Let’s begin with a familiar scenario: suppose that we need to schedule the matches of a round-robin tournament involving \( n \) teams, so that once the tournament has concluded each team will have played against each other team exactly once. In total there would be \( \binom{n}{2} \) games. Potentially they could be scheduled sequentially, but that could make for a very long and drawn out tournament. In the interest of completing the event as quickly as possible, we instead want to have several teams competing simultaneously. The question that now arises is this: given that no team can play more than one game at a time, how few time slots are needed in order to schedule the whole tournament?

This particular question was answered long ago by modelling it with graph theory. In the 1890s Édouard Lucas [5] published a solution, for which he gave credit as follows:

\begin{quote}
Parmi les diverses méthodes qui nous ont été indiquées, nous exposeons, de préférence, les solutions simples et ingénieuses de M. Walecki, professeur de Mathématiques spéciales au lycée Condorcet.
\end{quote}

At this stage it would be good to know exactly what a graph is. Formally, a graph \( G \) consists of a set \( V \) of elements called vertices, accompanied by a set \( E \) of edges, which themselves are pairs of vertices. Any graph can easily be represented in the form of a drawing. For instance, in Figure 1 are two drawings of the graph having \( V = \{a, b, c, d, e\} \) and \( E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, d\}, \{c, d\}, \{d, e\}\} \). Each vertex is depicted as a circular node, and each edge is illustrated by drawing a line between its two vertices.

![Figure 1: Two different drawings of a graph](image-url)

Note that there is no specified location for the vertices or edges of a graph. Moreover, the edges do not need to be drawn as straight lines, but are free to have bends and curves. In practice the vertices may represent real entities (such as sports teams which do have a geographical placement) and the edges might also represent connections with physical form (such as railway lines between cities), but what is most important here is that the graph captures the existence of a relationship between pairs of vertices (such as the need for their corresponding...
teams to play against each other).

For our scenario of a round-robin tournament, we will want to consider a graph with \( n \) vertices such that each pair of vertices is joined by an edge. Such a graph is called a complete graph and is denoted by \( K_n \). Figure 2 illustrates the graph \( K_7 \), with the vertices named 0 to 6. As can be seen in this example, we also allow edges to be drawn so that their lines intersect.

![Figure 2: The complete graph \( K_7 \)](image)

If we have seven teams that must each play against each other during a tournament, then each edge of \( K_7 \) corresponds to an individual game that has to be played. But we still need to find a way to schedule the games, preferably into as few time slots as possible. Since each edge of \( K_7 \) represents a distinct game that must be played, then the games within a single time slot correspond to a set of edges, no two of which share a vertex. So to find a schedule, we need only find a way to partition the edges of \( K_7 \) into sets of this form.

**Exercise 1** Find a schedule for these 21 games.

Having just found a schedule for the games that the seven teams must play, we now have to consider whether there might be a better schedule.

**Question 1** Is there a schedule that uses fewer time slots?

If not, then how can you be sure that yours is indeed the best? To determine just how few time slots there are in an optimal schedule, we need to do some mathematical thinking.

Observe that each of the seven teams has to play six games, so right away we know that the number of time slots that are in any valid schedule has to be at least six. Were you able to find a schedule that only used six time slots?

**Question 2** Can you find a schedule with six time slots? If not, then can you prove that there is no schedule with only six time slots?

_Crux Mathematicorum_, Vol. 41(2), February 2015
As it happens, there is no way to schedule the tournament with seven teams so that all of the games fit into only six time slots. To convince yourself that this is the case, note that the teams that are playing games within a single time slot are playing in pairs. So with seven teams, at most six of them can actually be playing at any given time, which in turn means that at most three games can take place at a time. In total, there are 21 games that must be played, and with at most three that can be scheduled per time slot, we need at least seven time slots for the tournament. With this argument in mind, we can now conclude that any schedule that uses only seven time slots must in fact be an optimal solution.

Let’s move away from the example of $n = 7$ now and consider what might happen when $n$ is even. Is it still the case that $n - 1$ time slots is impossible?

**Exercise 2** Use $K_4$ and $K_6$ to find optimal schedules for $n = 4$ and $n = 6$.

You should find that for these two small examples it is actually possible to find schedules with as few as 3 and 5 time slots, respectively. To try to see a general pattern we will now consider $n = 8$. With teams named 0 to 6 and $\infty$, Figure 3 illustrates how to form four pairs of teams for the first time slot of the tournament, and then how to form four pairs for the second time slot. Looking at this figure, a general approach ought to become apparent: rotate the edges clockwise for each subsequent time slot. To be a bit more technical, for each edge $\{u, v\}$ of one time slot, for the next time slot use the edge $\{u + 1, v + 1\}$ where we treat $\infty + 1$ as $\infty$ and $(n - 2) + 1$ as 0.

![Figure 3: Team pairings for two different time slots in $K_8$](image)

So for even $n$, an optimal solution is to use this technique with the edges $\{0, \infty\}$ and $\{1, n - 2\}, \{2, n - 3\}, \ldots, \{\frac{n-2}{2}, \frac{n}{2}\}$ for the initial time slot. The resulting schedule will have a total of $(n - 1)$ time slots for the tournament.

For odd $n$, however, recall that each time slot has to have a team that sits out. We can find a schedule that uses $n$ time slots by introducing a phantom team called $\infty$, building an optimal schedule for $(n + 1)$ teams (note that $n + 1$ is even), and then assigning byes to teams whenever they are paired with the phantom team.
So when \( n \) is odd, we know that a schedule with \( n \) time slots can be achieved, although for \( n \neq 7 \) we have not yet proved that \( n - 1 \) time slots are insufficient.

At this point hopefully you are beginning to wonder what any of this has to do with colouring, although perhaps you’ve already discovered that each time slot can be associated with a distinct colour. If we colour the edges of a graph so that two edges that meet at a common vertex are not allowed to share the same colour, then it is possible to use the colouring to form a schedule of pairings. Alternatively, if we do not actually have colours to work with (such as this black-and-white article), we can emulate the idea of colours with dashed lines, etc., similar to what we have done for the graph shown in Figure 4. The solid black edges \{a, c\} and \{b, d\} could be used to indicate two games to be played during the first day of a competition (with team \( e \) having a bye), the short-dashed edge \{c, d\} provides for just one game on the second day, the dotted edges \{a, b\} and \{d, e\} tell us which games are to take place on the third day, and finally the long-dashed edge \{a, d\} corresponds to the sole game on the fourth day of the competition. Note that in this example the competition is not a round-robin tournament (since not every pair of teams will play against each other).

![Figure 4: Example of an edge colouring](image)

An edge colouring for which edges of the same colour never meet at a common vertex is called a *proper* edge colouring. An easy way to obtain a proper edge colouring is to give each edge a distinct colour, but this would result in no games taking place at the same time. As before, our goal is to determine how few time slots are needed. With our new terminology, given a graph \( G \) we want to know the smallest number of colours for which a proper edge colouring exists; this value is called the *chromatic index* of the graph and is denoted by \( \chi'(G) \). An obvious lower bound on the chromatic index is that \( \chi'(G) \geq \Delta(G) \), where \( \Delta(G) \) denotes the number of edges at any vertex with the most edges (for the graph in Figure 4, \( \Delta(G) \) is 4 thanks to vertex \( d \) belonging to four edges).

For complete graphs, we have already seen that \( \chi'(K_{2\ell}) = 2\ell - 1 \) and \( \chi'(K_{2\ell+1}) \leq 2\ell + 1 \). So already we have examples of graphs, some of which have \( \chi'(G) = \Delta(G) \) and some for which \( \chi'(G) \) might be as high as \( \Delta(G) + 1 \). As it turns out, provided that each pair of vertices is joined by either one edge or none, then these are the
only two possible values for $\chi'(G)$, as was proved by Vadim Vizing in the 1960s (this result is proved in most graph theory textbooks, such as [5]).

**Vizing’s Theorem** If for each pair of vertices of a graph $G$ there is at most one edge between them, then $\chi'(G) \in \{\Delta(G), \Delta(G) + 1\}$.

Given that there are only two possible values for the chromatic index, it has become common practice to say that graphs for which $\chi'(G) = \Delta(G)$ are Class 1 and that graphs for which $\chi'(G) = \Delta(G) + 1$ are Class 2. Examples of Class 1 graphs include complete graphs with an even number of vertices, as well as all bipartite graphs; a graph is called *bipartite* if its vertex set $V$ can be partitioned into two subsets $A$ and $B$ so that every edge of the graph has one of its two vertices in $A$ and the other in $B$. Bipartite graphs have numerous applications; indeed, whole books have been written just on bipartite graphs (see [1] for one of them). Class 2 graphs include odd-length cycles (e.g., the graph having $V = \{1, 2, \ldots, 2\ell + 1\}$ and $E = \{\{1, 2\}, \{2, 3\}, \ldots, \{2\ell, 2\ell + 1\}, \{1, 2\ell + 1\}\}$).

Questions regarding how to identify which class a particular graph might be are natural to ask. As an example, in Figure 5 is the famous Petersen graph.

**Exercise 3** Determine whether the Petersen graph is Class 1 or Class 2.

![Figure 5: The Petersen graph](image)

For some graphs, there is an easy way to determine their class. To give a definition, we will say that a graph $G$ is *overfull* if $|E|$ strictly exceeds $\Delta(G)\lfloor \frac{|V|}{2} \rfloor$, where the notation $|S|$ denotes the cardinality of the set $S$ and $|x|$ denotes the greatest integer not exceeding the real number $x$ (so for example $[\pi] = 3$, $[7] = 7$ and $[-\pi] = -4$).

Amanda Chetwynd and Anthony Hilton pioneered some of the research on overfull graphs [2]. It is easy to prove that any graph that is overfull must be Class 2. By way of contradiction, suppose that there exists an overfull graph $G$ that happens to be Class 1. Since it is Class 1, $\chi'(G) = \Delta(G)$. Moreover, each colour can be used
on at most \( \left\lfloor \frac{|V|}{2} \right\rfloor \) edges, for if a colour occurred on any more edges then at least two edges of that colour would have to meet at a common vertex. With \( \Delta(G) \) colours, each occurring on at most \( \left\lfloor \frac{|V|}{2} \right\rfloor \) edges, it follows that \( |E| \leq \Delta(G) \left\lfloor \frac{|V|}{2} \right\rfloor \), in violation of the graph being overfull. Thus we have obtained the desired contradiction, from which we conclude that the graph cannot be Class 1.

Having previously determined that \( \chi'(K_n) \leq \Delta(G) + 1 \) when \( n \) is odd was not itself a proof that complete graphs with an odd number of vertices are Class 2. However, by verifying that any complete graph with an odd number of vertices is overfull, we can now confirm that \( K_{2t+1} \) is Class 2. The Petersen graph is also Class 2.

It is not too hard to deduce that if a graph is overfull then it necessarily must have an odd number of vertices. This condition is not sufficient though, for there do exist Class 1 graphs having an odd number of vertices (simply refer to Figure 4 to see an example).

The examples that we have seen so far have not been very difficult. However, determining whether a given graph is Class 1 versus Class 2 is generally not an easy problem. Indeed, Ian Holyer proved in 1981 that it is so hard that it is NP-complete [3]. Nevertheless, motivated both by scientific curiosity as well as the applications that exist for edge colourings, this continues to be an active area of research whereby people try to find faster colouring algorithms and also try to establish that certain types of graphs are Class 1 versus Class 2.

References


