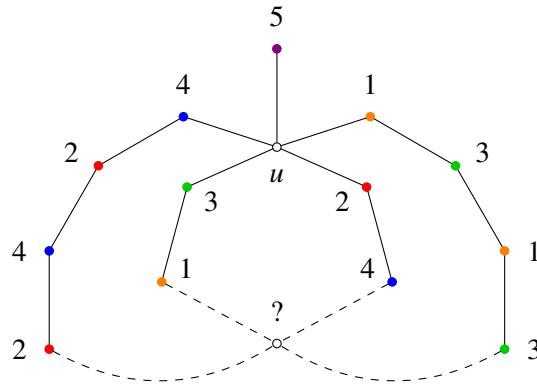
So, consider a vertex  $u$  of degree five or less. If it has degree four or less, we are done as follows: Remove  $u$ , color the remaining graph with 5 colors (by the induction hypothesis), and then put  $u$  back in and color it with a color that is missing from its neighbors. The same argument holds if  $u$  has five neighbors, and in the inductive coloring of  $G - u$  fewer than five colors are used for those neighbors.

Now consider the only remaining case where  $u$  has exactly five neighbors, and in the inductive coloring of  $G - u$  all of these neighbors receive different colors. Proceeding clockwise around  $u$ , label them  $u_1, u_2, u_3, u_4, u_5$ , and their respective colors 1, 2, 3, 4, 5.

Now try to change the color of  $u_2$  to color 4 by determining the connected component containing  $u_2$  and consisting of vertices that are colored 2 or 4. If  $u_4$  is not in this connected component, then we apply the switch operation to it and obtain a coloring in which only four colors are present in the neighbors of  $u$ —so we can color  $u$  with the remaining color and we are done. Conversely, if  $u_4$  is in this connected component, then switching colors 2 and 4 is not helpful; however, in that case we do know that there is a path between  $u_2$  and  $u_4$  colored with 2 and 4.

We can now similarly attempt a switch operation on colors 1 and 3 in the connected component with these colors containing  $u_1$ . As above, this will be successful and finish the argument unless there is a path between  $u_1$  and  $u_3$  colored with 1 and 3.

If both of these attempts fail, then we get two paths: one from  $u_1$  to  $u_3$  colored 1 and 3, and the other from  $u_2$  to  $u_4$  colored 2 and 4. But since  $G$  is planar, these two paths must intersect at some vertex, which is a contradiction because the colors in the two paths are disjoint! Hence we conclude that in fact at least one of the color switch operations must succeed, allowing us to legally color  $u$  and complete the inductive step.  $\square$



## 5 Connectivity and Hypercubes

As we have seen, graphs are an abstract and general construct allowing us to represent relationships between objects, such as houses and roads. One particularly important application of graphs is as a representation of the interconnections between routers on the internet (or some other network). To send a packet from one node to another, we need to find a path in this graph from our source to our destination. Therefore, to be able to send a packet from any node to any other node, we need the graph to be connected. A minimally connected graph is called a *tree*, and it is the most efficient graph (i.e., with the minimum number of edges) we can use to connect any set of vertices. More specifically, if any one edge (connection) in a tree network fails, then the network become disconnected and there are some nodes we cannot reach. To avoid this situation, we want some kind of redundancy, or robustness, in the connectivity of the network.

At the opposite extreme is the complete graph, which contains a connection between every pair of nodes. This network is extremely well connected, as the following exercise shows.

---

*Exercise.* What is the minimum number of edges which must be removed from  $K_n$  to make it disconnected? [Hint: how many edges have to be removed to contain an isolated vertex?]

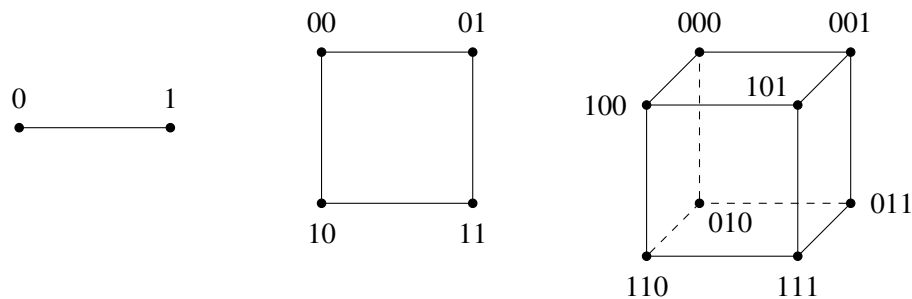
---

However, the complete graph is completely unrealistic as a network architecture because it contains far too many edges:  $\frac{n(n-1)}{2} \sim \frac{n^2}{2}$ , to be precise. And in practice there is no need for a direct connection between every pair of nodes, as long as the maximum node-to-node distance (also called the “diameter” of the network) is not too large.

In this section, we examine a nice family of graphs called the *hypercube* graphs, which combines the best of both worlds: they are robustly connected and have small diameter, but do not use too many edges.

We have discussed how complete graphs are a class of graphs whose vertices are particularly “well-connected.” However, to achieve this strong connectivity, a large number of edges is required, which in many applications of graph theory is infeasible. Consider the example of the *Connection Machine*, which was a massively parallel computer designed by the company Thinking Machines in the 1980s. The idea of the Connection Machine was to have a million processors working in parallel, all connected via a communications network. If you were to connect each pair of such processors with a direct wire to allow them to communicate (i.e., if you used a complete graph to model your communications network), this would require  $10^{12}$  wires! What the builders of the Connection Machine thus decided was to instead use a 20-dimensional *hypercube* to model their network, which still allowed a strong level of connectivity, while limiting the number of neighbors of each processor in the network to 20. This section is devoted to studying this particularly useful class of graphs, known as hypercubes.

The vertex set of the  $n$ -dimensional hypercube  $G = (V, E)$  is given by  $V = \{0, 1\}^n$ , where recall  $\{0, 1\}^n$  denotes the set of all  $n$ -bit strings. In other words, each vertex is labeled by a unique  $n$ -bit string, such as 00110...0100. The edge set  $E$  is defined as follows: Two vertices  $x$  and  $y$  are connected by edge  $\{x, y\}$  if and only if  $x$  and  $y$  differ in exactly one bit position. For example,  $x = 0000$  and  $y = 1000$  are neighbors, but  $x = 0000$  and  $y = 0011$  are not. More formally,  $x = x_1x_2 \dots x_n$  and  $y = y_1y_2 \dots y_n$  are neighbors if and only if there is an  $i \in \{1, \dots, n\}$  such that  $x_j = y_j$  for all  $j \neq i$ , and  $x_i \neq y_i$ . To help you visualize the hypercube, we depict the 1-, 2-, and 3-dimensional hypercubes below.



There is an alternative and useful way to define the  $n$ -dimensional hypercube via recursion, which we now discuss. Define the 0-subcube (respectively, 1-subcube) as the  $(n-1)$ -dimensional hypercube with vertices labeled by  $0x$  for  $x \in \{0, 1\}^{n-1}$  (respectively,  $1x$  for  $x \in \{0, 1\}^{n-1}$ ). Then, the  $n$ -dimensional hypercube is obtained by placing an edge between each pair of vertices  $0x$  in the 0-subcube and  $1x$  in the 1-subcube.

---

*Concept check!* Where are the 0- and 1-subcubes in the 3-dimensional hypercube depicted above? Can you use these along with the recursive definition above to draw the 4-dimensional hypercube?

---

*Exercise.* Prove that the  $n$ -dimensional hypercube has  $2^n$  vertices, and that its diameter (maximum length of a path between any two vertices) is  $n$ . [Hint: Use the fact that each bit has two possible settings, 0 or 1.] Note that the diameter is exponentially smaller than the number of vertices!

---

We began this section by singing the praises of the hypercube in terms of its connectivity properties; we now investigate these claims formally. Let us warm up by giving two proofs of a simple property of the hypercube. Each proof relies on one of our two equivalent (namely, direct and recursive) definitions of the hypercube.

**Lemma 5.1.** *The total number of edges in an  $n$ -dimensional hypercube is  $n2^{n-1}$ .*

*Proof 1.* The degree of each vertex is  $n$ , since  $n$  bit positions can be flipped in any  $x \in \{0, 1\}^n$ . Since each edge is counted twice, once from each endpoint, this yields a total of  $n2^n/2 = n2^{n-1}$  edges.  $\square$

*Proof 2.* By the second definition of the hypercube, it follows that  $E(n) = 2E(n-1) + 2^{n-1}$ , and  $E(1) = 1$ , where  $E(n)$  denotes the number of edges in the  $n$ -dimensional hypercube. A straightforward induction shows that  $E(n) = n2^{n-1}$ .  $\square$

---

*Exercise.* Use induction to show that in Proof 2 above,  $E(n) = n2^{n-1}$ .

---

Let us now focus on the question of connectivity, and prove that the  $n$ -dimensional hypercube is well-connected in the following sense: To disconnect any subset  $S \subseteq V$  of vertices from the rest of the graph, a large number of edges must be removed. In particular, we shall see that the number of removed edges must

scale with  $|S|$ . In the theorem below, recall that  $V - S = \{v \in V \mid v \notin S\}$  is the set of vertices that are not in  $S$ .

**Theorem 5.6.** *Let  $S \subseteq V$  be such that  $|S| \leq |V - S|$  (i.e., that  $|S| \leq 2^{n-1}$ ), and let  $E_S$  denote the set of edges connecting  $S$  to  $V - S$ , i.e.,*

$$E_S := \{\{u, v\} \in E \mid u \in S \text{ and } v \in V - S\}.$$

*Then, it holds that  $|E_S| \geq |S|$ .*

*Proof.* We proceed by induction on  $n$ .

*Base case ( $n = 1$ ):* The 1-dimensional hypercube graph has two vertices 0 and 1, and one edge  $\{0, 1\}$ . We also have the assumption  $|S| \leq 2^{1-1} = 1$ , so there are two possibilities. First, if  $|S| = 0$ , then the claim trivially holds. Otherwise, if  $|S| = 1$ , then  $S = \{0\}$  and  $V - S = \{1\}$ , or vice versa. In either case we have  $E_S = \{0, 1\}$ , so  $|E_S| = 1 = |S|$ .

*Inductive hypothesis:* Assume the claim holds for  $1 \leq n \leq k$ .

*Inductive step:* We prove the claim for  $n = k + 1$ . Recall that we have the assumption  $|S| \leq 2^k$ . Let  $S_0$  (respectively,  $S_1$ ) be the vertices from the 0-subcube (respectively, 1-subcube) in  $S$ . We may assume without loss of generality that  $|S_0| \geq |S_1|$ : if not, then we can just interchange the roles of  $S_0$  and  $S_1$  in the argument below. We have two cases to examine: Either  $S$  has a fairly equal intersection size with the 0- and 1-subcubes, or it does not.

1. **Case 1:**  $|S_0| \leq 2^{k-1}$  and  $|S_1| \leq 2^{k-1}$

In this case, we can apply the induction hypothesis separately to the 0- and 1-subcubes. This says that, restricted to just the 0-subcube, there are at least  $|S_0|$  edges between  $|S_0|$  and its complement (in the 0-subcube), and similarly there are at least  $|S_1|$  edges between  $|S_1|$  and its complement (in the 1-subcube). Thus, the total number of edges between  $S$  and  $V - S$  is at least  $|S_0| + |S_1| = |S|$ , as desired.

2. **Case 2:**  $|S_0| > 2^{k-1}$

In this case,  $S_0$  is unfortunately too large for the induction hypothesis to apply. However, note that since  $|S| \leq 2^k$ , we have  $|S_1| = |S| - |S_0| \leq 2^{k-1}$ , so we *can* apply the hypothesis to  $S_1$ . As in Case 1, this allows us to conclude that there are at least  $|S_1|$  edges in the 1-subcube crossing between  $S$  and  $V - S$ .

What about the 0-subcube? Here, we cannot apply the induction hypothesis directly, but there is a way to apply it after a little massaging. Consider the set  $V_0 - S_0$ , where  $V_0$  is the set of vertices in the 0-subcube. Note that  $|V_0| = 2^k$  and  $|V_0 - S_0| = |V_0| - |S_0| = 2^k - |S_0| < 2^k - 2^{k-1} = 2^{k-1}$ . Thus, we *can* apply the inductive hypothesis to the set  $V_0 - S_0$ . This yields that the number of edges between  $S_0$  and  $V_0 - S_0$  is at least  $2^k - |S_0|$ . Adding our totals for the 0-subcube and the 1-subcube so far, we conclude there are at least  $2^k - |S_0| + |S_1|$  crossing edges between  $S$  and  $V - S$ . However, recall our goal was to show that the number of crossing edges is at least  $|S|$ ; thus, we are still short of where we wish to be.

But there are still some edges we have not accounted for — namely, those in  $E_S$  which cross between the 0- and 1-subcubes. Since there is an edge between every vertex of the form  $0x$  and the corresponding vertex  $1x$ , we conclude there are at least  $|S_0| - |S_1|$  edges in  $E_S$  that cross between the two subcubes. Thus, the total number of edges crossing is at least  $2^k - |S_0| + |S_1| + |S_0| - |S_1| = 2^k \geq |S|$ , as desired.

## 6 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  $n = 3$ :

1    0  
 0        0  
 1        0  
 1    1

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  $a_1a_2\dots a_{n-1} \in V$  has two outgoing edges:

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

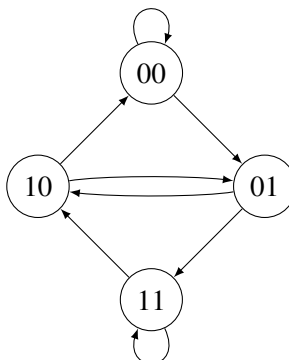
Therefore, each vertex also has two incoming edges:

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

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. In this question, we complete the induction component of the proof of Theorem 5.2.



- (a) Suppose in the proof that  $G'$  has two distinct connected components  $G'_1$  and  $G'_2$ . Complete the inductive step to show that  $G$  is connected and has  $n - 1$  edges. (Hint: Argue that you can apply the induction hypothesis to  $G'_1$  and  $G'_2$  separately. Note that this requires strong induction!)
- (b) More generally,  $G'$  may have  $t \geq 2$  distinct connected components  $G'_1$  through  $G'_t$  — generalize your argument above to this setting in order to complete the proof of Theorem 5.2.

---

*Life lesson.* This question teaches you a general paradigm for solving problems, be it in computer science research or everyday life. Specifically, when faced with a difficult problem (such as the proof of Theorem 5.2), first try to solve it in the simplest case possible (such as when  $G'$  is connected). Then, gradually extend your solution to handle more difficult cases until you establish the general claim (i.e.,  $G'$  has  $t$  connected components).

---

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