# CSC 202 Mathematics for Computer Science Lecture Notes

Marcus Schaefer
DePaul University[1]

# Part II

# Games and Puzzles

Wage du, zu irren und zu träumen! Hoher Sinn liegt oft in kind´schem Spiel.

Friedrich von Schiller

# Chapter 7

# Abstraction and Structure

We saw in the first part of this course how closely logic and relational databases interleave. At the same time we also saw that logic can be used for other purposes as well: that's the reason we teach logic, rather than just teaching relational databases. Logic as an abstraction can be applied in many domains. Logic, however, is only one of the many structures that mathematics has to offer. In this section we will encounter two other mathematical abstractions that are central to computer science: graphs and numbers. We will study these tools in the context of puzzles and games, where they occur naturally.

Here is a simple puzzle: suppose you placed your finger at the $X$ in Figure 7.1. Can you draw the string away, or will it get caught around your finger?

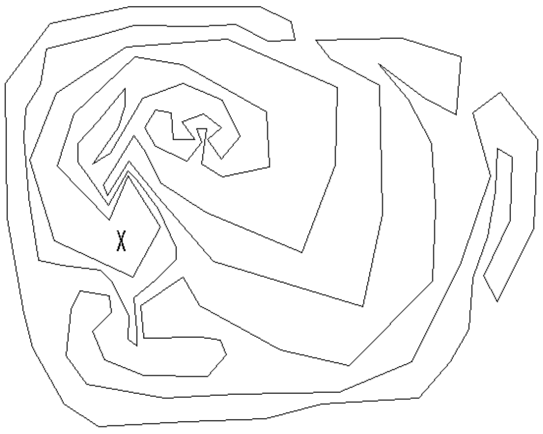


Figure 7.1: Pull or no pull?

The answer is quite intuitive: draw a line from the $X$ to the region outside of the curve. If that line intersects the curve an even number of times, the

$X$ is outside. Otherwise $X$ is inside. (We assume that the line never *touches* the curve, but always crosses it.) This result is known as the Jordan Curve theorem and is quite difficult to prove, although it really states a very intuitive fact about curves: they separate the plane into an *inside* and an *outside region.* Moreover, it gives us a very easy criterion for checking whether a point is inside or outside the curve (think computational geometry): it just depends on the *parity*, that is, the even- or oddness, of the number of intersections. Parity is one of the simplest, but at the same time one of the most powerful abstractions: everything falls into two categories: even or odd.

Let us start with a well-worn puzzle, which, nevertheless beautifully illustrates the power of a simple idea. Consider the chess board shown in Figure 7.2 with two opposite corner squares removed.

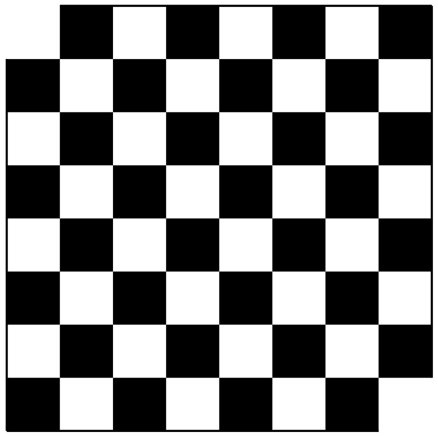


Figure 7.2: Tiling a chessboard

Can one tile this chessboard with dominoes? A *domino* tile consists of two adjacent squares; tiling the board means every square of the board has to be covered, and the tiles may not stick out or overlap. Is this possible? Following Maass' 4th bit of advice, you might want to consider simpler versions of the problem first: how about a modified $n \times n$ board where $n = 2, 3, 4, 5$. What do you learn from that?

Do this before you go on reading.

If you tried a couple of small board, you probably realized that none of them can be tiled with dominoes. Looking more closely at a particular case, say $n = 4$, you might have discovered the reason for this: for even $n$ the modification we make to the chessboard removes two squares of the same color (white squares in the illustration above). However, a domino tile covers two squares of opposite color. So if some board has a tiling by dominos it must have the same number of black and white squares.

We can express this by saying that having the same number of black and white squares is a *necessary* condition for a board being tileable by dominoes. It is not enough, however.

**Exercise 7.0.1.** Construct a board (part of a chessboard) that has the same number of black and white squares, but cannot be tiled by dominoes.

The necessary condition for tileability by dominoes is not met by any of the modified chessboard for $n$ even, since they have $n^2/2 - 1$ white and $n^2/2 + 1$ black squares (if the white corner squares were removed).

**Exercise 7.0.2.** As it turned out, the small examples for $n = 3$ and $n = 5$ were not relevant for settling the original question on the $8 \times 8$ board. But they should come in handy now: show that the modified $n \times n$ board cannot be tiled by dominoes for any odd $n$.

What settled the chessboard problem was thinking in terms of two: two colors, black and white, structuring the chessboard. Often the structure is not quite as obvious.

The next puzzle is a famous puzzle which created a furor in the 1880s comparable maybe to the Rubik's cube in the 1980s: The 14-15 puzzle. You have 15 square tiles labeled 1 through 15 within a wooden frame, arranged in a $4 \times 4$ layout in order, with the exception of the 14 and the 15 which are swapped.

```
-----------------
| 1 | 2 | 3 | 4 |
-----------------
| 5 | 6 | 7 | 8 |
-----------------
| 9 | 10| 11| 12|
-----------------
| 13| 15| 14|   |
-----------------
```

You can move the squares around within the frame making use of the sixteenth square which is empty. For example, there are two possible moves in the starting configuration: moving the 12 or the 14. If we move the 12 we would get:

```
-----------------
| 1 | 2 | 3 | 4 |
-----------------
| 5 | 6 | 7 | 8 |
-----------------
| 9 | 10| 11|   |
-----------------
```

```
| 13| 15| 14| 12|
-----------------
```

Since there is only one empty square, a sequence of moves can be written down as the sequence of tiles which are moved.

The goal now is to move the tiles around so that the 14 and 15 tiles are in the correct order.

The 14/15 puzzle was all the rage in the 1880*s*. Years later, Sam Loyd[1] promised a reward to anybody who could solve the puzzle, being well aware that solutions were impossible. There were numerous accounts of people claiming they had solved the puzzle but couldn't remember how. As it turns out the puzzle is unsolvable, as can be argued by a sophisticated parity argument. We will see the argument in its proper place: when we talk about permutations.

## 7.1 Exercises

1. Picture a chessboard; you want to traverse the chessboard starting at the upper left corner and going to the lower right corner, entering and leaving each square (except the first and last) exactly once. From a square you are only allowed to move to an adjacent square to the left, right, top or bottom (no diagonal moves).

   (*i*) Can you traverse an $8 \times 8$ chessboard this way?

   (*ii*) Can you traverse a $9 \times 9$ chessboard this way?

   (*iii*) After solving the previous two exercises formulate *and prove* a general result about traversing an $n \times n$ chessboard starting in the upper left corner and ending in the lower right corner entering and leaving every square (except the first and last) exactly once.

2. Here is a memorization card trick: a magician gives 64 cards to an audience and asks them to arrange them in a $8 \times 8$ square, randomly putting the cards face-up or face-down. The magician then adds another $17 = 8+8+1$ cards extending the $8 \times 8$ square to a $9 \times 9$ square. While he has turned around the audience can flip over a single card. When he turns back, the magician can quickly find the card that has been flipped. How does he do it? *Hint:* The solution is purely mathematical, no memorization skills are needed.

3. Try the following game: there are 10 matches on the table; two players alternate removing either one or two matches in each move. The player who removes the last match wins the game.

[1]Loyd claimed invention of the puzzle, a claim, which is as incorrect as his claim that he invented Pachesi an Indian game nearly 2000 years old. See *The* 15 *puzzle* by Slocum and Sonneveld on the history of the puzzle and many pictures of different version.

(*i*) Show that the player who goes first in this game can always win the game, by describing a winning strategy. *Hint:* A winning strategy is a set of rules that the player can follow in each move that guarantees a win in the end.

(*ii*) If we start with 11 matches on the table, who can force a win, the first or the second player? Give a winning strategy for the player.

(*iii*) Phrase a general result on which player can force a win in this game if we start with $n$ matches on the table, $n > 0$.

## 7.2 Notes and Additional Reading

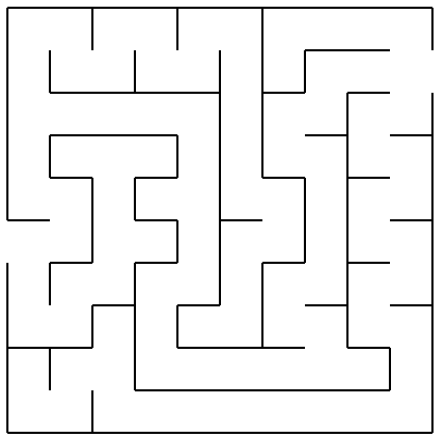
See *The* 15 *puzzle* by Slocum and Sonneveld for an amazingly detailed history of the 14/15 puzzle with numerous pictures of early versions of the puzzle.

# Chapter 8

# Games and Graphs

## 8.1 On Mazes

Let us start with a simple maze.



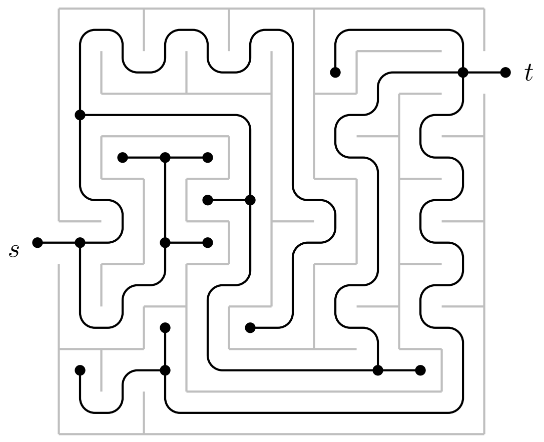
When you search for a path through a maze, how do you proceed?[1] We can distinguish two variants of the problem: local or global. Local means you are in the maze, and you can only see the part of the maze surrounding you. For that type of maze techniques include (*i*) the right-hand rule (place your hand on the right wall and keep it there while walking), (*ii*) the left-hand rule (similar), or (*iii*) you can simply start walking around making random decisions about where to continue when the path forks (a technique that works very well, and doesn't require you to memorize anything). A global technique means you have the layout of the maze in front of you to analyze it. Somewhere in between local

[1]If you are in Illinois, you can test your maze solving powers on foot at the Richardson Corn Maze, `http://www.richardsonfarm.com/r-maze.htm`, or the hedge maze at the Morton Arboretum, `http://www.mortonarb.org/maze/index.php`.

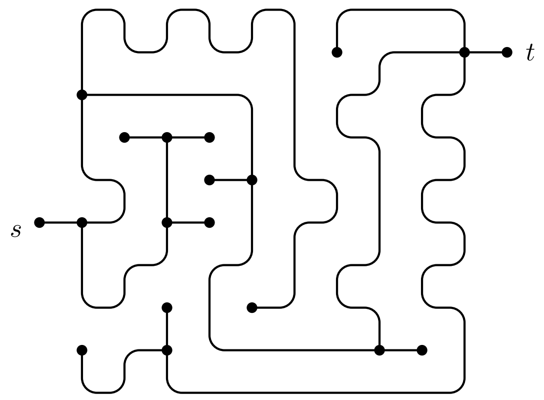
and global are using breadcrumbs and Ariane's thread, drawing a plan of the maze, and marking intersections.

**Exercise 8.1.1.** Draw a maze for which the right-hand rule fails. Draw a maze for which both the right-hand rule and the left-hand rule fail.

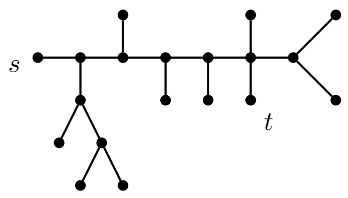
We concentrate on the global perspective. The drawing of a maze contains a lot of extraneous geometric information; we can simplify the maze quite a bit if we concentrate on the important parts: entrance, exits, forks and dead ends. We place nodes at those special locations and connect them by lines to model the connections between those locations.



Indeed, we can now imagine the nodes and their connections without the maze, and the resulting object still contains all the information we need to solve the maze:



Even more, we can now redraw the nodes and their connections without regard to the particular geometry of the maze, obtaining what is called a graph:



At this point it has become trivial to see how to go from the entrance node $s$ to the exit node $t$.

Note that in building the graph we abstracted nearly all geometric information away from the maze—except for the order in which the connections leave the nodes; this makes it particularly easy to translate a walk through the graph back into a walk through the maze.

The object we have constructed is known as a graph. A *graph* is a collection of *vertices* (also known as *nodes*) which can be connected by *edges*. More formally, a graph $G$ is a pair $G = (V, E)$ of vertices $V$ and edges $E$, where an edge in $E$ is a set of two vertices. For vertices we typically use letters such as $u, v, s, t, w$ and for edges $e, f, g$. If $e$ is the edge from $u$ to $v$ then, formally, $e = \{u, v\}$ but we will typically write $e = uv$ for simplicity. The vertices $u$ and $v$ are the *ends* of the edge $uv$, and we say that $u$ and $v$ are *incident* to the edge $uv$. Since $e$ is the set $\{u, v\}$ and $\{u, v\} = \{v, u\}$ we do not distinguish between $uv$ and $vu$ (at least not in this section).

**Remark 8.1.2.** By the definition of an edge we do not allow an edge from a vertex to itself or multiple edges between the same pair of vertices. Those features are allowed in *multigraphs*, though many books will also call these objects graphs (and call what we call graphs, simple graphs).

We translated the maze into a graph. Walking through a maze corresponds to the graph-theoretical notion of a walk. A *walk* is a sequence $u_1, e_1, u_2, e_2, \dots, e_{n-1}, u_n$ of vertices and edges that traverses the graph; i.e. if you start at $u_1$, $e_1$ takes you to $u_2$, etc. In other words, $e_1 = u_1u_2$, $e_2 = u_2u_3$, and so on up to $e_{n-1} = u_{n-1}u_n$.

There are two special types of walks that will interest us: A *path* is a walk in which every vertex occurs at most once, and a *cycle* is a trail with $n \geq 3$ whose first and last vertex are the same, $u_1 = u_n$, but there are no other vertex repetitions. (We exclude the case $n = 2$ as a cycle, since it simply means we walk back and forth along the same edge.)

**Exercise 8.1.3.** A *trail* is a walk which contains every edge at most once. By definition, every trail is a walk. Show that in a path every edge can occur at most once (this shows that every path is a trail). On the other hand construct a graph and a walk in the graph in which every edge occurs at most once, but there are vertices that occur multiple times. (So not every trail is a path.)

**Proposition 8.1.4.** *If $G$ contains a walk from $s$ to $t$ then it contains a path from $s$ to $t$.*

*Proof.* If $G$ contains a walk from $s$ to $t$, let

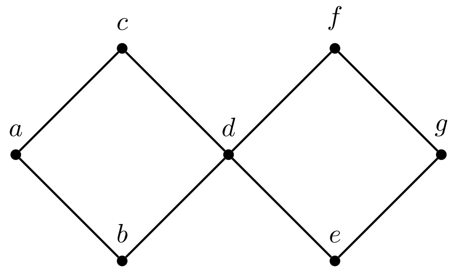
$$u_1, e_1, u_2, e_2, \ldots, e_{n-1}, u_n$$

be a shortest such walk. Suppose some vertex occurs twice on that walk, that is $u_i = u_j$ for some $1 \leq i < j \leq n$. Then

$$u_1, e_1, \ldots e_{i-1} u_j e_j u_{j+1} \ldots u_n$$

also is a walk from $s$ to $t$ contained in $G$ and it is shorter than the original walk. So we arrived at a contradiction by assuming that the shortest walk contains a repeated vertex. Hence, the shortest walk does not contain a repeated vertex and is therefore the path we are looking for. ■

**Example 8.1.5.** Let us study the following graph:



In this graph,

$$a, ab, b, bd, d, de, e, eg, g, gf, f, fd, d, dc, c$$

is a walk from $a$ to $c$. It is not a path, since the vertex $d$ is repeated (it occurs twice), however, the excursion $d, de, e, eg, g, fg, f, fd, d$ is unnecessary to get from $a$ to $c$ and can be removed from the walk. We get a path $a, ab, b, bd, d, dc, c$ from $a$ to $c$. Not the shortest path, since we could have simple walked $a, ac, c$, but a path. Note that the excursion we cut out of the walk is a cycle:[2]

$$d, de, e, eg, g, fg, f, fd, d.$$

There is another cycle in this graph: $a, ab, b, bd, d, dc, c, ac, a$. Note that we consider $a, ab, b, bd, d, dc, c, ac, a$ to be the same cycle as $b, bd, d, dc, c, ac, a, ab, b$, the second is just another way of writing the same cycle.[3] With this convention there are exactly two cycles in the graph; in particular,

$$a, ab, b, bd, d, de, e, eg, g, fg, f, fd, d, dc, c, ca, a$$

is not a cycle, since it uses $d$ twice.

[2]Should this have been $g, gf, f$ instead of $g, fg, f$? Doesn't matter since $fg$ and $gf$ are the same thing: an edge between $f$ and $g$, we are not distinguishing different directions yet.

[3]If we wanted to be a bit more formal here, we'd say that a cycle is the equivalence class of certain types of sequences under rotation.

Solving a maze corresponds to finding a path between two given vertices $s$ and $t$ in a graph. How do we do this? For small graphs trial and error seems fine, but what if the graphs are too large (imagine a graph representing the network of streets in the US)? How can we systematically determine whether we can get from $s$ to $t$ and if so, how?

There are two approaches to this problem, the conservative approach that starts at $s$ and sees what vertices can be reached from $s$ directly, that is, by an edge; this set of vertices is known as the *neighborhood* of the vertex:

$$N(u) := \{v : uv \in E\}.$$

If $t$ is among the vertices, $N(s)$, we are done, indeed, we can get from $s$ to $t$ in a single step. If not, we check the neighborhoods of the vertices in $N(s)$, that is, the vertices that can be reached in *two* steps from $s$, or, in other words, the neighborhood of the neighborhood of $s$. There is no reason to stop at a distance of 2 from $s$, of course. Let

$$N_k(u) := \{v : v \text{ can be reached in at most } k \text{ steps from } u\},$$

the *$k$-neighborhood* of $u$. Then $N_0(u) = \{u\}$ and $N_1(u) = N(u) \cup \{u\}$.

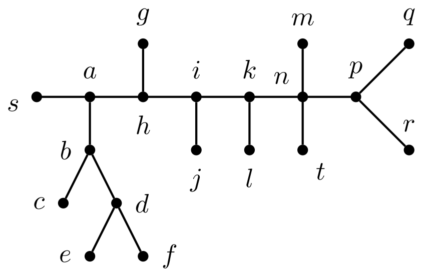
We saw that the $k$-neighborhood arises very naturally step by step (or *inductively* or *recursively*) as follows: $N_0(u) = \{u\}$ and for $k > 0$ we have:

$$N_k(u) := N_{k-1}(u) \cup \{w : vw \in E \wedge v \in N_{k-1}\}.$$

These sets correspond exactly to the conservative method we outlined above, known as *breadth-first search*: we investigate larger and larger neighborhoods of $s$ until we find $t$; we start with $N_0(s)$, and check $N_1(s)$, $N_2(s)$, and so on until we find the first $k$ such that $t \in N_k(s)$. This even gives us the smallest number of steps needed to get from $s$ to $t$.

**Remark 8.1.6.** In terms of the original maze, we first venture for one step in each direction from the entrance, returning to the entrance after each step. Then we start going two steps in all possible directions, always coming back. Then three steps, and so on. In a physical maze we need some way of finding our way back (breadcrumbs, or Ariane's thread will do), and we need to make sure we systematically try all possible routes (we can try them in clockwise or counterclockwise order, for example).

**Example 8.1.7.** Let us see what a breadth-first search for our sample maze looks like; we will perform it on the graph we abstracted from the maze. To distinguish all the vertices we have given them names, called *labels*.



We begin at $s$, $N_0(s) = \{s\}$. In at most one step we can reach $N_1(s) = \{s, a\}$, in two steps: $N_2(s) = \{s, a, b, h\}$. We see that it is more interesting to look at the new vertices we obtain; let's call that the *k-boundary*, $B_k(u)$; those are the vertices that can be reached in $k$ steps from $u$, but not by any shorter path; that is, $B_k(u) := N_k(u) - N_{k-1}(u)$. Then $B_0(s) = \{s\}$, $B_1(s) = \{a\}$, $B_2(s) = \{b, h\}$. Like a wave rippling through the maze. Here is what we see:

$$\begin{aligned}
B_0(s) &= \{s\}\\
B_1(s) &= \{a\}\\
B_2(s) &= \{b, h\}\\
B_3(s) &= \{c, d, g, i\}\\
B_4(s) &= \{e, f, j, k\}\\
B_5(s) &= \{l, n\}\\
B_6(s) &= \{m, t, p\}\\
B_7(s) &= \{q, r\}\\
B_8(s) &= \{\}
\end{aligned}$$

and at that point we have seen the whole graph.

Using pseudocode, we can describe the strategy of a breadth-first search more precisely; for the code we will use a queue. A *queue* is a set (and we write it as a set), in which elements are added to the end and taken out at the beginning (like a queue at the post-office). That is, a set in which order matters. We use operators append$(q, x)$ to add element $x$ to the end of queue $q$, and pop$(q)$ to unqueue the first element of $q$.

While performing the traversal we simultaneously calculate the distance of each vertex from $s$. We keep this information in an array `distance`. Initially, all vertices have a distance of infinity from $s$, they are unvisited. As the algorithm proceeds and we look at an unvisited neighbor $v$ of a vertex $c$ we know that $v$ has distance one more from $s$ than $c$ (since otherwise we would have already encountered it). We also use the distance information to tell whether we have already visited a vertex or not: a vertex at a distance of infinity is one that has not been visited yet. In the following code we use $-1$ to denote infinity.

```
procedure breadth_first_search(G,s,t)
for each v in G         // initially, every vertex+
   distance[v] := -1    // has distance infinity (we use -1)

reached := {s}          // current boundary
distance[s] := 0        // s has distance 0 from itself

while reached <> {}     // vertices whose neighbors need to be explored
   c := pop(reached)    // take the first element of the queue
   if (c = t)           // found the exit
      return true
   for each v in N(c)        // every element of c's neighborhood
      if distance[v] = -1    // which has not been visited yet
         append(reached, v)           // is appended to the queue
         distance[v] := distance[c] + 1    // distance is one more than c
return false            // exit not found
```

If the program reaches its last line, then all vertices that could be reached from $s$ have been visited and $t$ was not among them (otherwise we would have returned `true` and stopped), so we return false. The procedure breadth_first_search$(G, s, t)$ tells us whether there is a path from $s$ to $t$ in $G$. This problem is known as *st-connectivity*. We need some modifications to the code to trace the actual path from $s$ to $t$.

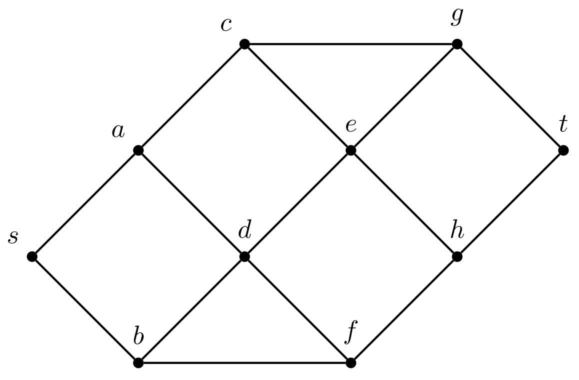
**Example 8.1.8.** Let us go through our maze above using the new procedure. Initially, `reached` $= \{s\}$. We pop $s$ off the queue, and note that $s$ has been visited: `distance[s]` $= 0$. We then attach $N(s) = \{a\}$ to `reached`, which is now $\{a\}$. Next, we pop $a$ off the queue and note that its distance is

$$\texttt{distance[a]} = \texttt{distance[s]} + 1 = 1.$$

We then go through $a$s neighborhood, $N(a) = \{s, b, h\}$, and add its unvisited neighbors ($b$ and $h$) to the queue: `reached` $= \{b, h\}$. Next we pop $b$ and store `distance[b]` $= 2$; $b$'s neighbors are $a$, $c$ and $d$. We enqueue the unvisited neighbors $c$ and $d$ and `reached` now equals $\{h, c, d\}$. And so on.

**Exercise 8.1.9.** (S) Finish the previous example; show the state of the queue `reached` after each stage and the distances that have changed.

**Exercise 8.1.10.** Perform a breadth-first search starting at vertex $a$ of the following graph; show the state of the queue `reached` after each stage.



The distance calculations in breadth-first search work correctly, since breadth-first search proceeds conservatively, it always finds a shortest path from $s$ to $t$, a useful property to have. On the other hand, it does go into breadth, visiting every neighbor even in the most unpromising direction, before going one step farther into depth. At the other extreme there is *depth first search* which first goes into depth, as far as it can go, and only when it has reached a dead end will it turn back (*backtrack*) and consider alternatives. A very useful strategy, and a good balance to breadth-first search.

We can describe the strategy of depth-first search as follows: starting with a vertex $s$ pick one of its neighbors and then continue depth-first searching from that vertex, until you have explored all possible connections. At that point return to $s$ and try the next neighbor of $s$ that has not been visited yet. Continue until you have seen all neighbors of $s$.

Let us assume that initially all vertices are unvisited, that is we have run the code

```
for each v in G         // initially, every vertex
   visited[v] := false  //     is unvisited
```

We now traverse the graph in depth; the following procedure traverses $G$ depth first starting at $s$.

```
procedure depth_first_traverse(G,s)
visited[s] := true
for each v in N(s)
    if not visited[v]   // the edge to v has not been explored yet
       depth_first_traverse(G,v) // visit v
```

So a depth-first search first needs to initialize the information about visited vertices and then call on `depth_first_traverse` to actually perform the search.

```
procedure depth_first_search(G,s,t)
for each v in G         // initially, every vertex
   visited[v] := false  //     is unvisited
```

```
depth_first_traverse(G,s)
if visited[t]
   return true // t was found during the traversal
else
   return false // t was not found
```

**Example 8.1.11.** Let's depth-first search our original maze; here are the calls that are made in order:

```
depth_first_traverse(G,s)
depth_first_traverse(G,a)
depth_first_traverse(G,b)
depth_first_traverse(G,c)
depth_first_traverse(G,d)
depth_first_traverse(G,e)
depth_first_traverse(G,f)
depth_first_traverse(G,h)
depth_first_traverse(G,g)
depth_first_traverse(G,i)
depth_first_traverse(G,j)
depth_first_traverse(G,k)
depth_first_traverse(G,l)
depth_first_traverse(G,n)
depth_first_traverse(G,m)
depth_first_traverse(G,p)
depth_first_traverse(G,q)
depth_first_traverse(G,r)
depth_first_traverse(G,t)
```

We typically abbreviate the traversal by simply listing the vertices in the order that they are visited, so in this case we could have written:

$$s, a, b, c, d, e, f, h, g, i, j, k, l, n, m, p, q, r, t.$$

Note that our depth-first traversal went through the whole maze before finding the exit. We could speed up the algorithm by aborting the search once the exit $t$ has been found. For that we need to modify the code for both search and traversal. Let us call these new versions `depth_first_traverse'` and `depth_first_search'`:

```
procedure depth_first_search'(G,s,t)
for each v in G         // initially, every vertex
   visited[v] := false  //     is unvisited
depth_first_traverse'(G,s,t)
if visited[t]
   return true // t was found during the traversal
```

```
else
   return false // t was not found


procedure depth_first_traverse'(G,s,t)
visited[s] := true
if s = t
    return control to depth_first_search'
for each v in N(s)
    if not visited[v]   // the edge to v has not been explored yet
       depth_first_traverse'(G,v,t) // visit v
```

These algorithms can do better than the original ones if the neighborhoods are added in a more felicitous order, since the algorithm will stop once it's found $t$ rather than continuing to traverse the rest of the graph. In the original traversal we always added neighbors in alphabetical order, let us try another order here:

```
depth_first_traverse'(G,s,t)
depth_first_traverse'(G,a,t)
depth_first_traverse'(G,h,t)
depth_first_traverse'(G,i,t)
depth_first_traverse'(G,k,t)
depth_first_traverse'(G,n,t)
depth_first_traverse'(G,t,t)   // found t
```

or, for short,

$$s, a, h, i, k, n, t.$$

In this last traversal, depth-first search found the shortest path from $s$ to $t$; there are many unexplored corridors left in the maze, but we don't care, since we have found the exit. The problem, of course, is that we don't know how to pick the ordering of the neighbors wisely.

**Exercise 8.1.12.** Traverse the graph from Exercise 8.1.10 using the original `depth_first_search`.

**Exercise 8.1.13.** Pick one of the easy mazes at `http://www.mazes.org.uk/`.

($a$) Draw the graph corresponding to the maze.

($b$) Perform a depth-first search on the maze. (List the vertices in the order that they are visited.)

($c$) Perform a breadth-first search on the maze. (Show the details of how `reached` and `distance` change.)

($d$) Does the maze contain cycles? Is there more than one solution?

How could it happen that there is no path from $s$ to $t$? In that case $s$ and $t$ must be in different parts of the graph, the graph is *disconnected*, that is, there are two vertices in the graph which are not connected to each other by a path.

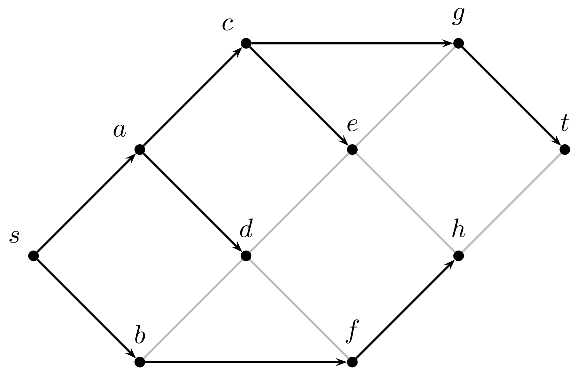
**Exercise 8.1.14.** Draw a maze on a $6 \times 6$ grid that has no solution. Draw the graph corresponding to the maze.

**Exercise 8.1.15.** Draw a small graph which is disconnected. How small can you make the graph?

**Exercise 8.1.16.** Let $G = (V, E)$ be a graph. Let $P(u, v, P)$ be the ternary relation expressing that $P$ is a path in $G$ with endpoints $u$ and $v$. Write a quantified formula expressing that a graph $G$ is connected.

Both breadth-first search and depth-first search will visit all the vertices that are reachable from a given vertex $s$. So we can use either one of them to tell whether the whole graph is *connected*, that is, every pair of vertices are reachable from each other: simply run one of the search algorithms on an arbitrary vertex and count how many vertices are visited. If the count is the same as the total number of vertices in the graph, the graph is connected; otherwise it is not.

Let us run one of our search algorithms on the graph from Exercise 8.1.10 and let us keep track of from which vertex another vertex is visited. We use a breadth-first traversal, where `reached` goes through the following stages: $\{s\}$, $\{a, b\}$, $\{b, c, d\}$, $\{c, d, f\}$, $\{d, f, e, g\}$, $\{f, e, g\}$, $\{e, g, h\}$, $\{g, h\}$, $\{h, t\}$, $\{t\}$. Going through this list again, we see that $a$ and $b$ are visited from $s$; $c$ and $d$ are visited from $a$; $f$ is visited from $b$; $e$ and $g$ are visited from $c$; $h$ is visited from $f$; $t$ is visited from $g$. Pictorially, we can represent this by arrows pointing from a vertex to the vertex whose visit it causes. (Edges that are not involved in the visiting structure are drawn in gray.)



Observe that the heavy edges form a special type of graph, a tree; a *tree* is a connected graph which does not contain a cycle.

**Exercise 8.1.17.** Show that a graph is a tree if and only if there is a unique path between any two of its vertices. *Hint:* It is easy to show one direction:

a graph containing a cycle has two vertices that do not have a unique path between them. The other direction needs a bit more care.

What our search procedures show is that every connected graph contains a tree on all the vertices of the graph. Such a tree is called a *spanning tree.*

**Theorem 8.1.18.** *Every connected graph contains a spanning tree.*

*Proof.* Let $G$ be the connected graph. Run depth-first search or breadth-first search on $G$; during the search, if some vertex $v$ is visited from some vertex $u$, then draw an arrow from $u$ to $v$. Consider the graph $T$ that consists of all the arrowed edges. We claim that $T$ contains the same vertices as $G$. The reason is that $G$ is connected, so the search reaches all vertices of $G$. Observe that every vertex is visited at most once, so there is at most one arrow pointing *towards* each vertex. Suppose $T$ contained a cycle $C$. Let $v$ be the vertex on $C$ that was visited first in the search. Then the two edges of $C$ adjacent to $v$ point away from $v$ (since it was the first vertex on $C$). But then $C$ must contain two edges that point towards the same vertex (follow the arrows starting at $v$ until you run into an edge pointing the other way; that must happen, since the two edges at $v$ point in different directions along $C$). But this is not possible: every vertex has at most one arrow pointing at it. ■

In this section, we have seen how to solve three fundamental problems on graphs using two traversal techniques: breadth-first and depth-first search.

- *st-connectivity problem*: given a graph $G$ and two vertices $s$ and $t$ can $t$ be reached from $s$. Use either breadth-first or depth-first search.
- *shortest path problem*: given a graph $G$ and two vertices $s$ and $t$ find a shortest path from $s$ to $t$ if one exists. Use breadth-first search.
- *spanning tree problem*: given $G$ find a tree that connects all vertices of $G$.

All of our problem solutions are very fast, since they are based on fast search algorithms: every vertex is visited only once in both breadth-first and depth-first search, so the running time of our algorithms is proportional to the size of our graphs.

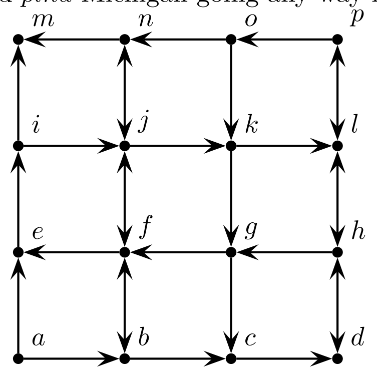
## Directed Graphs

If you wanted to use graphs to model the street plan of a city you would soon realize that our graphs have one deficiency: there is no difference between the edge $uv$ and the edge $vu$, they both correspond to $\{u, v\} = \{v, u\}$, we cannot distinguish direction. In other words: We have no one-way streets. This is easily remedied by using directed graphs. A *directed graph* (or *digraph*) $G = (V, E)$ consists of a set of vertices $V$ and a set of edges between vertices: $E \subseteq V \times V$. That is, an edge is now a directed pair: $(u, v)$ differs from $(v, u)$ (unless $u = v$, but we do not allow an edge from a vertex to the same vertex). We will write

$uv$ for $(u,v)$ for the directed edge from $u$ to $v$.[4] We draw a directed edge $uv$ with an arrow pointing from $u$ to $v$; $u$ is sometimes called the *child* vertex and $v$ the *parent* vertex, in particular if we are traversing the edge in the direction of the arrow: we go from a child to a parent.

We have encountered directed graphs already, although we didn't call them that: when analyzing the search procedures to prove the existence of a spanning tree, we drew that tree as a directed tree, pointing from a vertex $u$ to a vertex $v \in N(u)$ if $v$ is visited from $u$.

**Example 8.1.19.** The following graph is a graph representation of the streets in a $4 \times 4$ block of downtown Chicago. (*abcd* is Jackson going east, *ponm* is Madison going west, *aeim* Dearborn going north, *okgc* Wabash going south, and *plhd* Michigan going any way it wants.)



Suppose you want to drive from the Art Institute (which is at $d$ to the corner of Monroe and Wabash at $k$). What are your options? What are the quickest options? Obviously, it makes a difference whether we have to respect one-way streets or not.

**Exercise 8.1.20.** Pick a $4 \times 4$ block of your favorite downtown city, and model the way cars can travel using a directed graph. Is there an example, where the shortest path ignoring one-way streets differs from the shortest path respecting one-way streets?

Breadth-first search and depth-first search work fine on directed graphs, we don't even need to change the algorithm if we interpret the definition of the neighborhood of a vertex $u$,

$$N(u) := \{v : uv \in E\}$$

as referring to the directed edge $uv$. Breadth-first search will now give us the shortest *directed path* from $s$ to $t$, that is, with all edges along the path oriented from $s$ to $t$ (you are not driving backwards through a one-way street), and both breadth-first and depth-first search will find all the vertices reachable

[4]So is $uv$ directed or undirected? Depends on the context, i.e. whether we are talking about a directed graph or not.

from a particular vertex. There is one difference though, and that is in the notion of connectivity: if we remove the direction from the edges, that underlying undirected graph might be connected, without there being directed paths between all pairs of vertices: directed connectivity is a much stronger requirement than undirected connectivity (as those of you that like to drive through one-way streets the wrong way certainly know). We call a directed graph *strongly connected* if there is a *directed* path between any two vertices.

**Exercise 8.1.21.** Let $G = (V, E)$ be a graph. Let $R(u, v, P)$ be the ternary relation expressing that $P$ is a directed path in $G$ from $u$ to $v$. Write a quantified formula that expresses that the graph $G$ is strongly connected.

One consequence is that the notion of tree is no longer so interesting for directed graphs; it is replaced with the notion of a *directed acyclic graph*—better known as a *dag*, that is a graph that does not contain any directed cycle (all the edges in the cycle pointing in the same direction along the cycle).

**Exercise 8.1.22.** Construct a small directed graph which is acyclic, but whose underlying undirected graph contains an (undirected) cycle.

Dags occur very naturally when ordering objects: if $\prec$ is a strict, even partial, ordering relation on a set of objects $V$, then the graph $G = (V, E)$ defined by $E = \{uv : u \prec v\}$ is a dag.

**Exercise 8.1.23.** Prove that the graph $G = (V, E)$ defined by $E = \{uv : u \prec v\}$ is a dag if $\prec$ is a strict, partial ordering relation. *Hint:* the main thing to show is that $G$ is acyclic; this follows from anti-symmetry.

We can now show a result we hinted at earlier: every partial ordering can be extended to a total ordering which is consistent with the partial ordering.

**Theorem 8.1.24.** *If $\prec$ is a strict, partial ordering, there is a strict, total ordering $\prec'$ such that $x \prec y$ implies $x \prec' y$ for all $x$ and $y$.*

The proof will show how to construct the total ordering explicitly. Before we see the proof, let us do an example.

**Example 8.1.25.** Let us illustrate the strict, partial, ordering $x \prec y =$"course $x$ is a prerequisite of course $y$", or, in other words, "course $y$ requires course $x$"; we visualize this relationship by drawing an arrow pointing towards the required course.

$$\begin{tikzcd}
 & & \text{CSC383} \arrow[drr] & & & \text{CSC352} \arrow[dl] \arrow[d] \\
\text{CSC347} \arrow[urr] \arrow[rrr] \arrow[dr] & & & \text{CSC373} \arrow[r] & \text{CSC212} \arrow[d] & \text{CSC351} \arrow[d] \\
 & \text{CSC374} \arrow[urr] & & \text{IT130} & \text{CSC211} \arrow[l] & \text{IT240}
\end{tikzcd}$$

For example, you cannot take CSC 373 before having taken CSC 212, which in turn requires CSC 211, which requires IT 130.

**Exercise 8.1.26.** This exercise refers to the course prerequisite example above.

(*i*) List all courses you must have taken before you can take CSC 352.

(*ii*) List all courses you must have taken before you can take CSC 383.

(*iii*) Find a longest (directed) path in the prerequisite graph. What meaning does the length of that path have to somebody wanting to take all the classes listed? *Hint*: A longest path in diagrams of this type is often known as a *critical path*. Why?

Doing the example might have given you a hint on how to construct the ordering: we can take a course once we have taken all the prerequisites. If we think of the dag associated with the prerequisite ordering we can express this as saying: once we have taken care of all the children of a vertex, we can take the course represented by the vertex. But this is exactly the information that depth-first search gives us: at the point where a vertex is popped off the stack in depth-first search, all its children have been visited. So all we have to do is slightly modify the depth-first search procedure. Let us assume that initially all vertices are unvisited, that is, we have run the code

```
for each v in G         // initially, every vertex
   visited[v] := false  //     is unvisited
```

We can then pick some vertex $s$ and run the following modified depth-first search:

```
procedure topological_traverse(G,s)
visited[s] := true
for each v in N(s)
    if not visited[v]   // the edge to v has not been explored yet
       topological_traverse(G,v) // visit v
print(s) // all children of s have been explored
```

Since $G$ might not be strongly connected, we might have to restart the depth-first traversal for vertices that haven't been visited yet (that happened in the course prerequisite example as well). So our sorting algorithm, known as *topological sort* looks as follows:

```
procedure topological_sort(G)
for each v in G         // initially, every vertex
   visited[v] := false  //     is unvisited

for each v in G
   if visited[v] = false
      topological_traverse(G,v)
```

This algorithm will print out the vertices in an order which is consistent with the ordering represented by the dag.

**Exercise 8.1.27.** Produce a topological sort of the prerequisite example we saw above. Use the topological sort to design a more readable drawing of the prerequisite structure of those courses.

In this section we have seen directed and undirected graphs, two very fundamental modeling tools; there are many refinements of these models available, as needed: for example you can color edges and vertices (to express that they belong to different categories), and you can add weight to edges and vertices; e.g. an edge weight in a graph could correspond to the distance between two vertices representing physical locations. A shortest path between two vertices would be a minimum distance connection. You can see how Google maps would find this model useful.

In the end, though, the surprising revelation (already hinted at when we talked about orderings) is that graphs are just a visualization of binary relations: directed graphs of arbitrary binary relations, undirected graphs of symmetric binary relations.

**Exercise 8.1.28.** What does transitivity in a binary relation correspond to in the graph representing the relation? What about reflexivity and anti-reflexivity?

## 8.2 Euler Tours

Graph theory began with a famous paper by Leonard Euler on the Königsberg bridge problem. His question was whether one could take a walk through Königsberg which would cross each of the seven bridges of Königsberg exactly once. Instead of Königsberg let us consider a fictitious city, let's call it K, with rivers and bridges as shown in the next picture:

```
   1    ~~~~~~~~~~~~~~~~||~~~~~~~~~~~~~~
        ~          ~
        ~          ~
       ===    3   ===          4
        ~          ~
        ~          ~
~~||~~~~~~~~~||~~~~~~~~~~~~||~~~~~~
                    ~
       2            ===        5
                      ~
                       ~~~~~~~~~~~~
```

Can we walk through K crossing each bridge exactly once? Let us remove some of the unnecessary details of the problem by modeling it as a graph: the pieces of land become vertices and the bridges turn into edges.

**Exercise 8.2.1.** Model the bridge problem as a graph.

We can write the graph as: $K = (\{1,2,3,4,5\}, \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,5\}, \{3,4\}, \{4,5\}\})$.

Euler's problem is asking for a walk through this graph that contains every edge exactly once. Such a walk is called an *Eulerian trail* (the walk is a trail since each edge occurs at most once). A graph is called *Eulerian* if it contains an Eulerian trail. Note that we allow the special case where the start and end-vertex of the trail are identical. In that case the trail becomes an *Eulerian cycle*, also known as an *Eulerian tour*.

Now let us show that the bridge problem for the city of K cannot be solved. Imagine that there were a trail $s = u_1, e_1, u_2, e_2, \ldots, e_7, u_8 = t$ containing each edge of the graph exactly once. Note that, with the exception of $s$ and $t$ we leave each vertex as often as we enter it, so all vertices other than $s$ and $t$ must be adjacent to an even number of edges; $s$ and $t$, on the other hand, must be adjacent to an odd number of edges (adding one for the first leaving of $s$ and the last return to $t$).

The number of edges adjacent to a vertex $v$ in $G$ is called the *degree* of $v$, also written $\deg_G(v) = |\{e : v \in e\}|$. Note that $\deg_G(v) = |N(v)|$ (since each neighbor is reached by precisely one edge that $v$ is adjacent to; this would not be true for multigraphs).

What we have shown in general is:

**Theorem 8.2.2** (Euler). *If a graph $G$ has an Eulerian trail from $s$ to $t$ (with $s \neq t$), then all vertices except for $s$ and $t$ have even degree and $s$ and $t$ have odd degree. If $G$ contains an Eulerian tour, then all vertices have even degree.*

**Example 8.2.3.** The graph $K$ from our opening example has vertices $1, 2, 3, 4, 5$ of degrees $3, 3, 3, 3, 2$, respectively. By Euler's theorem there cannot be an Eulerian trail or tour in the graph.

The theorem establishes a *necessary* condition for being Eulerian: the graph must have either two or no vertices of odd degree. If $P \rightarrow Q$ then for $P$ to be true, $Q$ has to be true as well, since $P \rightarrow Q$ is equivalent to $\overline{Q(x)}$ implies $\overline{P(x)}$; that is, if $Q$ fails to be true, then so does $P$. Under these circumstances we say that $Q$ is a necessary condition for $P$. If on the other hand $Q \rightarrow P$, then $Q$ is a *sufficient* condition for $P$, since if $Q$ is true, then $P$ is also true.

**Example 8.2.4.** Writing a dissertation is a necessary condition for getting a PhD, but it is not sufficient (one might fail the exam or not take the required course work). Being born in Illinois is a sufficient condition for being born in America, but it is not necessary. One could have been born in Vermont.

Being featherless and a biped is a necessary and sufficient condition for being human (or so philosophers like to claim).

**Exercise 8.2.5.** Which of the following are true? (Assume the philosophical definition of a human as a featherless biped.) If a claim isn't true, state a counterexample that shows it is not true.

1. Being featherless is a necessary condition of being human.
2. Being featherless is a sufficient condition of being human.
3. Being human is a necessary condition of being featherless.
4. Being human is a sufficient condition of being featherless.

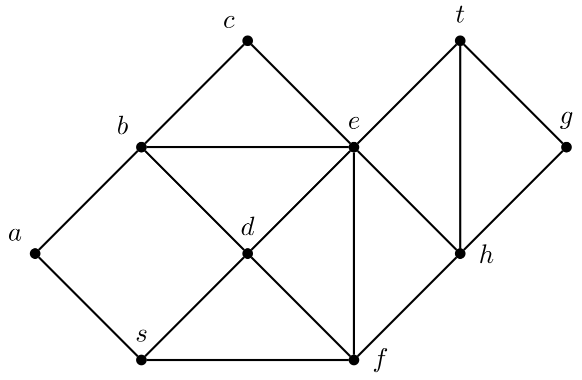
Interestingly, Euler's theorem is an instance of a more general phenomenon in which a necessary condition turns out to be sufficient. Well, nearly sufficient, we also need to assume that the graph is connected: it is easy to construct a graph which is not connected and for which all vertex degrees are even. However, there cannot be an Eulerian trail in that graph, since it is not even connected, so graphs that are not even connected are not interesting in this context.

**Exercise 8.2.6.** Draw a graph in which every vertex has even degree but which is not connected. What is the smallest such graph?

**Theorem 8.2.7.** *If $G$ is connected, then $G$ is Eulerian if and only if all vertices of $G$ have even degree or exactly two vertices of $G$ have odd degree. More precisely, if all vertices of $G$ are even, then $G$ contains an Eulerian tour. If exactly two vertices of $G$ are odd, then there is an Eulerian trail between those two vertices.*

Euler claims, but does not prove, this result in his paper. Instead of a formal proof, which gets technical, we illustrate the idea of the proof in several examples.

**Example 8.2.8.** By Euler's theorem, the following graph contains an Eulerian trail starting at $s$ and ending at $t$.



Let us start at $s$ and look for a trail through the graph; we cannot get stuck at a vertex of even degree, since we enter it as often as we leave it (reducing the available edges at the vertex by 2, which always leaves an even number).

Hence, our trail (whatever choices we make) has to end at $t$. For example, we might have chosen

$$s, sd, d, db, b, be, e, ed, d, df, f, fe, e, et, t.$$

After removing the edges in this trail all vertices in the remaining graph have even degree; hence, if we start looking for an Eulerian trail starting at one of the vertices that is still incident to some edge, we will eventually have to return to that vertex, giving us a cycle in the graph. We can add that cycle to the trail we already have, and continue the process until we have run out of edges in the graph. This will give us an Eulerian trail for the original graph (do this, by hand, on the graph; try different alternatives).

Euler's criterion has a nice property: it is easily verified, all we have to do is count the number of edges leaving each vertex and check that at most two of those degrees are odd. In the program below we write `x % y` for the remainder of dividing $x$ by $y$. Then `x % 2` is 0 if $x$ is even and 1 if $x$ is odd.

```
procedure Eulerian(G)
odd_degree_vertices = 0
for each v in V(G)
   degree := 0 // to compute degree of v
   for each u in N(v)
      degree := degree + 1
   if degree % 2 = 0
      odd_degree_vertices = odd_degree_vertices + 1
if odd_degree_vertices > 2
   return false
else
   return true
```

The running time of this algorithm is proportional to the number of edges and vertices, since we look at each vertex at most once, and at each edge at most twice (for each of its endpoints).

**Exercise 8.2.9.** Formulate a version of Euler's criterion for directed graphs.

The criterion for being Eulerian states that either none or two of the vertices are of odd degree. This suggests a question: What does a graph look like which has exactly one vertex of odd degree. Can you draw one?

The answer is no, because of the following result.

**Lemma 8.2.10.** *If $G = (V, E)$ and $V = \{v_1, \dots, v_n\}$, then*

$$2|E| = \deg(v_1) + \deg(v_2) + \cdots \deg(v_n).$$

*Proof.* Look at $\deg(v_1) + \deg(v_2) + \cdots \deg(v_n)$. Every edge gets counted twice in this sum, once at each endpoint, so the sum is $2|E|$. ■

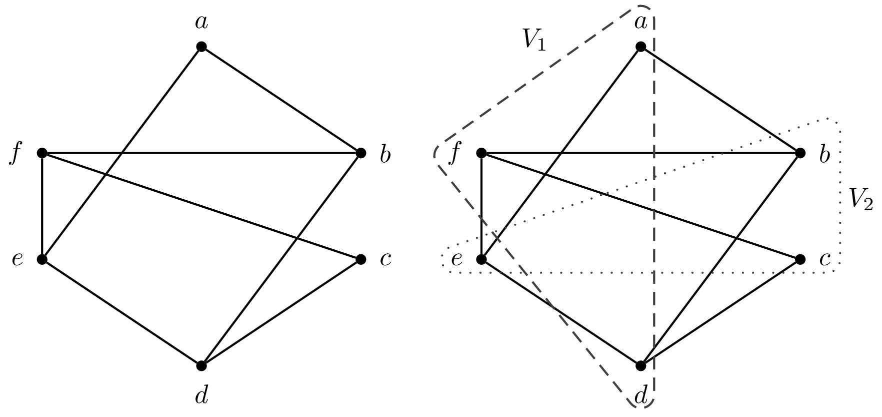
So it cannot be that a single vertex has odd degree, because then the sum of the degrees would be odd, which is not possible by the lemma. The lemma is known as the Handshake lemma: if you have a group of people some of them shaking hands, some of them not, then the number of hands involved in handshakes is even (and it equals twice the number of times hands are shaken).

You are working on the seating arrangements for a dinner. There are two tables and you are trying to place people that don't like each other at different tables. Your guests are Alice, Bob, Carol, Dan, Eve, Fred, Ginger, Homer, and Irene. We'll assume that dislike is mutual, so when we specify that Alice doesn't like Carol that also means that Carol doesn't like Alice. You know that Alice doesn't particularly care for Carol, Ginger, or Homer. Bob doesn't like Carol, Eve, and Irene. Carol can't abide Dan and Fred. Dan couldn't care less about Eve, Ginger, and Homer, while Eve doesn't like Fred. Can you seat these people at two tables?

Before you start trying to work out this example, it may pay off to think about how to model it. (Do this before continuing to read.)

We model the problem as a graph: the people in the problem become vertices and edges denote mutual aversion. What we are looking for is a partition[5] of the vertex set $V$ into two sets $V_1, V_2$ such that all edges are between vertices of $V_1$ and $V_2$; that is, there are no edges between two $V_1$ or two $V_2$ vertices. Graphs whose vertex set can be partitioned in this way are called *bipartite*.

**Example 8.2.11.** Consider the graph $G = (V, E)$ with $V = \{a, b, c, d, e, f\}$ and $E = \{ab, ae, bd, bf, cd, cf, de, ef\}$. The graph pictured on the left side in the following figure. Its vertex set can be bipartitioned into sets $V_1$ and $V_2$ such that there are no edges between any two $V_1$ vertices or any two $V_2$ vertices. The picture on the right shows that bipartition: $V_1 = \{a, d, f\}$ and $V_2 = \{b, c, e\}$.



**Exercise 8.2.12.** 1. Construct graphs which are bipartite. (Include graphs with different numbers of vertices.)

[5]Remember that $V_1, V_2$ form a partition of $V$ if $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$.

2. Construct some graphs which are not bipartite. (Include graphs with different numbers of vertices.)
3. What is the smallest non-bipartite graph? *Hint: It has three vertices.*
4. Can you find a non-bipartite graph that does not contain the graph you found in answer to the previous question?

The exercise should illustrate that cycles of odd length are an obstruction to bipartiteness; let $C_n$ denote a cycle on $n$ vertices (or $n$ edges, the same thing). Then a graph containing an odd cycle, $C_{2n+1}$ cannot be bipartite.

**Lemma 8.2.13.** *Odd cycles are not bipartite; consequently, any graph containing an odd cycle is not bipartite.*

*Proof.* Suppose $C_{2n+1}$ were bipartite. Using a simple parity argument we show that this is not possible: suppose the bipartition is $V_0$, $V_1$. Start at some vertex $s \in V_0$ and follow the cycle in one direction. Each step along the cycle takes you into the other partition. After $2n+1$ steps (an odd number) you are in $V_1$ and not back at $s$ in $V_0$.

If a graph is bipartite, then every subgraph of it is also bipartite, so a graph containing an odd cycle cannot be bipartite itself. ■

The lemma says that not containing an odd cycle as a subgraph is a necessary condition for being bipartite; as it turns out that condition is also sufficient.

**Theorem 8.2.14.** *A graph is bipartite if and only if it does not contain an odd cycle as a subgraph.*

We will give an algorithmic proof of the theorem that constructs a bipartition of $G$ if there is one (and we will show that the only obstacle to the algorithm succeeding is an odd cycle). We think of bipartitioning the vertices as coloring them: initially all vertices are without color; we will color them red and green, thereby placing them into the red and the green partition.

*Proof.* We can assume that the graph is connected; if it is not, we can separate the graph into its components and find a bipartition for each graph separately; since the different components do not have any edges between them, we can combine the different bipartitions into a single one.

Pick an arbitrary vertex $s$ of the graph and color it red; then run breadth-first search and color the layers of distance 1, 2, and so on, from $s$ alternately green and red. If we try to color a vertex that is already colored, the algorithm fails and the graph is not bipartite. If the algorithm succeeds, then the graph is bipartite, one partition being all the red vertices and the other partition consisting of the green vertices. Unvisted vertices in the algorithm are gray.

```
procedure bipartite(G)
for each v in G          // initially, every vertex
   color[v] := gray      // is gray
```

```
reached := {s}          // current boundary
color[s] := red         // s is red

while reached <> {}     // vertices whose neighbors need to be explored
   c := pop(reached)    // take the first element of the queue
   for each v in N(c)        // every element of c's neighborhood
      if color[v] = gray     // unvisited vertex
         if color[c] = red
            color[v] = green
         else
            color[v] = red
      else
         if color[v] = color[c] // both vertices are green or red
            return false        // bipartition not possible
return true
```

If the algorithm colors a vertex $u$ red, that vertex must have even distance from $s$, since the algorithm finds a shortest path from $s$ to $u$ and colors vertices along that path in alternating colors. Similarly, if $u$ is colored green, it must have odd distance from $s$. Suppose the algorithm colors two vertices $u$ and $v$ with the same color and there is an edge between $u$ and $v$. Then $u$ and $v$ must have the same distance from $s$ (their distances differ by at most one, because of the edge $uv$, but they cannot differ by one, since then the colors of $u$ and $v$ would be different, so the distances must differ by 0). Follow the shortest-path tree back starting at $u$ and $v$ until you find a common ancestor $z$ of both $u$ and $v$. Then the path from $z$ to $u$, followed by the edge $uv$, followed by the path from $v$ to $z$ forms a cycle of odd length (since the paths from $u$ to $z$ and $v$ to $z$ have the same length), but we assumed that $G$ did not contain any odd cycles. Hence, in the absence of any odd cycles, we can never run into the situation that a vertex $c$ and its child $v$ have the same color; so the algorithm succeeds in finding a coloring of the whole graph, and there are no edges between any two vertices of the same color, so the graph is bipartite.

This finishes the proof, since we argued in Lemma 8.2.13 that a bipartite graph cannot contain an odd cycle. ■

## 8.3 Graph Drawing

Ernst Dudeney, one of the great British puzzlemakers wrote:

> There are some half-dozen puzzles, as old as the hills, that are perpetually cropping up, and there is hardly a month in the year that does not bring inquiries as to their solution. Occasionally one of these, that one had thought was an extinct volcano, bursts into eruption in a surprising manner. I have received an extraordinary

> number of letters respecting the ancient puzzle that I have called "Water, Gas and Electricity". It is much older than electric lighting, or even gas, but the new dress brings it up to date. The puzzle is to lay on water, gas, and electricity, from W, G and E, to each of the three houses, A, B and C, without any pipe crossing another.

```
W         G         E



A         B         C
```

After trying this for a while, you might be giving it up as impossible. Indeed, it is impossible, but how can we see this? Quite easily, actually, all we need is to remember the Jordan curve theorem, which said that any closed curve has an inside and an outside and separates the two.

We can rephrase the water, gas, electricity problem as a graph problem as follows: let $K_{3,3}$ be the graph consisting of two sets $\{A, B, C\}$ and $\{W, G, E\}$ of three vertices each, with edges between vertices belonging to different sets. We are looking for a *planar* drawing of $K_{3,3}$, that is, a drawing without intersections. Now, $K_{3,3}$ contains the cycle $AWBGCEA$. Since we are assuming that the drawing is planar, the edges of this cycle are drawn without any intersections:

```
W----B----G
|         |
|         |
A----E----C
```

Now consider the three edges $AG$, $WC$, and $BE$. Since they cannot intersect the cycle, they must all lie entirely within or without the cycle; so at least two of them must lie on the same side, let us say they both lie inside. But then they have to cross each other, which we assumed was not the case.

Graphs which can be drawn in the plane without edge intersections are called planar; we have just seen that $K_{3,3}$ is not a planar graph, and indeed any graph containing $K_{3,3}$ cannot be planar. To state this more formally, we need the notion of subgraph: $H = (U, F)$ is a subgraph of $G = (V, E)$ if $U \subseteq V$ and $F \subseteq E$. This notion is rather restrictive: for example, $H = (\{1, 2\}, \{12\})$ (a simple edge) is *not* a subgraph of $G_1 = (\{a, b, c\}, \{ab, bc\})$ since these graphs are on different vertex sets and it is not a subgraph of $G_2 = (\{1, 2, 3\}, \{13, 23\})$ either. Hence, we typically use the notion of subgraph in a slightly wider sense to mean: there is a bijection between $H$ and a subgraph of $G$. In that case, $H$, the simple edge in the example above, does occur as a subgraph in $G_1$ (as $ab$, for example) and in $G_2$ (as 13). Two graphs that are the same up to a bijection (a renaming of the vertices) are called *isomorphic*. We typically do not distinguish between two graphs that are isomorphic. (Another useful equivalence relation: isomorphism between graphs.)

So when we speak about a graph containing a $K_{3,3}$, or not, we are talking about a graph containing a subgraph isomorphic to $K_{3,3}$. We can then summarize the discussion by stating that no graph containing a $K_{3,3}$ is planar. More is true: take a $K_{3,3}$ and split one of its edges by introducing a vertex along it; formally, let $uv$ be an edge of $K_{3,3}$. Remove the edge $uv$ and add edges $uw$ and $wv$, where $w$ is a new vertex. This operation is called *subdividing an edge* or *edge subdivision.* The resulting graph is still not planar, since, if we could draw it, we could also draw a $K_{3,3}$: simply remove the vertex $w$ and reintroduce the edge $uv$ in place of $uwv$ to get a planar drawing of $K_{3,3}$, which is impossible.

**Lemma 8.3.1.** *Any subdivision of a non-planar graph is non-planar.*

*Proof.* Let $G$ be a subdivision of $H$, which is not planar. If $G$ is planar, we can draw it without intersections in the plane. Do so, and remove all the subdivision vertices (they have a degree of 2) from the drawing of $G$ (and bridge the gap between the two edges ending in that vertex). This results in a planar drawing of $G$ contradicting the assumption that $H$ was non-planar. ■

If a subdivision of $G$ is isomorphic to a subgraph of $H$, we call $G$ a *topological minor* of $H$. By the lemma we just proved any graph containing $K_{3,3}$ as a topological minor is not planar. $K_{3,3}$ is only one obstruction to planarity; as it turns out there is one more: $K_5$, the *complete graph* on 5 vertices; that is, a graph on five vertices with an edge between any pair of vertices.

**Exercise 8.3.2.** Draw a $K_5$ (with edge intersections, perforce). What is the smallest number of edge intersections you can achieve?

**Exercise 8.3.3.** Show that $K_5$ is not planar, that is, any drawing of a $K_5$ in the plane must contain an edge intersection. *Hint:* Let the vertices be $a, b, c, d, e$. Consider the triangle formed by $a$, $b$ and $c$. How must $d$ and $e$ lie with respect to the triangle? Then argue that either the triangle $dab$ contains $eab$ or vice versa. That will force an edge intersection.

We have seen that $K_{3,3}$ and $K_5$ are non-planar graphs, and, indeed, any graph containing a subdivision of either one is non-planar. In other words, it is a necessary condition of a graph $G$ being planar that $G$ do *not* contain a subdivision of either $K_{3,3}$ or $K_5$. This condition turns out to be sufficient as well, a famous result known as Kuratowski's theorem.

**Theorem 8.3.4.** *A graph is planar if and only if it does not contain a subdivision of $K_{3,3}$ or $K_5$ as a subgraph.*

The crossing number of a graph is the smallest number of intersections in a drawing of the graph (where we do not allow more than two edges to cross at a time, and edges are not allowed to pass through vertices). Determining the crossing number of a graph is a hard problem, unfortunately so, since it plays a central role in graph drawing and visualization.

**Exercise 8.3.5.** Show that the crossing number of $K_{3,3}$ is 1. *Hint:* lower and upper bound.

A special type of drawing often considered in graph drawing are *straight-line drawings* in which every edge is drawn as a straight-line segment.[6] The smallest number of crossings required by a straight-line drawing of a graph is known as the graph's *rectilinear crossing number.* In general, the rectilinear crossing number can be much larger than the crossing number, but for $K_5$ and $K_{3,3}$ it is 1 again.

**Exercise 8.3.6.** Find straight-line drawings of $K_5$ and $K_{3,3}$ realizing a crossing number of 1.

Also, for planar graphs, crossing number and rectilinear crossing number agree:

**Theorem 8.3.7.** *Every planar graph has an intersection-free straight-line drawing in the plane.*

We omit the proof.

**Theorem 8.3.8.** *In the drawing of a planar, connected graph we always have*

$$n - m + f = 2,$$

*where $n$ is the number of vertices, $m$ the number of edges and $f$ is the number of faces.*

*Proof.* Start with the drawing of the given graph. We will show how to modify the drawing in such a way that the value of $f - m + n$ does not change and we obtain a drawing for which we can explicitly verify that $f - m + n = 2$. From this the claim of the theorem follows.

For the purposes of this proof (and this proof only) we will allow multigraphs, that is, multiple edges between two vertices and loops, edges that lead from a vertex to itself; these will naturally occur in the contraction process underlying the proof.

Pick an arbitrary edge $e = uv$ of the graph; contract that edge by moving $u$ along $e$ towards $v$, extending edges incident to $u$. Finally, pull $u$ over $v$, identifying $u$ and $v$ and attaching the edges incident to $u$ to $v$. This contraction operation removed a single vertex and a single edge, so $f - m + n$ does not change (since it contains the term $n - m$). At the end of the contractions we are left with a single vertex (remember that the graph is connected), so $n = 1$. Moreover, all the edges form loops at the single vertex, so the resulting graph is a bouquet of loops. Every loop bounds two different faces, so if we remove a loop, we reduce the number of faces by 1. Again, $f - m + n$ does not change during that operation, because of the term $f - m$. If we remove all loops this

[6] To get a feeling for straight-line drawings, play around with the applet mentioned in the notes at the end.

way, we will be left with a single vertex, $n = 1$ and a single face, $f = 1$ and no edges, $m = 0$. So $f - m + n = 1 - 0 + 1 = 2$. We argue that the value did not change throughout the process, so $f - m + n$ was 2 at the start. ■

One conclusion we can draw from Euler's theorem is that a planar graph cannot contain too many edges: if $n - m + f = 2$, then $m = n + f - 2$. A face in a planar graph (without multiple edges now), must be bounded by at least three edges. Since each edge can bound at most 2 different faces, we know that $f \leq 2/3m$. So $m = n + f - 2 \leq n + 2/3m - 2$, which implies $m/3 \leq n - 2$ or $m \leq 3n - 6$.

**Corollary 8.3.9.** *If $G$ is a planar connected graph with $n \geq 3$, then $m \leq 3n - 6$*

**Exercise 8.3.10.** Give a new proof that $K_5$ is not planar using the corollary to Euler's formula.

**Exercise 8.3.11.** The average degree of a graph $G = (V, E)$ is $\sum_{v \in V} \deg(v)/n$. Show that the average degree of a planar connected graph is less than 6. Conclude that a planar connected graph always contains a vertex of degree at most 5. *Hint:* To compute the average degree use the handshake lemma.

The ancient Greeks suspected that there could be only five regular solids; that is, shapes made up from boundary pieces that are regular polygons such as equilateral triangles, squares, pentagons, hexagons, and so on. For example, we can put six squares together so they form a cube; similarly, four equilateral triangles form a tetrahedron, eight equilateral triangles form an octahedron (try it), twenty triangles make an icosahedron and twelve pentagons a dodecahedron. And this is it, there are no more regular solids; we can show that this is true using Euler's formula.

Before we state the theorem, we need to make our terms precise: we call a solid regular if all of its faces are made of the same type of polygon, that is every face has the same number $b$ of edges, and, if every vertex of the solid is incident to the same number $r$ of faces. We are interested in convex solids, that is, with every two points the convex solid also contains any point on the line segment between the two points (there are no dents in the solid, it bulges outwards).

**Theorem 8.3.12.** *There are only five regular convex solids, corresponding to $(b, r)$ being $(3, 3)$, the* tetrahedron*; $(3, 4)$, the* octahedron*; $(4, 3)$, the* cube*; $(3, 5)$, the* icosahedron *and $(5, 3)$, the* dodecahedron*.*

**Exercise 8.3.13.** The fact that these five regular solids exist can be demonstrated by building them out of paper; do so.

*Proof.* Take a regular convex solid; it is made up of polygons each of which has $b$ edges, and there are $r$ faces incident to each vertex. We have to show that $(3, 3), (3, 4), (4, 3), (3, 5), (5, 3)$ are the only possible values for $(b, r)$.

Imagine the solid is made out of wire, so only the edges and vertices are visible. If we look at the wires from a point very close to the center of one of the polygons, we can see all the wires, and, furthermore, they will not obstruct each other (all this is true because the solid is convex). In other words, the edges of the solid are really a planar graph; another way of seeing this is by picking one of the polygons and stretching it and the remaining faces with it, until they come to rest in the plane, without intersections. (Try doing this for some, not necessarily regular, convex solids.)

By Euler's formula, we then know that $f - m + n = 2$, where $f$ is the number of faces of the solid, $m$ the number of edges, and $n$ the number of vertices. Since we have $f$ faces, and every face is made up of $b$ edges, we count $fb$ edges; however, as in the handshake lemma, we double-counted each edge, since every edge belongs to two faces, so $2m = fb$. Since there are $r$ faces that meet at each vertex, there are also $r$ edges that meet at each vertex; so, again, we double-count the edges as $nr$ and conclude that $nr = 2m$. Substituting this in

$$f - m + n = 2,$$

gives us

$$2m/b - m + 2m/r = 2.$$

Now, $b \geq 3$, since a polygon must have at least three edges, and $r \geq 3$, since otherwise we would have a vertex adjacent to at most two edges, which would mean the solid is flat. If both $b > 3$ and $r > 3$, then

$$2m/b - m + 2m/r = m(2/b + 2/r - 1) < m(1/2 + 1/2 - 1) = 0,$$

which is not possible, so either $b = 3$ or $r = 3$. If, for example, $b = 3$, we obtain

$$2m/3 - m + 2m/r = 2,$$

or, solving for $r$,

$$r = 6m/(6 + m).$$

The only possible values for $r$ are 3, 4 and 5, giving us $m = 6$, $m = 12$ and $m = 20$.

If, on the other hand, we assume that $r = 3$, we use the same argument that $b$ can have only values 3, 4 and 5. This proves the theorem. ■

## 8.4 *Ramsey Theory

It is a dark winter morning and the lights in your clothes cabinet aren't working. So you can't see the color of the socks you are picking. Your socks all look the same, but they come in five different colors that you can't distinguish in the dim light. How many socks do you need to grab to make sure that you have two socks of the same color?

The smallest answer is six. Five won't do, since you could have one sock of each kind. Why is six sufficient? Imagine sorting the socks you grabbed into five heaps depending on their color. Since there are six socks and only five heaps, one heap must contain at least two socks.

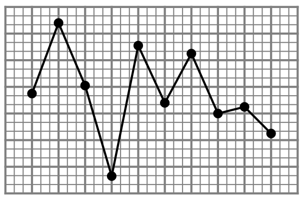
This principle is known as the *pigeonhole principle* (or *Dirichlet's principle*): if you place more than $n$ pigeons into $n$ pigeonholes, then one of the pigeonholes must contain at least two pigeons.

The pigeonhole principle is an example of a type of combinatorial phenomena that are collectively known as *Ramsey theory*, named after the Cambridge philosopher and mathematician F.R. Ramsey. Results from Ramsey theory show that if you take a large enough, but otherwise arbitrary collection of any type of object, there will be a part of that collection that is structured. In other words, we cannot avoid structure: if there are only five colors of socks then any collection of more than five socks will contain two socks of the *same* color. That is, we couldn't avoid structure.

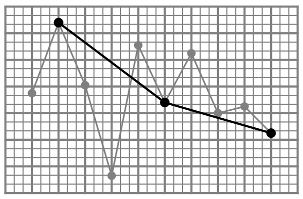
The following table contains the stock information for a famous internet company in the year 2006:

| 9/1 | 9/5 | 9/6 | 9/7 | 9/8 | 9/11 | 9/12 | 9/13 | 9/14 | 9/15 |
|---|---|---|---|---|---|---|---|---|---|
| 28.15 | 28.68 | 28.21 | 27.53 | 28.51 | 28.08 | 28.45 | 28.00 | 28.05 | 27.85 |

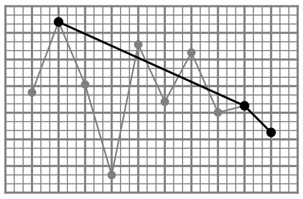
The same data displayed in a chart:



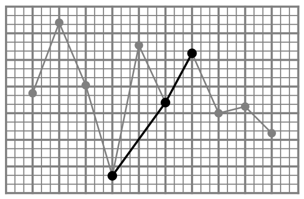
The look at the graph makes the data look nearly random; but imagine what you can do with this. If you want to hurt the company, you could point out how the stock has been decreasing: just pick the data on 9/5, 9/11, 9/15: 28.68, 28.08, 27.85;



or, if you like it more dramatic: 9/5, 9/6, 9/7: 28.68, 28.21, 27.53. Or, more subtle and devious, you can suggest a slow, but continuing decline: 9/5, 9/8, 9/12, 9/14, 9/15: 28.68, 28.51, 28.45, 28.05, 27.85:



On the other hand you can as easily make the company look good: consider the dramatic increase 9/7, 9/11, 9/12: $27.53, 28.08, 28.45$.



An illustration, how you can make your data prove anything you want, While this illustrates the importance of good judgement in tabulating, visualizing, and interpreting data (and how hard it is to keep these apart), there also is a Ramsey-theoretic observation lurking in the wings. With sufficient data it is impossible to avoid local structure: There will always have to be long increasing or decreasing subsequences in our data, whatever it looks like.

Let us make the terminology precise (we will talk more about sequences in the next chapter). A sequence $(a_i)_{i \in I}$ is a series of values, indexed by parameters in the index set $I \subseteq \mathbb{N}$. The number of indices, $|I|$, is the *length* of the sequence. For example, we could have written the stock market example above as a sequence of length ten: $(a_i)_{i\in\{1,\ldots,10\}}$ or $a_1, \ldots, a_{10}$, where $a_1 = 28.15, a_2 = 28.68, \ldots, a_10 = 27.85$.

As local structure we consider subsequences of the original sequence, as we did with the stock market values. Formally, a subsequence of a sequence results by selecting some of the indices: $J \subseteq I$ for inclusion in the subsequence. E.g. with $J = \{4, 6, 7\}$ we get $a_4, a_6, a_7 = 27.53, 28.08, 28.45$ an *increasing* subsequence. A sequence $(a_i)_{i\in I}$ is *increasing* if $a_i \leq a_j$ for every $i \leq j$, $i, j \in I$; it is *decreasing* if $a_i \geq a_j$ for every $i \leq j$, $i, j \in I$. A sequence is *monotone* if it is either decreasing or increasing.

Our sequence of ten stock values contained many monotone subsequences as we saw. In general, we cannot claim that there will always be long increasing and decreasing subsequences in an arbitrary sequence; for example,

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$$

contains only increasing subsequences (we ignore subsequences of length 1, which are, by definition, both increasing and decreasing, but not very interesting).

**Exercise 8.4.1.** Build a sequence of length 6 that only contains decreasing subsequences (of length 2 and more).

**Theorem 8.4.2** (Erdős-Szekeres). *If we have a sequence of length at least $n^2+1$, then the sequence contains a monotone subsequence of length at least $n+1$.*

**Example 8.4.3.** The theorem tells us that in a sequence of length 10 there must be a monotone subsequence of length at least 4. In the stock market example, we even saw a decreasing sequence of length 5. On the other hand, the stock market example did not contain an increasing sequence of length 4.

**Exercise 8.4.4.** Find longest monotone subsequences in the following sequences:

1. $9, 1, 7, 5, 4, 2, 8, 3, 10, 6$.
2. $2, 1, 4, 3, 6, 5, 8, 7, 10, 9$.

**Exercise 8.4.5.** In this exercise we will see that the bounds given in the Erdős-Szekeres theorem as sharp, i.e. they cannot be improved:

1. Construct a sequence of length 4 that contains no monotone subsequence of length 3.
2. Construct a sequence of length 9 that contains no monotone subsequence of length 4.
3. Show how to construct, for an arbitrary $n$, a sequence of length $n^2$ that does not contain a monotone subsequence of length $n + 1$.

We still owe the proof of the theorem; it turns out to be surprisingly simple. Look at a sequence, let's say $a_1, \ldots, a_{10} = 7, 5, 8, 2, 6, 1, 10, 9, 4, 3$. For each index $i$ in the sequence, we tabulate the length of the longest increasing and the longest decreasing subsequence *ending* at that index:

| sequence | 7 | 5 | 8 | 2 | 6 | 1 | 10 | 9 | 4 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|
| increasing | 1 | 1 | 2 | 1 | 2 | 1 | 3 | 3 | 2 | 2 |
| decreasing | 1 | 2 | 1 | 3 | 2 | 4 | 1 | 2 | 3 | 4 |

Let us verify some of the values: initially, $a_1 = 7$ and the longest increasing and decreasing subsequence is 7, for a length of 1. For $a_2 = 5$ we have a longest decreasing subsequence ending at $a_2$ of length 2, namely $7, 5$. The longest increasing subsequence ending at $a_2$ is 5, for a length of 1. For $a_3$, the longest increasing subsequence ending at $a_3$ is of length 2 (actually, there are two options $7, 8$ and $5, 8$). Since 8 is the largest number we have seen so far, the longest decreasing subsequence ending in $a_3$ is just 8 itself, for a length of 1. And so on.

Note that there is something interesting about the two sequences we tabulated below the values of $a_i$; if we consider them as pairs of values, no pair ever repeats: $(1, 1), (1, 2), (2, 1), (1, 3)$ and so on. There is a reason for this; suppose you have a pair $(p, q)$ as position $i$, and let us also look at the pair $(r, s)$ at position $j$ (for $j > i$). Take, for example, $i = 5$ and $j = 8$, then $(p.q) = (2, 2)$ and $(r, s) = (3, 2)$. There are two possibilities: either $a_i < a_j$, then $a_j$ can be used to extend the longest increasing subsequence ending at $i$ to get an increasing sequence of length $p + 1$ ending at $j$; hence, $r \geq p + 1$ in this case. Otherwise, $a_i \geq a_j$, then $a_j$ can be used to extend the longest decreasing subsequence ending at $i$ to get a decreasing sequence of length $q + 1$ ending at $j$; hence $s \geq q + 1$.

So, either $r \geq p+1$ or $s \geq q+1$. In either case, $(p,q) \neq (r,s)$. For $i=5$ and $j=8$, we see that $6 = a_5 < a_8 = 9$, so $r \geq p+1$ in this case, and, indeed, they are equal: $r=3$ and $p=2$.

We conclude that every pair occurs at most once; however, if all monotone sequences have length at most 3, then there are at most $3^2 = 9$ possible pairs. However, the sequence has length 10. This is impossible, so the value 4 has to occur somewhere (indeed, it does twice in the example.

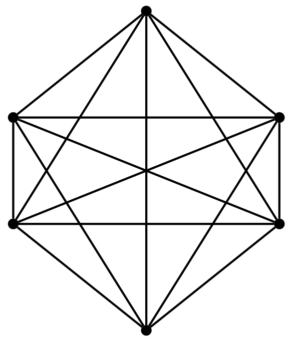
The same proof works in general: if you have a sequence of length at least $n^2+1$ but all monotone sequences have length at most $n$, then there are at most $n^2$ pairs; however, since all pairs are pairwise different, this cannot actually happen, and there must be a monotone subsequence of length at least $n+1$.

**Exercise 8.4.6.** As we did in the proof of the Erdös-Szekeres theorem, compute the table of longest increasing and decreasing subsequences ending at each position for our original data set:

| 9/1 | 9/5 | 9/6 | 9/7 | 9/8 | 9/11 | 9/12 | 9/13 | 9/14 | 9/15 |
|---|---|---|---|---|---|---|---|---|---|
| 28.15 | 28.68 | 28.21 | 27.53 | 28.51 | 28.08 | 28.45 | 28.00 | 28.05 | 27.85 |

## Ramsey's Theorem

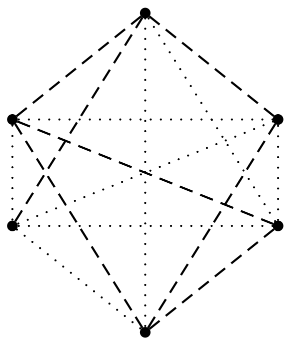
In a group of six people there will always be three people who know each other or three people who do not know each other; in other words, any two of the three know each other or any two of them don't know each other. We can phrase this as a graph question by using colored edges; the six people are represented by six vertices. We also add all possible edges to the graph. This graph is known as $K_6$, the complete graph on 6 vertices:



In general, the *complete graph*, $K_n$ on $n$ vertices is the (undirected) graph consisting of $n$ vertices and all possible edges.

**Exercise 8.4.7.** How many edges does $K_6$ have? How many edges does $K_n$ have?

We color an edge between two vertices red if the two people represented by the vertices know each other; otherwise we color the edge green (in the picture we represent red as a dotted edge and green as a dashed edge):



Our claim that there are always three people who know each other or three people who do not know each other translates into the presence of a *monochromatic triangle*, that is a $K_3$ all of whose edges have the same color. We can easily verify the presence of a monochromatic triangle in the above coloring of $K_6$; the claim is that there always is such a triangle.[7]

**Exercise 8.4.8.** Show that the claim is not true for a $K_5$, that is, find a coloring of a $K_5$ that does not contain a monochromatic triangle.

**Exercise 8.4.9.** Maybe the claim looks trivial to you; does it easily generalize? How about: every coloring of a $K_8$ contains a monochromatic $K_4$. Is that true? *Hint:* No. Find a coloring that proves this generalization is not true.

The proof of the claim is not very hard, it is two step sequence of pigeonhole arguments. Take a $K_6$ and fix any coloring of the $K_6$. We have to find a monochromatic triangle. Pick an arbitrary vertex of the $K_6$. That vertex is incident to 5 edges. Since there are only two colors, at least 3 of the edges must have the same color. Let us say that color is red (if it's green, the same argument will work with colors exchanged). Look at the three vertices the red edges connect us to. If there is a red edge between any two of them, we have completed a red triangle. Otherwise all the edges between those three vertices are green, and we have found a green triangle.

The result is not about people knowing or not knowing each other, of course; it is about a binary relation $R$: if your universe contains at least 6 objects, then, for any binary relation, there are either three objects such that $R$ is always true for any two of them, or always false.

Compare this result to the Erdös-Szekeres theorem: that result is a for a special type of relation: an ordering relation, and it gives us a stronger conclusion: a sequence of $n$ elements contains a monotone subsequence of length $\sqrt{n-1}$. For arbitrary relations, the bound is quite a bit worse:

**Theorem 8.4.10** (Ramsey)**.** *Every coloring of a $K_n$ with two colors contains a monochromatic $K_{\log n/2}$.*

[7]For all colorings of $K_6$ there is a triangle in $K_6$ such that all edges in the triangle have the same color. The logical structure here is $\forall\exists\forall$.

We will prove Ramsey's theorem in the following form: every coloring of a $K_{2^n}$ contains a monochromatic $K_{\lceil n/2 \rceil}$. The proof is similar to the one we saw earlier. Fix a coloring of $K_{2^n}$ and let the vertices be named $v_1, \ldots, v_{2^n}$. Consider $v_1$. At least $\lceil (2^n - 1)/2 \rceil = 2^{n-1}$ of the edges leaving $v_1$ have the same color, let us say $c_1$. Remove all but the $2^{n-1}$ vertices that are connected to $v_1$ by an edge of color $c_1$. Now consider $v_2$. Among its $2^{n-1} - 1$ neighbors there must be $\lceil (2^{n-1} - 1)/2 \rceil = 2^{n-2}$ that are reached by edges of the same color, $c_2$. Again, restrict the graph to those $2^{n-2}$ vertices. We can continue picking vertices like this, until $2^{n-k} = 1$, that is $k = n$. So we have a sequence of vertices $v_1, \ldots, v_n$ such that all edges $v_i v_j$ (with $j > i$) have color $c_i$. Now there are only two colors, so among the $n$ values of $c_i$ at least $\lceil n/2 \rceil$ must be the same. If we choose the corresponding vertices, we have found $\lceil n/2 \rceil$ vertices such that all edges between them have the same color.

**Remark 8.4.11.** The argument just given is essentially Ramsey's own; this beautiful proof still works if the set of vertices is infinite.

**Exercise 8.4.12.** Ramsey's theorem is stated in terms of graphs; we can do the same for the Erdös-Szekeres theorem: restate that theorem as a theorem about directed graphs. *Hint:* A *tournament* graph is a directed graph $G = (V, E)$ in which for every $u, v \in V$ exactly one of $uv$ or $vu$ is in $E$ (for $u \neq v$), and if $uv \in E$ and $vw \in E$, then $uw \in E$.

## 8.5 Exercises

1. Check out the picture of the hedge maze at the Morton Arboretum (`http://www.mortonarb.org/maze/images/imgMap.gif`) and draw the corresponding to the maze; there should be one vertex for each fork, dead end, entrance and for the goal (rather than exit), the sycamore tree.

2. There is a famous game called "The Kevin Bacon Game". Before you read the exercise, familiarize yourself with the rules of the game at

   `http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon,`

   and try playing the game yourself at "The Oracle of Bacon at Virginia", see

   `http://oracleofbacon.org/.`

   (*a*) Describe how you would model this game as a graph. Hint: you need to determine what are the vertices and what are the edges of your graph. How would you model the actors in terms of graphs?

   (*b*) In your graph model what does it mean for somebody (like Paul Robeson) to have a Bacon number of 3?

   (*c*) In your graph model what does it mane for somebody to have a Bacon number of $k$? Rephrase in graph-theoretical terms.

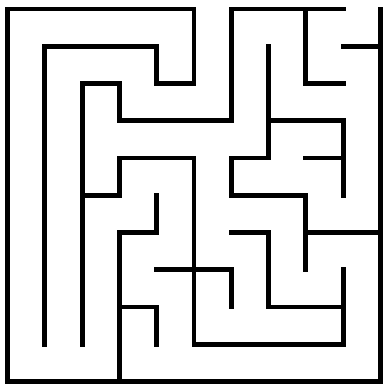


Figure 8.1: A maze.

3. Consider the maze in Figure 8.1
   - (*a*) Draw the graph corresponding to the maze; there should be one vertex for each fork, dead end, entrance and for the goal.
   - (*b*) Perform a depth-first search on the maze. (Show the details of how `reached` changes.)
   - (*c*) Perform a breadth-first search on the maze. (Show the details of how `reached` changes.)
   - (*d*) Does the maze contain cycles? Is there more than one solution?
4. The maze in Figure 8.2 is taken from Sam Loyd's Cyclopedia of puzzles (published in 1914 as prepared for publication by Loyd's son, who was also known as Sam Loyd).

   The accompanying text reads (with spelling mistakes left intact):

   > Any or every style of puzzle which excites interest or affords amusement is beneficial, in that it trains the mind to concentrate and pursue a line of thought to a definite purpose. Maze puzzles are always interesting to both young and old on account of the historical associations which connects them with noted mazes in ancient parks and gardens, as well as from the inate pleasure we all feel in over coming seeming obstacles. Of course there are many styles of labarynths with various conditions which make them more or less difficult, but the above may be said to be one of the best because the crossing of paths by means of bridges permits of a much wider range of travel than the old fashioned limitation to branch walks. This puzzle is by Lewis Caroll, who

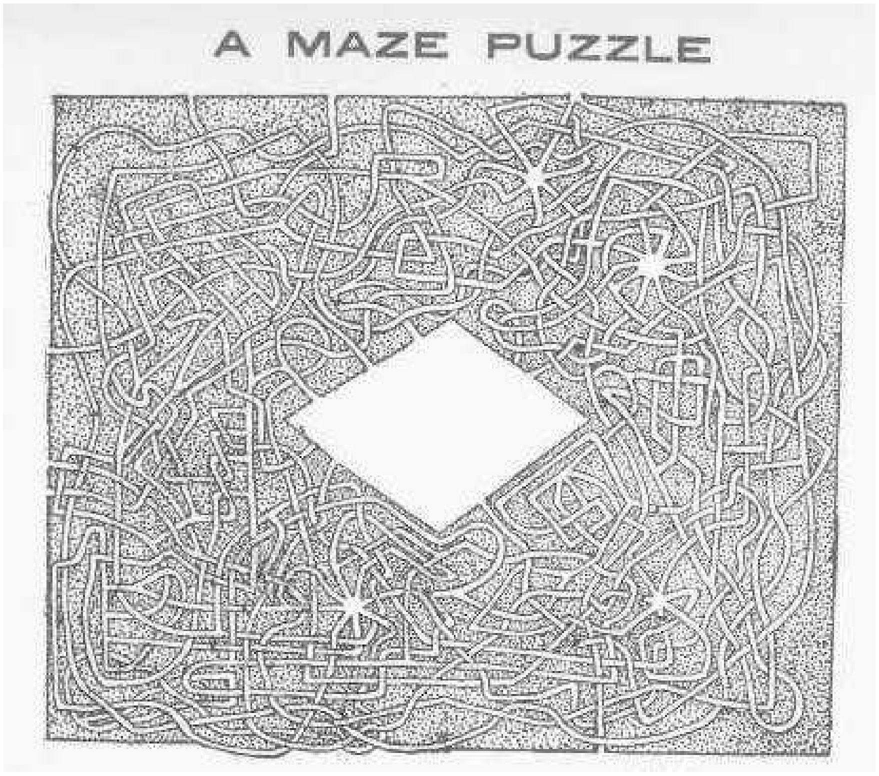


Figure 8.2: A maze by Lewis Carroll.

as you all remember, wrote Alice in Wonderland, was a great mathematician and a noted puzzlist. It is sup posed to represent poor little Alice lost in the woods ; she starts from the little park in the centre and wishes to get out of the woods to go home. Can you give her any assistance in finding the correct path? You will notice that some of the paths are obstructed so as to make you retrace your steps, but not to be discouraged just remember that Euler formulated a rule for solving all mazes. Nevertheless it is quite a clever and difficult puzzle.

**Note:** There are several little obstructions in the maze that may not be crossed.

5. Suppose we are given the drawing of a graph representing a maze; in particular, we have special vertices $s$ (entrance) and $t$ (exit), and vertices for each fork and dead end in the maze; two vertices are connected by an edge if there is a corridor directly leading from one location to the other (without passing through any other location that is represented as

a vertex). We also assume that the order of the edges leaving a vertex represents the order in which the corresponding corridors leave the vertex location.

Show that you can find a way from $s$ to $t$ using the right-hand rule if and only if the graph does not contain a cycle $C$ which separates $s$ and $t$, in the sense that exactly one of $s$ and $t$ lies within the region enclosed by the cycle, and the other one lies outside.

6. We present another way of implementing topological sort; recall that for a directed graph the out-degree of a vertex is the number of edges leaving the vertex (as opposed to entering the vertex). Given a directed graph $G$ without any directed cycles proceed as follows: repeatedly (and as long as you can) pick a vertex of out-degree 0, remove it and all its outgoing edges from $G$. (Do this on the course prerequisite example.) The vertices are a topological sort in the order you remove them.

    ($a$) Why is there always a vertex of out-degree 0? *Hint*: what can you say about a graph in which every vertex has out-degree at least 1 (keep following that edge)?

    ($b$) Argue that the algorithm described in this exercise produces a topological sort.

7. Here is a simplified version of the prerequisite structure of the graphics degree. GPH 213 and GPH 212 require GPH 211, GPH 338 requires GPH 213 and GPH 325. GPH 325 requires CSC 212, which requires CSC 211, which requires IT 130. GPH 339 requires GPH 325 and CSC 212. CSC 321 requires CSC 393 and MAT 140. CSC 393 requires CSC 212. MAT 220 requires MAT 141, which requires MAT 140. GPH 329 requires CSC 393 and MAT 220. GPH 375 quires GPH 329. GPH 372 requires GPH 329. GPH 395 requires GPH 338 and GPH 372.

    ($i$) Draw the dependency graph for the classes in the graphics degree.

    ($ii$) Run topological sort on the graph.

    ($iii$) Find a longest path in the graph. What does it mean?

    ($iv$) Redraw your dependency graph so it is easy to follow.

8. Which of the five regular solids have Hamiltonian cycles? Include a drawing with the Hamiltonian cycle if it exists.

    ($i$) Tetrahedron

    ($ii$) Octahedron

    ($iii$) Cube

    ($iv$) Icosahedron

    ($v$) Dodecahedron

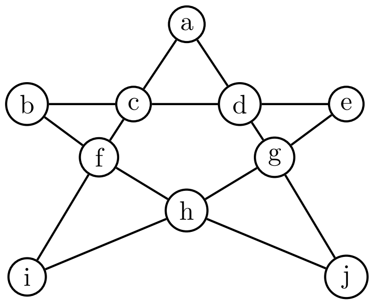
9. You are given a $3 \times 3 \times 3$ cube consisting of 27 smaller cubes (from outside it would look rather like a Rubik's cube).[8] By making six straight cuts you can get all 27 of the smaller cubes. Is there a solution that uses less than six straight cuts? (Either find a better solution, or argue that it cannot be done.) In a single cut you are allowed to cut multiple pieces at the same time (by piling them on top of each other), but all the cuts you make have to be straight.

10. Another classic puzzle (Loyd includes it in his Cyclopedia under "After dinner tricks" on page 41. On your table there are 8 wineglasses in a row, four empty glasses followed by four full glasses: $\bigcup\bigcup\bigcup\bigcup\biguplus\biguplus\biguplus\biguplus$. In Loyd's own words:

    > Proposition: Pick up two adjacent glasses at a time and in four moves change the positions so that each alternate glass will be empty.

11. A small variation of the previous problem: you have 10 wineglasses, five full, five empty arranged in a circle on your table. The full and empty glasses alternate along the circle. A single move consists of swapping two of the glasses. How many moves do you need at most to bring all the full glasses together into a single group?

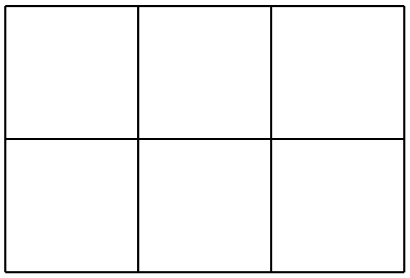
12. Look at the following playing field:

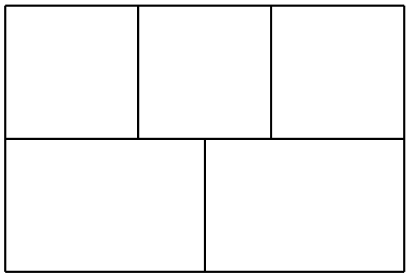
    

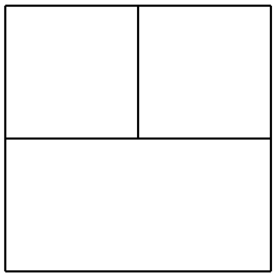
    Place a coin on an arbitrary node and move it to an adjacent node. Overall, you want to place nine coins like this, without ever placing a coin on a node already occupied and without moving a coin onto a field that is already occupied. Spell out a sequence of moves that achieves this.

13. For each of the following figures try to draw a single curve which crosses each line of the drawing exactly once. Your curve should not intersect itself. There might not always be a solution; in that case, give an argument that there cannot be a solution. *Hint:* think of modeling the problem using multigraphs.

[8]If you don't know what a Rubik's cube is, go buy one.


(*i*)


(*ii*)


(*iii*)

14. [H.E. Dudeney] The following puzzle is by Dudeney entitled "A Lodging-House Difficulty"; his original description reads as follows:

> The Dobsons secured apartments at Slocomb-on-Sea. There were six rooms on the same floor, all communicating, as shown in the diagram. The rooms they took were numbers 4, 5, and 6, all facing the sea. But a little difficulty arose. Mr. Dobson insisted that the piano and the bookcase should change rooms. This was wily, for the Dobsons were not musical, but they wanted to prevent any one else playing the instrument. Now, the rooms were very small and the pieces of furniture indicated were very big, so that no two of these articles could be got into any room at the same time. How was the exchange to be made with the least possible labour? Suppose, for example, you first move the wardrobe into No. 2; then you can move the bookcase to No. 5 and the piano to No. 6, and so on. It is a fascinating puzzle, but the landlady had reasons for not appreciating it. Try to solve

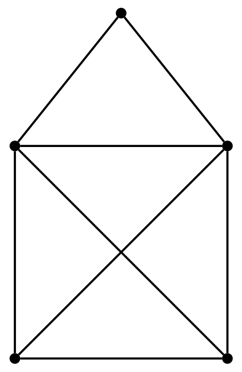
her difficulty in the fewest possible removals with counters on a sheet of paper.

```
------------------------------
| Cabinet|         |  Piano  |
|                            |
|1       |2        |3        |
---    -----     -----     ---
|        |         |         |
|Drawers  Wardrobe   Bookcase|
| 4      |5        |6        |
------------------------------
```

Write down your solution as a sequence of numbers, where a number represents the room that you are moving a piece of furniture *from*. (Since there is only one empty room, the room you are moving it *to* is always determined.) *Hint:* The solution will change the order of Cabinet, Drawers and Wardrobe. Unavoidably so, as we will see later.

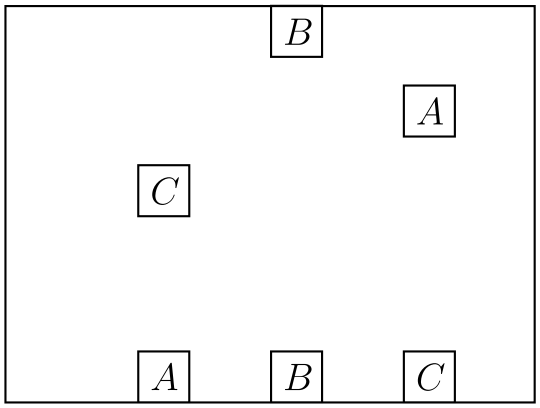
15. [Martin Gardner] We start with a $3 \times 3$ chessboard. Place a knight on the chessboard and make one move with it. You want to repeat this six more times. Where do you have to place the knights in what order to make this possible? How did you arrive at the solution?

16. An old children's game; draw the picture

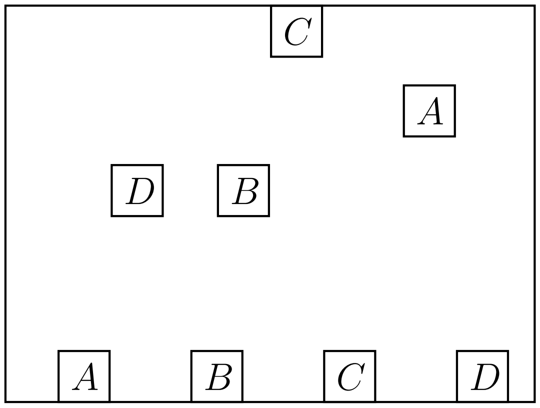


without lifting your pen. That is, start at some point, and then draw the picture, finishing it before lifting up the pen again. Can you do that and end at the same point where you started?

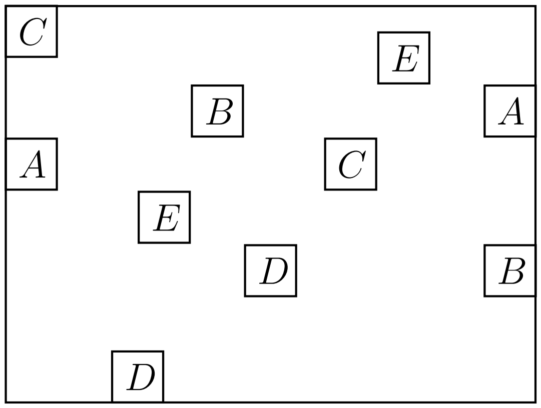
17. Connect $A$ to $A$, $B$ to $B$ and $C$ to $C$ (without leaving the field).



18. Connect $A$ to $A$, $B$ to $B$, $C$ to $C$ and $D$ to $D$ (without leaving the field).



19. Connect $A$ to $A$, $B$ to $B$, $C$ to $C$, $D$ to $D$ and $E$ to $E$ (without leaving the field).



20. [Interview Question] You have a bucket of jelly beans. Some are red, some are blue, and some green. With your eyes closed, you pick jelly beans from the bucket. How many do you have to grab to be sure you have 2 of the same color?

21. Does a longest monotone subsequence of a sequence have to contain the first or the last element of the sequence? Either prove that it does, or show a counterexample.

22. In the proof of the Erdős-Szekeres theorem we used pairs of numbers $(p_i, q_i)_{i \in I}$ where $p_i$ was the length of the longest increasing subsequence ending at position $i$ and $q_i$ was the length of the longest decreasing subsequence ending at position $i$. Write an algorithm in pseudocode that computes these numbers quickly, by just looking at all the previous value. *Hint:* if you were filling the table by hand, how would you proceed systematically (that is, without explicitly searching for subsequences).

23. The full Erdős-Szekeres theorem is a bit stronger than what we stated above:

    **Theorem 8.5.1** (Erdős-Szekeres). *If we have a sequence of length at least $rs + 1$, then the sequence contains either an increasing subsequence of length $r + 1$ or a decreasing subsequence of length $s + 1$.*

    ($i$) Show an application of the full Erdős-Szekeres that doesn't follow from the version presented earlier.

    ($ii$) Prove the full version of the theorem. *Hint:* The original proof will only need minor adjustments.

24. Imagine a $K_6$ and two players alternately coloring the edges red (Player I) and green (Player II). Player I goes first. Players will win or lose depending on whether there is a monochromatic triangle in their color. By Ramsey's

theorem we know that there will always be a monochromatic triangle, so there cannot be a tie in this game.

(*i*) A player wins if he is the first to complete a monochromatic triangle in his color. Does Player I have a winning strategy, that is, can he always win?

(*ii*) A player wins if the *other* player completes a monochromatic triangle in their color. Play this version of the game online at `http://www.dbai.tuwien.ac.at/proj/ramsey/index.html`.

25. A farmer claims he has four trees that are pairwise equidistant; is he lying?

## 8.6 Notes and Additional Reading

There are many good books and web-pages on mazes; an old classic is William Henry Matthews' *Mazes and Labyrinths: Their History and Development*, which is available as a Dover reprint. There are several web-pages that allow you to generate your own mazes, for example, Maze Maker at

`http://hereandabove.com/maze/`.

Euler's formula is a ubiquitous tool; the web-page

`http://www.ics.uci.edu/%7Eeppstein/junkyard/euler/`

collects nineteen different proofs of it.

Sam Loyd was one of the most imaginative American puzzle inventors; there is a famous collection of his puzzles edited posthumously by his son under the title *Sam Loyd's Cyclopedia of Puzzles*. The MAA uploaded the book in its entirety at

`http://www.maa.org/editorial/mathgames/mathgames\_01\_03\_05.html`.

E.H. Dudeney was the British counterpart to Sam Loyd; his *Amusements in Mathematics* are still worth working through; the book is online at

`http://www.web-books.com/Classics/Nonfiction/Science/AmuseMath/Contents.htm`.

For a nice discussion of the utilities problem, see

`http://www.cut-the-knot.org/do_you_know/3Utilities.shtml#wells`.

There are many graph-drawing tools, professional, as well as for fun. If you want to get familiar with straight-line drawings, check out the applet at

`http://mathdl.org/images/upload\_library/3/EnsleyPlanar/planarApplets.html`,

there are exercises there as well.

The Erdös-Szekeres theorem might not be as ubiquitous a tool as Euler's formula, but it is very well-known and there are many proofs of it; there is a fine survey article by J. Michael Steele called *Variations on the Monotone Subsequence Problem of Erdös and Szekeres*, available online at

`http://www-stat.wharton.upenn.edu/~steele/Publications/PDF/VOTMSTOEAS.pdf`.