Fair Division

Contents

Introduction

The origin of cake itself is difficult to trace but it is thought to date back to ancient times [1]. It is fair to assume that the concept of sharing goes back even further.

A fair division of an object is a distribution between n people such that everyone believes they have received at least a $1/n$ share. A similar problem is that of envy-free division. This is where every person believes that they have received a fair share and that they have a portion that is equally as valuable as any other. This means that nobody would want to trade their share after it has all been distributed.

Many of the problems and concepts with the mathematical process of fair division can be reduced to the issue of fairly dividing a cake. Mathematically, we define S to be the whole cake. Each slice is a subset and $S = \{S_1\} \cup \{S_2\} \cup \ldots \cup \{S_n\}$, where S_i is the i^{th} slice. Therefore, the whole cake is the disjoint union of all the slices. We also have the intersection, $\{S_i\} \cap \{S_j\} = \emptyset$ (the empty set) for $i \neq j$. If we assign the cake a total value of 1, then every slice has a value between 0 and 1 depending on what fraction of the cake a person thinks it is worth.

When applying this problem in a mathematical context, certain theoretical assumptions need to be made about the cake and the people it is to be divided between. We assume the cake is an object that can be sliced and recombined as many times as is necessary without losing any of its value. Throughout this paper, the ‘cutter’ refers to the person who cuts the cake. The people who are sharing are assumed to be rational when dividing the cake, and they do not need to know each other’s preferences of certain parts. For example, a person would be considered irrational if they deemed a slice unacceptable as a whole but acceptable if it was cut into two or more smaller slices.

Protocols were composed to produce fair and systematic ways to divide a cake. These are simply instructions that each person must follow in order for the cake to be divided fairly.

History of Fair Division

Throughout this chapter, we will denote the n people as $A_1, A_2, \ldots, A_n$. It was not until the 1940s that a mathematical method to divide a cake fairly between more than two people was first devised [2, P9]. Even when this was published, it was not completely satisfactory. A person could receive a slice they would originally deem acceptable, but after looking at other slices, they may believe that someone has received a larger slice than they did. This was finally solved in 1994, when Alan Taylor and Stefan Brams published their paper, *An Envy Free Cake Division* [2, P14].

The first means of dividing cake is ‘You Cut, I Choose’ which works for two people ($n=2$). This is one of the simplest, yet most effective, protocols and is still used in everyday life. Also known as ‘Divide and Choose’, this method guarantees envy-free distribution for two people.

Step 1 A_1 cuts the cake into two slices.
Step 2 A_2 chooses one slice and A_1 receives the other.

Both parties are satisfied with what they receive. A_2, having picked first, receives the slice she considers largest. We can safely assume that A_1 must consider both slices to be exactly half; otherwise he could end up with a smaller slice.

It was during the Second World War when Polish mathematician, Hugo Steinhaus, made a breakthrough in this problem [2, P12]. He advanced ‘You Cut, I Choose’ to make it suitable for more than two people. Steinhaus came up with *The Proportional Protocol for n=3.* This is set out very clearly in Brams and Taylors’ *An Envy-free Cake Division Protocol.*

Step 1 A_1 cuts the cake into three slices.
Step 2 A_2 is given a choice. She can pass (do nothing) or she can choose to label two of them as ‘bad’.
Step 3 If A_2 passed in Step 2, then A_3, A_2, and A_1, in that order, take a slice.
Step 4 If A_2 did not pass at Step 2, then A_3 is given the same options that A_2 had in Step 2. He must ignore labels given by A_2.
Step 5 If A_3 passed in Step 4, then A_2, A_3, and A_1, in that order, take a slice.
Step 6 If A_3 did not pass at Step 4, then A_1 is required to take a slice that both A_2 and A_3 have labelled as ‘bad’.
Step 7 The other two slices are reassembled, making a new cake. A_2 cuts this new cake into two slices.
Step 8 A_3 takes the slice that he wants.
Step 9 A_2 takes the remaining slice.

A_1 divides the cake into what he believes are three equal slices. He must attempt to cut the cake into equal slices, otherwise he could end up with a smaller slice. Therefore, he must be satisfied to receive any of them. A_2 passes if she thinks two or more of the slices are at least $1/3$ of the cake. If she decides to pass, then A_3 gets to choose his slice first. A_3 is satisfied with his choice as he gets whichever slice he prefers. A_2 now chooses a slice, and, as she was satisfied that at least two of the slices were an acceptable size, there is at least one slice left that she will be satisfied with. The last person to get their slice is A_1 and as shown above, he is satisfied with any of the slices.

Another possibility is that A_2 does not pass on her go. This means that she believes that two slices are less than $1/3$ of the cake. She labels which slices she believes are too small as ‘bad’. A_3 must ignore the labels and is offered the choice of passing or not passing - just as A_2 was before. If A_3 passes, he is satisfied with at least two of the slices.

A_2 can choose any slice and there will still be one that A_3 is happy to receive. A_2 would then choose first, followed by A_3, and then A_1 gets the remaining slice. Therefore, all three people are satisfied.

However, if A_3 also believes that two slices are not a fair size, he will decide not to pass. A_1 then has to have a slice that both A_2 and A_3 have labelled as being too small. Having cut the cake, he is satisfied he is receiving $1/3$ of the cake so is content with this result. The cake is then reassembled, and A_2 cuts it into what she believes are two equal slices. As in 'You Cut, I Choose', A_3 chooses his slice, and A_2 must take the remaining slice. Thus, everyone is content with what they have received. By their own measure, they have all received at least $1/3$ of the cake.

Despite the protocol being more useful than the simple 'You Cut, I Choose', it didn't solve the problem for sharing between n people. If there were more than three people who wanted cake, this protocol was of no real use.

Some of the protocols that we talk about refer to a concept called trimming. Non-mathematically speaking, this is exactly as the term suggests. You can cut part of one slice away, to make that slice smaller. You would also be left with the trimmings which are the bits you have cut off.

Shortly after Steinhaus published his protocol, Stefan Banach and Bronisław Knaster extended it to make it suitable for n people [2, P13]. This is known as 'The Last Diminisher'. This protocol is the first one which involves trimming. It is important to remember that the cake we use here is a mathematical cake. We can trim the cake and recombine it without any difficulty, and without losing any of the total value.

Step 1	A_1 cuts a slice, S_1, of the cake.
Step 2	A_2 is given the choice of either trimming the slice or passing (doing nothing). The slice S_1, now perhaps trimmed, is renamed S_2. The trimmings are set aside.
Step 3	For $3 \leq i \leq n$, A_i takes the slice S_{i-1} and proceeds exactly as A_2 did in Step 2, with the resulting slice now called S_i.
Step 4	The last player to trim the slice, or A_1 if no one trimmed it, is given S_n.
Step 5	The trimmings are recombined with the remainder of the cake, and Steps 1-4 are repeated for the remaining $n-1$ people.
Step 6	Step 5 is iterated until there are only two people left. The last two people use 'You Cut, I Choose'.

A_1 is randomly chosen and this person is the cutter. He cuts what he considers to be exactly $1/n$ of the cake. Similarly to Steinhaus' protocol, the second person gets a choice. She decides whether she wants to pass or trim the slice. If she decides to pass, she is satisfied that the slice is at most $1/n$ of the cake (it could be less than that). She decides to trim the slice if she thinks the slice is larger than $1/n$ of the cake. She trims it to what

she considers to be exactly $1/n$. The trimmings are set aside and are ignored at this stage. Moving in order, everyone has the choice to trim the slice or pass it on. The last person to trim a slice, takes it. If nobody trims, then the cutter gets to keep that slice. When someone receives their slice, the trimmings (if any) are re-attached to the cake and this process is repeated. This continues for every slice, excluding people who have already received one, until only two people remain. When this is the case, 'You Cut, I Choose' is applied. This guarantees everyone is getting what they perceive to be a fair slice of the cake.

What still remained unsolved was how to divide the cake so that it is envy-free. In all previous protocols, everyone is satisfied that they received at least $1/n$ of the cake. However, there is a chance that someone may be satisfied with their slice but feel someone else has a larger one. The next task was to create a protocol where nobody would want to swap their slice at the end. This took much longer to solve.

In 1960, John Selfridge managed to find an envy-free protocol for three people. He didn't publish this however, and it was not until the 1990s when John Conway solved this problem and had it published. This protocol has been credited to both mathematicians and is known as *An Envy-free Protocol for n=3.* Here, we need a slightly different definition of trimming. To trim a slice would create two subsets of that slice. If you trimmed the slice $\{S_i\}$, you would now have $\{T_i\}$ and $\{L_i\}$, where T_i refers to the trimmed slice, L_i refers to the leftovers of each slice and L is the total of all the leftovers. Therefore, $\{S_i\}=\{T_i\}\cup\{L_i\}$ and $\{T_i\}\cap\{L_i\}=\emptyset$ for all i. Also, $\{L\}=\{L_1\}\cup\{L_2\}\cup\ldots\cup\{L_n\}$ and $\{L_i\}\cap\{L_j\}=\emptyset$ for $i\neq j$.

Step 1 A_1 cuts the cake into three slices.

Step 2 A_2 is given the choice of either passing or trimming one of the three slices. If A_2 trimmed a slice, then the trimmings are named L (for leftovers) and set aside.

Step 3 A_3, A_2, and then A_1, in that order, choose a slice. One of which may have been trimmed in Step 2. If A_2 did not pass in Step 2, then he is required to choose the slice he trimmed if A_3 chose a different one.

Step 4 If A_2 passed at Step 2, we are done. Otherwise, either A_2 or A_3 received the trimmed slice, and the other received an untrimmed slice. Whichever player received the untrimmed slice now divides L into three slices. Call this player the 'divider' and the other the 'non-cutter'.

Step 5 The three slices into which L has been divided are now chosen by the players in the order: non-cutter first, cutter second, divider third.

One person is randomly chosen to be the cutter. He cuts the cake into what he believes are three equal slices. The second person then gets a choice. If she believes that there are at least two slices which are tied to be the largest, then she passes. If this is the case, the third person chooses a slice he wants, the second person chooses what she wants,

and then the cutter gets the last slice. All three are satisfied regardless of which slices the others get. Therefore, it is envy-free.

If the second person believes there is one slice which is the largest, she can trim it so that it is, in her mind, equal largest. As before, anything that has been trimmed is put aside at this point. The third person now gets to choose the slice he considers to be the largest. He is satisfied with what he has received, regardless of what anyone else gets, as he has chosen first.

The second person now gets to choose her slice. If she trimmed a slice and the third person did not choose it, the second person must choose that slice. If this happens, she is satisfied, as in her mind, the trimmed slice is equal to the largest. If the third person chose the trimmed slice, the second person chooses the slice that she believes is equally large. Therefore, she is satisfied in this case too. The original cutter receives the remaining slice and is satisfied as he cut all slices to be equal in his mind.

If the second person trimmed a slice, then there are leftovers. Someone, other than the cutter, will have received an untrimmed slice. This person, the divider, now cuts the leftovers into what they believe to be three equal slices. The order of who chooses now is important. The person who received the trimmed slice of the main cake gets to choose their share of the leftovers first. This is because the other two people will not be jealous regardless of which one they choose. The cutter of the main cake cannot be jealous as, in his mind, the total of the leftovers and the trimmed slice would equal his slice so he is receiving more than his fair share in his opinion. To the divider, all three slices of the leftovers are the same. Therefore, the order of choice for leftovers is:

(1) The person who originally received the trimmed slice
(2) The original cutter
(3) The divider.

This method was the first to guarantee that everyone will be satisfied regardless of what other people receive. The problem remains that this is only suitable if there are three people wishing to share the cake.

The idea of trimming was key to how Steven Brams and Alan Taylor eventually solved the problem of *An Envy Free Cake Division* in 1994. In their paper of the same name, they write:

> ‘The central feature of our envy-free protocol, like that for the $n=3$ protocol, is that players trim slices of the cake to create ties, rendering them indifferent among these slices. When $n>3$, however, one needs to start the trimming and choosing process leading to an envy-free partial allocation with more slices than there are players.’

This protocol is much more complicated than the others and as a result is included in appendix A at the back of this paper. Included with the protocol are small asides which help explain each step and why it is envy-free.

Method Using Matrices

We will now introduce some important mathematical results which lend themselves very nicely to the problem of fair division. This allows us to prove that a fair division of the cake must always be possible using the following protocol utilising matrices, similar to the way described in [6].

Step 1 Randomly assign the n people as $A_1, A_2, \ldots, A_n$.

Step 2 A_1 cuts the cake into n slices. Denote the n slices of cake as $S_1, S_2, \ldots, S_n$.

Step 3 The remaining people $A_2, \ldots, A_n$ assign a value of either 1 or 0 to each slice of cake. If they deem that slice acceptable (at least $1/n$) they assign it a value of 1. If they deem it unacceptable, they assign it a value of 0.

Step 4 These values are entered into a matrix with entries (i,j), corresponding to the i^{th} row and j^{th} column. With each entry being a 1 or 0 depending on whether the i^{th} person believes the j^{th} slice represents greater than or equal to $1/n$ of the cake.

The randomisation is important as the protocol only guarantees the cutter a maximum of $1/n$ of the cake. The assignments of $A_2, \ldots, A_n$ can be arbitrary. It is safe for us to assume that A_1 would cut all n slices such that he would be satisfied with receiving any of them. This is logical because if he cut some slices larger than $1/n$, it is possible he would receive less than his fair share. Therefore, the first row of the matrix will always be all 1s. Every person must consider at least one of the slices to be acceptable. As the cake is given a total value of 1, it would be impossible for all slices to be less than $1/n$. So, there must be at least a single 1 in every row. This kind of matrix is called a 0-1 matrix.

Taken directly from [10, P1] the definition of a permanent is as follows.

> For an $n \times n$ matrix $C = (c_{ij})$, where $1 \leq i \leq n$ is the row and $1 \leq j \leq n$ is the column. The permanent of C, denoted $per(C)$, is defined by
>
> $$per(C) = \sum_{\pi \epsilon Sn} \prod_{i=1}^{n} x_{i,\pi(i)},$$
>
> where S_n is the symmetric group on n elements.

The only difference to the determinant, denoted $det(C)$, is in the signs of terms:

$$det(C) = \sum_{\pi \epsilon Sn} sgn(\pi) \cdot \prod_{i=1}^{n} x_{i,\pi(i)}.$$

To make this definition clearer here is an example of how to calculate the permanent of the following matrix:

$$Y = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

The permanent of Y, $per(Y)$, is:

$$per(Y) = 1(5\cdot 9 + 6\cdot 8) + 2(4\cdot 9 + 6\cdot 7) + 3(4\cdot 8 + 5\cdot 7)$$
$$= 93 + 156 + 201$$
$$= 450.$$

In the case of $0-1$ matrices, we observe that the permanent has value 0 if and only if every diagonal contains a 0 entry [6]. Ferdinand Georg Frobenius and Dénes König proved a theorem based on these permanents.

Frobenius-König (F-K) theorem [3, P32]:

> Let C be an nxn square matrix. A necessary and sufficient condition for every diagonal of C to contain a zero entry is that C contains an sxt zero submatrix such that $s+t=n+1$.

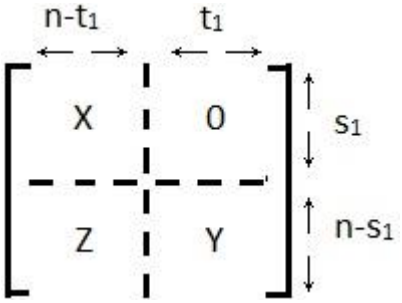
This theorem is equivalent to:

> Let C be an nxn square non-negative matrix. Then $per(C) = 0$ if and only if C contains an sxt zero submatrix such that $s + t = n + 1$.

The proof of this theorem, taken directly from [3, P32-34], is as follows.

> Throughout this proof $C(h|j)$ refers to the matrix C with the row h and column j removed and $C[h|j]$ refers to just the h^{th} row and j^{th} column of C. We prove the necessity by induction on n.
>
> Let $per(C) = 0$. If $n=1$, then $C = [0]$, the zero matrix. Assume that the condition is necessary for all square $0-1$ matrices of order less than n. If every entry of C is zero, the proof of necessity is finished. Suppose that $c_{hk} > 0$. Then,
>
> $$0 = per(C) = \sum_{j=1}^{n} c_{hj} per(C(h|j))$$
>
> and since all the products $c_{hj}.per(C(h|j))$ are non-negative and c_{hk} is positive, we must have
>
> $$per(C(h|k)) = 0.$$
>
> Hence, by the induction hypothesis, we can find an $s_1 x t_1$ zero submatrix of $C(h|k)$ such that $s_1 + t_1 = (n-1) + 1$. Permute the rows and columns of C so that the zero submatrix appears in the top right corner.

$B = PCQ =$



> Where P and Q are matrices to permute the rows and columns of C such that the $s_1 x t_1$ zero submatrix appears in the top right corner.

Now, neither $n-s_1$ nor $n-t_1$ can be zero, because s_1 and t_1 are positive integers and $s_1+t_1=n$. Since $n-s_1=t_1$ and $n-t_1=s_1$, it follows that X is s_1-square and Y is t_1-square. Then, $per(C)=per(B)=per(X)\bullet per(Y)$ because all terms in the $t_1 x s_1$ submatrix Z will not make any contribution to the permanent. This is because of the $s_1 x t_1$ zero submatrix diagonally opposite. Thus,

$$0=per(C)=per(B)=per(X)\bullet per(Y).$$

As a consequence, either $per(X)$ or $per(Y)$ must be zero.

We can assume, without loss of generality, that $per(X)=0$. Then, by the induction hypothesis, X contains a uxv zero submatrix such that $u+v=s_1+1$. Suppose that $X[\alpha_1,\ldots,\alpha_u|\beta_1,\ldots,\beta_v]=0$; that is $B[\alpha_1,\ldots,\alpha_u|\beta_1,\ldots,\beta_v]=0$. Consider the submatrix

$$C=B[\alpha_1,\ldots,\alpha_u|\beta_1,\ldots,\ \beta_v\ ,\ s_1+1,s_1+2,\ldots,n].$$

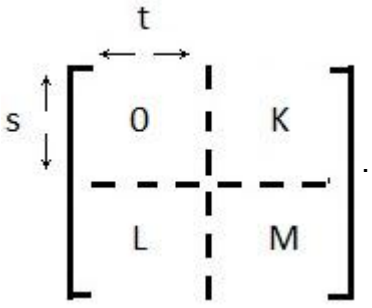
It is a zero submatrix of C with u rows and $v+(n-s_1)$ columns. Moreover,

$$u+v+(n-s_1)=(u+v)+n-s_1=s_1+1+n-s_1=n+1.$$

This completes the proof of necessity.

To prove the converse, suppose that C contains an sxt zero submatrix with $s+t=n+1$. $Q'_{r,m}$ is defined as the entry in the r^{th} row and m^{th} column of the matrix Q'.

Permute the rows and columns of C so that

$B'=P'CQ'=$ 

.

Where P' and Q' are matrices to permute the rows and columns of C, such that the sxt zero submatrix appears in the top left corner.

The Laplace expansion theorem states that, if C is an mxn matrix, $2\leq m\leq n$, and $\alpha\in Q'_{r,m}$, then

$$per(C)=\sum_{w\epsilon Q'} per(C[\alpha|w])per(C(\alpha|w)).$$

Using this gives

$$per(C)=per(B')=\sum_{w\epsilon Q'} per(B'[1,\ldots,s|w])per(B'(1,\ldots,s|w)).\ (\dagger)$$

However, $B'[1,\ldots,s|w]$ is an sxs submatrix of $B'[1,\ldots,s|1,\ldots,n]$ and the latter matrix has at most

$$n-t=s+t-1-t=s-1$$

non-zero columns. Hence, every submatrix $B'[1,\ldots,s|w]$ must have at least one zero column, and therefore $per(B'[1,\ldots,s|w])=0$ for all $w \in Q'_{s,n}$. It follows from (†) that $per(C)=0$.

■

There is some amount of controversy surrounding the origins of this theorem, stemming from the relation between the F-K theorem and other results by König. Henryk Minc states [3, P32] 'It was first obtained by Frobenius, then later reproved by König by elementary graph-theoretical methods, and again re-proved by an elementary method by Frobenius.' It is likely that neither König, nor Frobenius, would be too pleased to have his name used in conjunction with the other; Frobenius regarded König's work to be inferior, whereas König felt all Frobenius did was appropriate his own work. In fact König explicitly said [4, P5] '[Frobenius] is revealed as an extraordinarily quarrelsome and aggressive man inclined to temper tantrums who was immoderate in his dislike of people and things. His lectures were excellent in their polish, richness and depth as is attested by an unnamed correspondent. As a researcher, Frobenius enjoyed an excellent reputation. But, it is surely one of his greatest merits that he recognised I. Schur's greatness and importance from the first day and sought to further him on every occasion that offered itself.' The F-K theorem relates to Hall's marriage theorem and the König-Egeváry theorem.

Hall's marriage theorem [5, P21]:

> A set of women can always find a husband from amongst the men they know if and only if for each r, any set r of the women knows at least r men between them.

There is a partial proof of this theorem included in appendix B [5].

There are numerous corollaries to this famous theorem. The one that is particularly useful, when leading into the König-Egeváry theorem for matrices, is the Hall's theorem - matrix version.

Hall's theorem - matrix version [5, P23]:

> Let M be a matrix. The following two conditions are equivalent:
> (1) There exists a 0 in each row of M with no two 0s in the same column.
> (2) For any r rows of M, those rows between them contain 0s in at least r columns.

To make this clearer consider the following example:

$$M=\begin{bmatrix} 1 & 2 & 0 & \mathbf{0} \\ \mathbf{0} & 3 & 4 & 0 \\ 5 & 6 & \mathbf{0} & 7 \end{bmatrix}.$$

There is a 0 in every row, without two being in the same column (these are the three 0s shown in bold in the example). Therefore, by Hall's theorem it must also be true that for any r rows of M, those rows between them contain 0s in at least r columns.

The König-Egeváry (K-E) theorem for matrices is as follows [5, P24]:

> In any matrix the minimum number of lines to include all the 0s is equal to the maximum number of 0s with no two in the same line. Here, a line means a row or a column.

For example, consider the following matrix:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 0 & 0 & \mathbf{0} \\ 5 & \mathbf{0} & 6 & 7 \\ \mathbf{0} & 8 & 8 & 8 \end{bmatrix}.$$

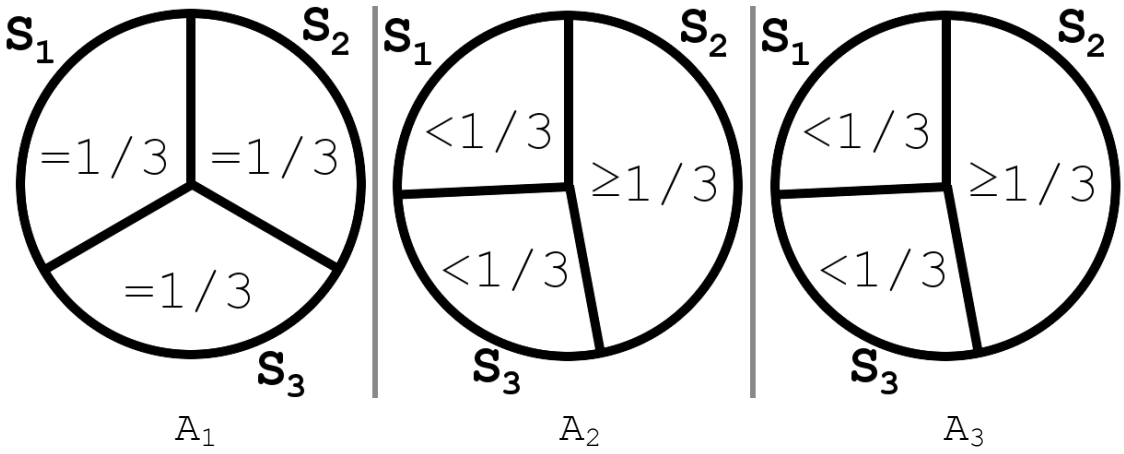
Here the maximum number of 0s, with no two on the same line (highlighted) and the minimum number of lines to include all the 0s, is three. These are always equal by the K-E theorem.

There is a proof of this theorem from [5] included in appendix C.

Now that we have looked into the origins of the F-K theorem, we need to show how it applies to the problem of fair division. Consider the following matrix:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

This is the matrix formed when A_1 cuts the cake and when A_2 and A_3 consider only S_2 as acceptable. As a visualisation, this is how each person could see the cake.



The permanent of this matrix is 0 as the entries $(2,1)$, $(2,3)$, $(3,1)$ and $(3,3)$ form a $2x2$ zero submatrix. The formula given in the F-K theorem, $s+t=n+1$, holds as $2+2=3+1$. The permanent equaling 0 means there is not a simple solution by just distributing slices. However, S_1 or S_3 can be given to A_1. The remaining slices can then be recombined and split using 'You Cut, I Choose'. As A_2 and A_3 deem S_1 and S_3 unacceptable there is more than two thirds left, in their opinion, for them to recombine and share.

Now we have these mathematical theorems, we can prove that n people can divide a cake so that each is satisfied that they have received at least $1/n$ by their own measure. For this we use induction on n. Following the proof from [6, P25-28], we take the induction

hypothesis that any number of people less than `n` can split the cake fairly. A matrix with a diagonal containing all `1`s can easily be distributed to the people so they are all satisfied. The case when every diagonal contains a `0`, by the F-K theorem, must have permanent `0` and also must contain an `sxt` submatrix of `0`s with $s+t=n+1$. As the first row of the matrix represents which slices the cutter thinks are worth `1/n` of the cake, all the entries must be `1`. It is in his interest to cut the cake into slices of size `1/n`, as if they are unequal sizes he may receive one of the smaller slices. We therefore have the condition that $s \leq n-1$. Suppose we want to make `s` as large as possible, this is $s=n-1$. The F-K theorem, $s+t=n+1$, must be satisfied, therefore $t=2$. A matrix of this kind has this appearance:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 0 & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & \cdots & * \end{bmatrix},$$

where `*` can represent a `1` or `0`, with the only condition being that there must be at least a single `1` on each row.

In this case, the first slice could be given to the cutter who is satisfied regardless of which slice he receives. In the opinion of everyone else, this slice was less than `1/n` of the cake. Therefore, what is left must constitute more than `(n-1)/n`. As this is a mathematical cake, these leftover slices can be reconstructed to be reconsidered as a whole cake and the `n-1` participants can then divide it in a way which ensures they all get their fair share.

Now consider the remaining case. We must have $s<n-1$ because of how the protocol was constructed. When we find the `sxt` submatrix of `0`s, we choose it such that `s` is maximal (as large as possible). In this case, the matrix would look like:

$$\begin{array}{c} \overbrace{}^{t-1} \\ \begin{array}{cc} t-1 & \left\{ \right. \\ s & \left\{ \right. \end{array} \left[\begin{array}{ccccc|c|ccc} 1 & 1 & \cdots & 1 & 1 & 1 & 1 & \cdots & 1 \\ \multicolumn{5}{c|}{E} & * & \multicolumn{3}{c}{*} \\ \hline 0 & 0 & \cdots & 0 & 0 & & & & \\ 0 & 0 & \cdots & 0 & 0 & & & & \\ \vdots & \vdots & \ddots & \vdots & \vdots & & & * & \\ 0 & 0 & \cdots & 0 & 0 & & & & \end{array}\right]. \\ \underbrace{}_{t} \end{array}$$

Now, consider the `(t-1)x(t-1)` matrix in the upper left hand corner and any `pxq` sub matrices of `0`s that can be found within it. Explicitly this is the matrix `D`:

$$\mathrm{D} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ & & E & \\ & & & \end{bmatrix}.$$

Combining the pxq submatrix found in this matrix with the sxt matrix we had in the entire nxn matrix, we now have a $(s+p)xq$ submatrix of 0s in the nxn matrix. Now, using the formula from the F-K theorem, if $(s+p)+q \geq n+1$, it could be possible to find a submatrix of 0s whose dimensions sum to $n+1$ but which has more rows than we chose earlier (in the matrix D). However, since we chose s to be maximal, it follows that $(s+p)+q < n+1$. This is the same as $p+q < n+1-s = t$.

This shows that the matrix D cannot contain a pxq submatrix of 0s with $p+q \geq (t-1)+1$ and, using the F-K theorem, there must be at least one diagonal of D that does not contain a 0. As the diagonal does not contain a 0, it must represent a fair way of dividing the first $t-1$ slices of cake among the first $t-1$ people. The remaining s people believe the total remaining portion of cake represents a value of more than their fair $1/n$ each. This is because they all deemed the first $t-1$ slices as less than $1/n$. Once again, our induction hypothesis guarantees that these s people can share this remainder so each believes they are receiving at least $1/n$ of the cake by their own measure. As we are dealing with a mathematical cake, there is no problem with reforming the remaining slices and conducting the procedure again.

Computer Script

To complement this section of the paper, we have written a web-based computer script which uses the matrices protocol (from the previous chapter) to determine a fair division of the cake. Given a cake that has already been cut into n slices, the script produces a solution that each person will deem fair. The script can only satisfy the condition that everyone believes they have obtained at least $1/n$ of the cake by their own measure. However, the script cannot provide an envy-free division.

First and foremost, n is defined as the number of people wishing to share the cake and is entered on the front page. If $n=2$, the script states that 'You Cut, I Choose' should be used, and if $n=1$, the script kindly informs you that the cake is all yours! The user is then presented with a matrix which has fields to populate. We define the different people wishing to share the cake as A_i and the different slices as S_j, where i and j are integers between 1 and n. A person should randomly be chosen as the cutter and is assigned A_1. The assignments of $A_2, A_3, \ldots, A_n$ to each other person can be arbitrary. Then, the entry in the i^{th} row and j^{th} column is 1 if A_i believes that slice S_j is worth at least $1/n$ of the cake and 0 otherwise. The script will then evaluate this matrix and return a solution. This will be presented in a table and highlighted in the original matrix to simplify viewing. Only one solution to each situation will be found. The script does not attempt to evaluate all valid solutions.

The script is written in PHP (**P**HP: **H**ypertext **P**reprocessor) and uses HTML (**H**yper**T**ext **M**arkup **L**anguage) for the front end. PHP cannot natively handle matrices so a two-dimensional array of the form

```
Array(
  0 => Array(0 => M[1,1], 1 => M[1,2], … , n-1 => M[1,n]),
  1 => Array(0 => M[2,1], 1 => M[2,2], … , n-1 => M[2,n]),
    …
n-1 => Array(0 => M[n,1], 1 => M[n,2], … , n-1 => M[n,n])
);
```

is used where `M[i,j]` denotes the entry in the i^{th} row and j^{th} column of the matrix. If we define `$matrix` to be an array of the above form, then this allows us to easily access a specific entry with `$matrix[$i][$j]` where `$i` and `$j` are integer variables representing the $(i+1)^{th}$ row and $(j+1)^{th}$ column respectively (a `$` simply denotes a variable in PHP). When `n` is defined by user input, the script generates a null square matrix of size `n` and populates the top row with the value `1` repeated. This matrix is then presented to the user for amendment in the manner described above. The matrix returned is then validated. This is to ensure that:

- The top row still only contains the value `1` (which should always be true)
- Every entry contains either a `0` or `1` (any other values are undefined and converted to `0`)
- The sum of each row is at least `1` (at least one slice must be deemed at least `1/n`).

As we have seen in the previous chapter, there is always a fair solution. How simple the solution is depends on the permanent. If it is non-zero, there is a simple solution possible by just distributing the slices. However, if it is zero, a slightly more complex procedure is necessary. The script's next step is to calculate the permanent of the matrix using a custom (recursive) function that calculates permanents of submatrices in a similar fashion that you would employ by hand.

If the permanent is non-zero, we find the simple solution. To find this we devised an algorithm (denoted Ψ). The steps of which are outlined below:

(1) Label each row and column $A_1, A_2, \ldots, A_n$ and $S_1, S_2, \ldots, S_n$ respectively
(2) Calculate the total sum of each row (now referred to as 'row sum')
(3) Calculate the total sum of each column (now referred to as 'column sum')
(4) Whilst maintaining row labels, reorder the rows with smallest row sum at the top and largest at the bottom
(5) Whilst maintaining column labels, reorder the columns with smallest column sum to the left and largest to the right

(6) Starting with the first row and working left-to-right, read each entry:
    - If the entry is `0` then proceed to the next entry, or,
    - If the entry is `1` then store the current row and column label and proceed to the next row
(7) After the last row, return the associations of row label to column label as a

map.

This map contains a bijection from A_i to S_j; each and every person will have been assigned a slice. The script then highlights each association in the matrix and also displays the map as a table.

As a short example, a user is wishing to share a cake between four people and, on the script, they enter the matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The script informs the user that the permanent of this matrix is 2 and that A_1 receives S_1, A_2 receives S_3, A_3 receives S_2, and A_4 receives S_4 then displays this information in a table. Also, the matrix will be highlighted as demonstrated below:

$$\begin{bmatrix} \mathbf{1} & 1 & 1 & 1 \\ 1 & 1 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 1 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix}.$$

When the permanent is zero, there is not a simple solution, but there is still a procedure that enables each person to receive at least $1/n$ by their own measure. This involves immediately distributing some slices and allowing other people to recombine and re-share the remaining slices between them. The cutter will always be able to receive a slice immediately. This method has a different and more complex algorithm, which we devised. The steps of which are outlined below:

(1) Define $M[i,j]$ to be the i^{th} row and j^{th} column in the original matrix, $P[i,j]$ to be the i^{th} row and j^{th} column in a new matrix, $R_M[i]$ to be the row sum of the i^{th} row in M and $C_M[j]$ to be the column sum of the j^{th} column in M. Then:

$$P[i,j]=M[i,j]/R_M[i]$$

(2) Define $Q[i,j]$ to be the i^{th} row and j^{th} column in a new matrix, $R_P[i]$ to be the row sum of the i^{th} row in P and $C_P[j]$ to be the column sum of the j^{th} column in P. (Remember, n has already been defined.) Then:

$$Q[i,j]=n \bullet P[i,j]+(R_P[i]+C_P[j])/R_P[i]$$

(3) Obtain the maximum entry from each row

(4) Let the maximum of these maximums, from (3), be denoted simply by x

(5) For each row where x is the maximum of the row and working left-to-right, read each row entry (i is the row number and j is the column number):

- If the entry equals x and j has yet to be assigned then assign j to i, or,
- If the entry equals x and j has already been assigned then proceed to

the next entry that equals x that has not had j assigned and assign that j to i, but,
    - If all entries equalling x have had j assigned then assign the minimum entry in the current row (that does not have j assigned) to i
(6) Copy the matrix to a new matrix T. For each assignment of j to i, remove the i^{th} row and j^{th} column of T
(7) Store the assignments as a 'map to share'
(8) Calculate the permanent of T. Then:
    - If the permanent is 0, go back to step (1) with T as the new original matrix, but,
    - If the permanent is not 0, start algorithm Ψ on T and store the returned map as the map to give
(9) Return the 'map(s) to share' and the single 'map to give'.

The 'map to give' contains associations between a single person, or multiple people, that need not partake any further in the sharing exercise, and, the slice or slices they can receive without any remaining people complaining. All the people who have yet to receive a slice must deem any slice(s) assigned here as unacceptable, otherwise they would have reason to complain. The cutter, A_1, will always be in this map since they believe all slices are acceptable. A 'map to share' isn't strictly a map as it only contains a list of people and a list of slices. The listed people need to recombine the listed slices and re-share them in exactly the same manner as we initially used, just with a smaller n. The script highlights which people and slices in the matrix need to be re-shared and which can be removed and also displays this information in a table.

Consider a user wishing to share a cake between six people. On the script the user enters the matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

The script informs the user that the permanent of this matrix is 0 and that there are three stages to continue. The first stage is that A_1 receives S_3 (which is highlighted in green in the matrix). The second stage is that A_4, A_5 and A_6 receive S_1, S_5 and S_6 to recombine and re-share (which is highlighted in blue in the matrix). They are satisfied with this because only slices they deemed of an unacceptable size have been given away. Therefore, in their opinion there is greater than $(6-3)/6 = 1/2$ of the original cake left which is enough for each remaining person to receive at least $1/6$ each by their own measure. The third stage is that A_2 and A_3 receive S_2 and S_6 to recombine and re-share (which is highlighted in yellow in the matrix). By the same argument as above, there is at least enough of the cake remaining to obtain a slice of size at least $1/6$ each by their own measure. These highlights are demonstrated below:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

The script is currently online at *http://rowanparker.com/cake* complete with source code and contact details. The source code is also included in appendix D for reference.

Applications

So far we have witnessed a few methods that show how it is possible to fairly divide a cake, but what use is fair division in the world today? Does it go beyond simply splitting food between a group of people? Simply put, yes it does. Before demonstrating any examples, we first need to explore a particularly remarkable property about fair division.

Disagreement is something that, in the case of fair division, can be quite beneficial. Consider the very basic two person cake division problem, as has been seen so far we (probably) would initially go into the usual routine of 'You Cut, I Choose', where both people receive a fair share of the cake. This is a very reasonable approach but, in terms of each person's valuation of the cake, it could be better for both of them.

To show how they can both get more than half of the cake, by their own valuation, assume that both these people disagree (even if it is only slightly) on the half-way point of the cake and put them both in the role of the divider.

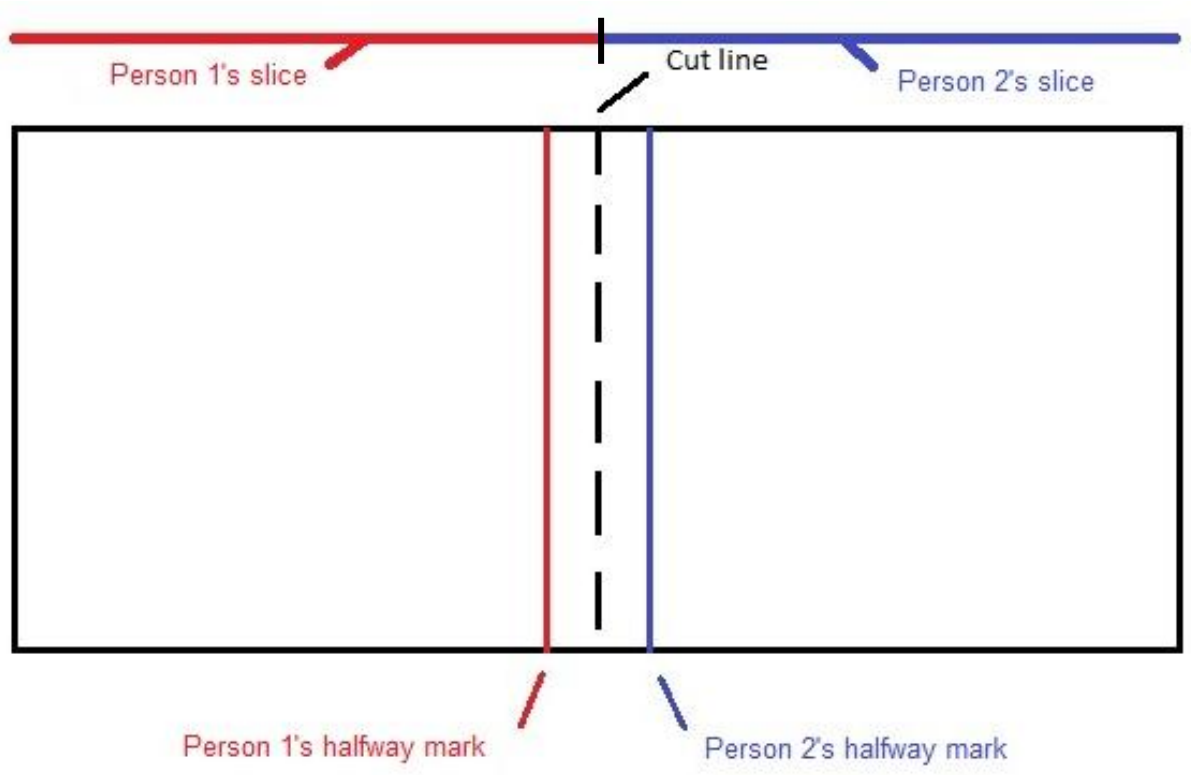
Step 1 Define A_1 and A_2 to be the people involved. Both A_1 and A_2 secretly mark where they would cut the cake if they were to be the divider in the 'You Cut, I Choose' scenario.

Step 2 Reveal both marks made by A_1 and A_2 to each other.

Step 3 Have a randomly chosen person cut the cake in the middle of the two marks to form two slices.

Step 4 Give each person the slice that has their mark line included in the slice.

Assuming that both people disagree on the half-way point, each person will get a slice of the cake that is worth more, in their mind, than the other person's slice. An illustration of this can be seen below.



If we have n people to split a cake between, we can take a similar but slightly altered approach. Here, A_i is the i^{th} person where $1 \leq i \leq n$.

Step 1 Randomly assign the n people as $A_1, A_2, \ldots, A_n$. Each A_i puts a mark on the cake where they would cut it.

Step 2 Starting from one end of the cake, find the first two cut marks and cut the cake between these points.

Step 3 Give this slice to the A_i who placed the first cut mark from the end.

Step 4 Repeat steps 1 to 3 until every person has a slice.

The strategy here is for each person to place a cut line on where they think exactly $1/n$ of the cake lies from the end. Assuming that everyone disagrees about where the $1/n$ partition lies, then there will be a first and second mark from the end. When the cake is cut here, and the slice is given to the person with the mark closest to the end of the cake, all of the other people will think this slice is worth less than $1/n$. The person who received the slice thinks he got more than what he thought was fair. Continuing the protocol, everyone will receive more than what they think is fair by their own valuation. An in depth proof of this can be seen in [6, P31].

The above phenomenon exists in a lot of practical applications that can be used in society. However, we cannot keep assuming that everyone has a different opinion, because it is entirely possible for two or more people to put the same value on the same object. Fortunately, this is of little importance. This difference means that in the event of people agreeing on the valuation of something, they may only get what they perceive to be the same as the other people involved. The outcome that nobody will feel hard done by is still very much sought after.

Consider a married couple going through a divorce. The divorce settlement of jointly owned items is something that fair division copes remarkably well with. Imagine that two people are in such a situation and between them they own a house, a boat and two cars. The problem arises when both people value the assets differently. A fair division protocol 'Sealed Bids' is proposed to solve this effectively. Here, A_i is the i^{th} person where $1 \leq i \leq n$.

Step 1 Each A_i secretly writes down a value for each item and totals these values up. Denote the total T_i for each A_i.

Step 2 Each A_i then works out their own T_i/n share of the monetary total.

Step 3 Each item is assigned to the highest bidder of that item.

Step 4 If any A_i receives items that total up to be more than their own individual fair share, they must pay out the difference to a money pool.

Step 5 Any A_i who has received less than their own fair share, withdraws money from the pool so that any received items added with this withdrawn money equals their own fair share valuation.

Step 6 Any remaining money left within the pool, after steps 4 and 5 have been completed, is to be split equally between all A_i.

This protocol, due to the disagreement involved, normally generates a supply of money left over, which can be shared equally between all of the people involved to give everyone a total of more than their fair share.

To show this, consider the following example. Two people make independent monetary assessments which are displayed in the following table.

Asset	Person 1	Person 2
House	£50000	£45000
Boat	£15000	£13000
Car 1	£11000	£13000
Car 2	£8000	£9000
Total assessment	£84000	£80000
Their share	£42000	£40000

As person 1 values the house and boat more than person 2, he is given them. Similarly, person 2 receives both of the cars. We can now work out the following.

	Items received	Their own value of the received items	Money difference to their own fair share
Person 1	House, Boat	£65000	+ £23000
Person 2	Car 1, Car 2	£22000	- £18000
Surplus			+ £5000

Now, if person 1 pays to the pool £23000 and person 2 receives £18000 from it. Both people are happy that they have got their own fair share of the assets, yet, there is still

£5000 in the pool available. This can be split equally between both person 1 and person 2 to leave each of them with more than their fair share by their own valuation. In this situation, the disagreement is in the valuation of the total of the assets. If both people had totalled the value to be the same as each other, then there is no surplus. The most important feature of the protocol is that each person always gets at least their fair share so the surplus simply functions as a bonus.

This method can easily be generalised and still works for any number of people. Indeed, the more people that are involved, the more chance there is for disagreement. An example for a three person inheritance scheme can be seen in [6, P34]. 'Sealed Bids' can also be manipulated very easily into something called 'Chore Division'. As a simple explanation, imagine a parent splitting household chores between her children. One child may prefer to do a chore that the other one dislikes doing. Each child can assign negative values to each chore and then each child receives the ones they bid the 'highest' money for (remember in this case that the highest will be the least negative). Each child then either pays out or receives chores to bring them to their fair share, and the surplus is divided between them. A fully worked example of this can be seen in [7].

A problem with most situations in the real world is that different people have contrasting valuations and preferences over the different portions of the items that are to be divided. An example of this can be seen in [8], where a few friends share a pizza with multiple toppings, with each person having different likes and dislikes of each topping. Many common problems are much more complicated and important than this scenario and a more advanced protocol is required. 'Last Diminisher' is one such protocol that provides the ability to split such complicated divisions. This method stems from the Banach-Knaster cake protocol that introduces the idea of trimming that we saw in the first chapter of this paper. Here, we can see how for real life (and cakes which have different toppings in different places) this is remarkably useful.

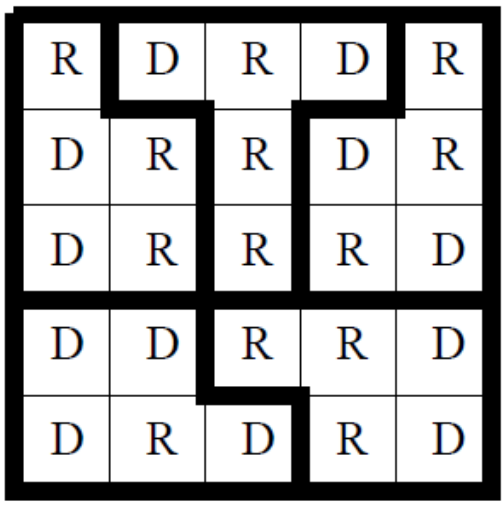
Consider a group of salespeople dividing up land, for example a county or a city. Each person wants to get an area of land that has an equal number of housing estates to the other people's regions of land. If this county has a big city in one section and mostly countryside in the rest, then the area cannot be divided up into the same size portions of land as there will clearly be more houses in a smaller area of land within the city than a similar sized area of land in the countryside. By using 'Last Diminisher', sections of land can be worked out in a similar way to the cake, but taking account of different factors within the land itself. If a person prefers to work in the city, they will value a smaller portion of the city to be $1/n$ than someone who has no preference. A visual example of this protocol can be seen in [7].

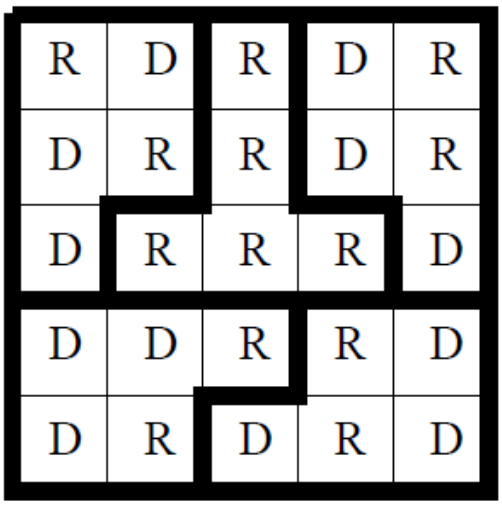
Land division has turned out to be a very important matter in modern society. The United States of America has recently utilised a quite complicated but effective protocol which solves the problem of redistricting. From [9], 'Redistricting is the political practice of dividing states into electoral districts of equal population in response to decennial census results to ensure equal representation in the legislative body.' This is a very important use

of fair division and has been completely overlooked until the turn of the century. Essentially, it allows each state to be represented fairly with the correct amount of districts headed by each political party corresponding to the percentage of votes that each party received in the election.

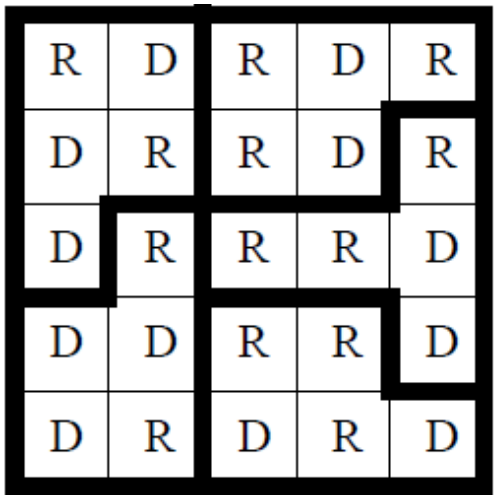
Continuing from [9], 'Where the boundaries are drawn can dramatically alter the number of districts a given political party can win. As a result, a political party which has control over the legislature can manipulate the boundaries to win a larger number of districts, thus affecting the balance of power in the U.S. House of Representatives. Obviously this poses a problem if it is not carefully divided equally. The idea of the protocol involved is to allow political parties to properly achieve the fair representation that they deserve.'

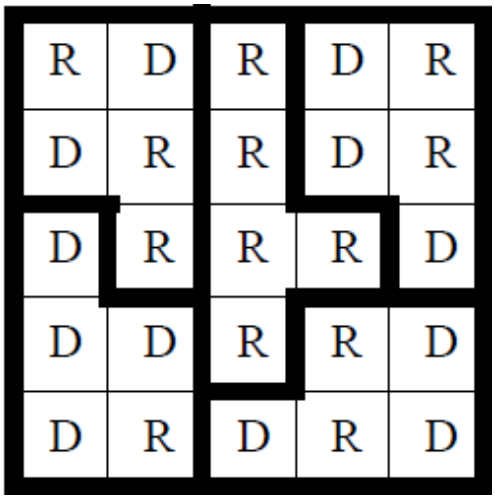
For the purpose of explanation, [9] provides a simplified example, which we'll see here. The full protocol can be seen in the paper. In previous years, most states went under a process that chose one of either the Republicans or the Democrats to be the 'drawing' party that would divide the land into districts that would each be represented by the party who received the most votes in that district. We shall simplify a state into a `5x5` grid where each square is an area of land and is represented by a `D` or an `R`, where a `D` represents that the square is a Democratic vote and an `R` is for a Republican vote. In this particular example, we shall assume there is a `52%` majority of votes for Republican. We can now generate a random grid and show how, if this state was going to be split into five districts, the drawing party will always come out with more districts than the overall votes would suggest.





In the first grid, the Republican party is the drawing party and receives four out of five of the districts. Whereas in the second grid, the Democratic party is the drawing party and also receives four out of five of the districts. This old protocol is clearly biased, yet it was still used as recently as 2002. With the addition of the new protocol, described in [9], the grid is systematically reorganised and divided to be a much fairer share. The following are two outcomes of the protocol that both provide a `2:3` ratio of districts, a much fairer division that represents the `52%` to `48%` of votes much better than the previous protocol would have allowed.





These two grids have both been drawn from the same protocol. There is an element of randomness involved, if needed, to split the divide which the 5x5 grid generates. In a much bigger and more complicated grid, such as the actual state itself, this will not cause such a big difference between the two parties, as is seen in this simpler version. The same protocol that provided these grids is the very same that is used today in the United States of America.

With these examples above, we have seen that fair division is not just something witnessed when a couple of people cannot decide on a slice of cake. There are lots of uses for the ideas that have been developed from the initial fair division scheme. More protocols are being developed to help with situations in the world; as the population increases, so does the number of disagreements. As we have seen though, disagreements are not always a problem; they can be the basis for everyone being much happier than they might be without them.

Conclusion

Fair division is a mathematical problem that has been investigated for over seventy years. The pinnacle of this research was the protocol for envy-free cake division for n people. Many mathematicians have made progress in this area of research. Other theorems have proved useful to allow mathematicians to solve problems, such as proving a fair division of cake is always possible. The concepts that arise in fairly dividing a cake can be applied to several different modern day situations, thus showing this is not just a theoretical problem. There may be even more situations where fair division is useful that have not yet been discovered.

References

Throughout the paper [X, PY] refers to reference number X on page Y.

[1] Lynne Olver, http://www.foodtimeline.org/foodcakes.html, 2000.
[2] Steven J. Brams and Alan D. Taylor, An Envy-Free Cake Division Protocol, 1995.
[3] Henryk Minc, Permanents, 1978.
[4] Hans Schneider, University of Wisconsin – Madison, http://www.math.wisc.edu/~hans/frob.pdf,October 1996.
[5] Sarah Whitehouse, MAS334 Combinatorics, Lecture notes, Autumn Semester 2012-2013, University of Sheffield.
[6] Kenneth Rebman, Ross Hosberger, Mathematical Plums-How to get (at least) a fair share of the cake, 1979.
[7] David Lippman, Math in Society, http://dlippman.imathas.com/mathinsociety/FairDivision1.3.pdf, September 2012
[8] http://www.colorado.edu/education/DMP/fair_division.html, University of Colorado, 1997
[9] Z. Landau, O. Reid, I. Yershov, http://www.sci.ccny.cuny.edu/~landau/papers/redistricting.pdf, March 23, 2006.
[10] Manindra Agrawal, Determinant versus permanent, http://www.icm2006.org/proceedings/Vol_III/contents/ICM_Vol_3_48.pdf, 2006

Appendices

Appendix A – Envy-Free Protocol for Arbitrary n (the $n=4$ version)

This has been taken directly from *An Envy-Free Cake Division Protocol* written by Steven Brams and Alan Taylor. The $n=4$ version has been presented for simplicity but can be extended for arbitrary n.

Step 1 — Player 2 cuts the cake into 4 pieces (that he considers to be the same size), keeps one piece, and hands one piece to each player.

Step 2 — Each of the other three players is asked whether or not he objects to this allocation. (A player objects if and only if he envies some other player.)

Step 3 — If no one objects, then each keeps the piece he was given in Step 1, and we are done.

Step 4 — Otherwise, we choose the smallest i so that Player i objected. For notational simplicity, assume $i=1$. Player 1 now chooses a piece originally given to some other player (whom he envied) and calls that piece A. The piece originally given to Player 1 is called B.

Aside — Once we have A and B, the other two pieces in the allocation from Step 1 are reassembled. That part of the cake will be allocated later. Note that Player 1 thinks A is larger than B. Player 2 thinks A and B are the same size.

Step 5 — Player 1 now names a positive integer $r \geq 10$ (chosen so that, for any partition of A into r sets, Player 1 will prefer A, even with the 7 smallest - according to Player 1 - pieces in the partition of A removed, to B).

Aside — Player 1 can easily choose such an r. That is, the union of the 7 smallest pieces is certainly no larger than 7 times the average size of all r pieces. Hence, Player 1 simply chooses r large enough so that $7\mu(A)/r < \mu(A) - \mu(B)$, where, μ is his measure.

Step 6 — Player 2 now partitions A into exactly r sets (that he considers to be the same size), and does the same to B.

Step 7 — Player 1 chooses (the smallest) 3 sets from the partition of B and names these Z_1, Z_2 and Z_3 He also chooses either (the largest) 3 sets from the partition of A (if he thinks these are all strictly larger than all the Zs), and trims at most 2 of these (to the size of the smallest among the three), or he partitions (the largest) one of the sets in the partition of A into 3 pieces (that he considers to be the same size). In either case, he names these Y_1, Y_2 and Y_3.

Aside — Player 1's strategy in Step 7 guarantees that he will think all three Ys are the same size, and each strictly larger than all three Zs. This is true even if he chooses the second option. Player 2 thinks all three Zs are the same size, and each is at least as large as all three Ys.

Step 8 Player 3 takes the collection of 6 pieces, and either passes (if he thinks there already is at least a 2-way tie for largest), or trims (the largest) one of these (to the size of the next largest), thus creating at least a 2-way tie for largest).

Step 9 Players 4, 3, 2, and 1, in that order, choose a piece from among the 6 Ys and Zs as modified in Step 8 (that they consider to be largest or tied for largest), with Player 3 required to take the piece he trimmed if it is available. Player 2 must choose Z_1, Z_2, or Z_3. Player 1 must choose Y_1, Y_2, or Y_3.

Aside This yields a partition $\{X_1, X_2, X_3, X_4, L_1\}$ of the cake such that $\{X_1, X_2, X_3, X_4\}$ is an envy-free partial allocation, and L_1 is the leftover piece. Moreover, Player 1 thinks his piece X_1 is strictly larger - say by ε - than Player 2's piece X_2.

Step 10 Player 1 names a positive integer s (chosen so that $[4\mu_1(L_1)/5]^5 < \varepsilon$, where μ_1 is Player 1's measure).

Aside The integer s specifies how many times the players will iterate the basic trim-and-choose sequence to follow. Notice that if the rules were instead to allow the iterations to continue until Player 1 said ‘stop’ (which he could strategically do at the point at which he thinks the leftover crumb is smaller than the advantage he has over Player 2), then there is no guarantee that a strategically misguided Player 1 would not keep the game going forever.

Step 11 Player 1 cuts L_1 into 5 pieces (that he considers to be the same size).

Step 12 Player 2 takes the collection of 5 pieces, selects (the largest) 3 pieces, and trims (the largest) 2 or fewer of these (to the size of the smallest, thereby creating at least a 3-way tie for largest).

Step 13 Player 3 takes the collection of 5 pieces, perhaps trimmed in Step 12, selects (the largest) two, and trims, if he wants to, (the largest) one of these (to the size of the smallest, thus creating at least a 2-way tie for largest).

Step 14 Players 4, 3, 2, and 1, in that order, choose a piece (that they consider to be largest or tied for largest), with Players 3 and 2 required to take a piece they trimmed if one is available.

Step 15 Steps 11-14 are repeated $s-1$ more times, with each application of these four steps applied to the leftover piece from the preceding application.

Aside This yields a partition $\{X'_1, X'_2, X'_3, X'_4, L_2\}$ of the cake such that $\{X'_1, X'_2, X'_3, X'_4\}$ is an envy-free partial allocation, and such that Player 1 thinks that X'_1 is larger than $\{X'_2\} \cup \{L_2\}$. We now declare that Player 1 has an irrevocable advantage over Player 2, and we begin creation of a subset of $\{1, 2, 3, 4\} \times \{1, 2, 3, 4\}$, which we call ‘$IA$’ for ‘irrevocable advantage,’ by putting $(1,2) \in IA$.

Step 16 Player 2 cuts L_2 into 12 pieces (that he considers to be the same size).

Step 17 Each of the other players declares himself to be of type A (if he agrees all the pieces are the same size), or type D (if he disagrees). Player 2 is declared to be of type A.

Step 18 If $D \times A \subset IA$, then we give the 12 pieces to the players in A, with each of them receiving the same number of pieces. In this case, we are done.

Step 19 Otherwise, we choose the lexicographically least pair (i,j) from $D \times A$ that is not in IA, and we return to Step 4 with Player i in the role of Player 1, Player j in the role of Player 2, and L_2 in place of the cake.

Step 20 Steps 5-18 are repeated.

Aside Each time we pass through Step 15, we add an ordered pair to IA. Notice that since $D \times A \subset \{1, 2, 3, 4\} \times \{1, 2, 3, 4\}$, and $IA \subset \{1, 2, 3, 4\} \times \{1, 2, 3, 4\}$, we must have $D \times A \subset IA$ after at most 16 iterations. At this point, we conclude at Step 18 with an envy-free division of the entire cake.

This ends the protocol.

Appendix B – Partial Proof of Hall's Marriage Theorem

It is easy to check that if all the women can find husbands, then any r of them of them know at least r men, namely their husbands. This proves that a set of women can always find a husband from the men they know. This implies for each r, any set of the women know at least r men.

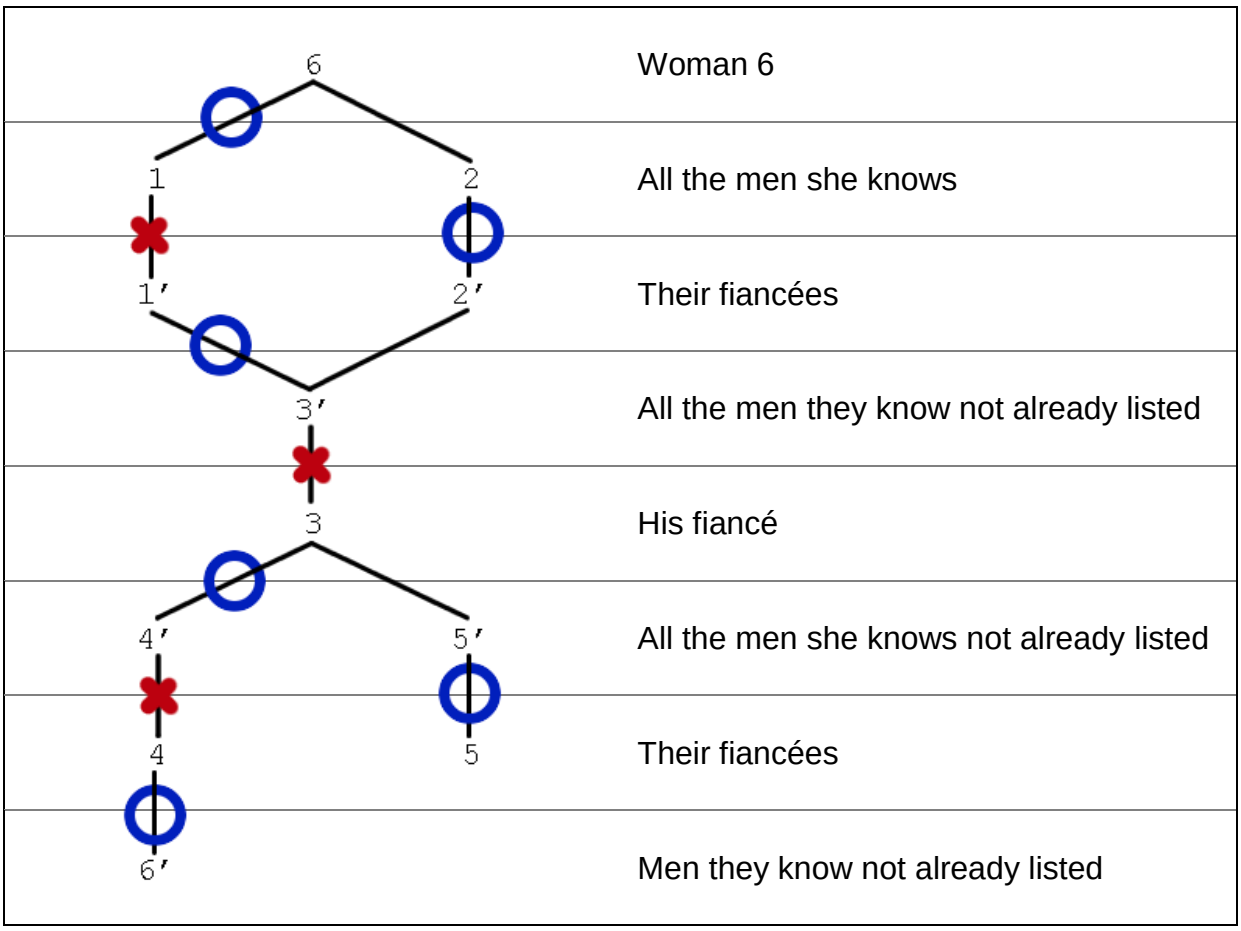
Proving the converse is more difficult and this is where we need to construct a systematic procedure. Take the following example:

Woman	Knows men
1	1′, 3′
2	2′, 3′
3	1′, 3′, 4′, 5′
4	2′, 4′, 6′
5	1′, 5′
6	1′, 2′

The example has been constructed such that it is true that any r of the women know at least r men between them. Our goal is to find husbands for all the women. We accomplish this by the following procedure. Firstly we form engagements as follows:

Woman 1 gets engaged to 1′
Woman 2 gets engaged to 2′
Woman 3 gets engaged to 3′
Woman 4 gets engaged to 4′
Woman 5 gets engaged to 5′

However, woman 6 only knows men who are already engaged. Therefore we have to construct a procedure for turning five couples into six. The following diagram illustrates this:



Where a line with a blue ring around it means a new engagement is made and a line with a red cross on it means an engagement is broken. The lines themselves mean that the men and women know each other.

We now start with $6'$ and work our way back up the diagram. This makes new engagements and breaks old ones as indicated. The new couples are now:

1 and $3'$
2 and $2'$
3 and $4'$
4 and $6'$
5 and $5'$
6 and $1'$.

Now, all 6 women are engaged to men they know and we have a systematic procedure that can be used for larger n.

■

Appendix C – Proof of the König-Egeváry Theorem

Let α be the minimum number of lines to include all the 0s in the matrix and let β be the maximum number of 0s in the matrix all on different lines. We want to prove $\alpha = \beta$.

If there are β 0s with no two of them on the same line, we need at least β lines to cover them all. Therefore, we have the condition $\alpha \geq \beta$. Hence, to complete the proof we need to show that $\alpha \leq \beta$, which then proves $\alpha = \beta$.

To show $\alpha \leq \beta$, we will find α 0s on different lines. Without loss of generality, let $\alpha = p + q$, where the first p rows and the first q columns contain all the 0s. We can rearrange the rows and columns of the matrix so that the relevant ones containing the 0s are in the top and left of the matrix as shown below:

$$M = \begin{bmatrix} & M' & \\ M'' & No & 0s \end{bmatrix} \begin{matrix} p \\ \\ \end{matrix}$$

$$\begin{matrix} \\ q \end{matrix}$$

We will show that M' has a 0 in each row, all in different columns. Exactly the same argument is used to show that M'' has a 0 in each column, all in different rows. Overall this gives $p + q = \alpha$.

Consider M', by the matrix version of Hall's marriage theorem, it is enough to show that any r rows of M' have 0s in at least r columns. We will prove this by contradiction. Suppose this is not true. So r rows of M' have 0s in strictly fewer than r columns, say in s columns with $s < r$. However, we then could have covered all the 0s choosing the s columns in place of the r rows and we would have covered all the 0s with fewer than α lines, contradicting the definition of α.

■

Appendix D – Script Source Code

This is revision 8 of the script's source code and is best viewed in colour as the syntax has been highlighted. At the time of publication, this revision was the most recent. Check *http://rowanparker.com/cake/source.php* for updates.

```
<?php
/*
Fair Cake Division Script (r8 : 21/11/12)

Copyright (c) 2012 Rowan Parker (rowanparker.uk@gmail.com / rowanparker.com)

Permission is hereby granted, free of charge, to any person obtaining a copy of this s
oftware and associated documentation files (the "Software"), to deal in the Software w
ithout restriction, including without limitation the rights to use, copy, modify, merg
e, publish, distribute, sublicense, and/or sell copies of the Software, and to permit
persons to whom the Software is furnished to do so, subject to the following condition
s:

The above copyright notice and this permission notice shall be included in all copies
or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN
CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
 PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LI
ABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
 OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR O
THER DEALINGS IN THE SOFTWARE.
*/
?>
<?php
$q = explode('&', $_SERVER['QUERY_STRING']); $q = $q[0];
if (empty($q)) { header("Location: ?first"); exit(); } else { $page = $q; }
function isSquareMatrix($matrix) {
    if (!is_array($matrix)) { return false; }
    $r = count($matrix);
    foreach ($matrix as $row) {
        if (!is_array($row)) { return false; }
        if (count($row) != $r) { return false; }
        foreach ($row as $item) {
            if (!is_numeric($item) && !is_numeric(substr($item, 1))) { return false; }
        }
    }
    return true;
}
function displayMatrix($matrix, $locked=true) {
    if (!isSquareMatrix($matrix)) { return false; }
    echo "<br><br><form name=\"postmatrix\" method=\"post\" action=\"?third\"><table c
ellpadding=\"0\" cellspacing=\"0\" border=\"0\"><tr><td></td><td></td>";
    $ilabs = array_keys($matrix); $jlabs = array_keys(current($matrix));
    $c = count($matrix); $i = 1;
    for ($ti=1;$ti<=$c;$ti++) { echo "<td align=\"center\"><samp>S<sub>" . ($jlabs[$ti
-1]+1) . "</sub></samp></td>"; }
    $colours = arrayHexColours();
    foreach ($matrix as $k1 => $row) {
        echo "<tr><td><samp>A<sub>" . ($ilabs[$i-1]+1) . "</sub></samp></td>";
```

```
        if ($i == 1) {
            echo "<td style=\"background-image:url('bracket_topleft.png');background-
repeat:no-repeat;width:40px;height:40px;\"></td>";
        } elseif ($i == $c) {
            echo "<td style=\"background-
image:url('bracket_bottomleft.png');background-repeat:no-
repeat;width:40px;height:40px;\"></td>";
        } else {
            echo "<td style=\"background-image:url('bracket_left.png');background-
repeat:repeat-y;width:40px;height:40px;\"></td>";
        }
        foreach ($row as $k2 => $item) {
            if ($locked) {
                if ($item{0} == 'z') {
                    echo "<td align=\"center\"><input value=\"" . substr($item,1) . "\
" size=\"1\" disabled=\"true\" style=\"color:#000000;background-
color:#08ff00;\" /></td>";
                } elseif (array_key_exists($item{0}, $colours)) {
                    echo "<td align=\"center\"><input value=\"" . substr($item,1) . "\
" size=\"1\" disabled=\"true\" style=\"color:#000000;background-
color:#" . $colours[$item{0}] . "\" /></td>";
                } else {
                    echo "<td align=\"center\"><input value=\"$item\" size=\"1\" disab
led=\"true\" style=\"color:#000000;\" /></td>";
                }
            } else {
                echo "<td align=\"center\"><input value=\"$item\" size=\"1\" name=\"po
stmatrix[$k1][$k2]\" onFocus=\"if (this.value==this.defaultValue) this.value = ''\" on
Blur=\"if (this.value=='') this.value = this.defaultValue\" autocomplete=\"off\" /></t
d>";
            }
        }
        if ($i == 1) {
            echo "<td style=\"background-image:url('bracket_topright.png');background-
repeat:no-repeat;width:40px;height:40px;\"></td></tr>";
        } elseif ($i == $c) {
            echo "<td style=\"background-
image:url('bracket_bottomright.png');background-repeat:no-
repeat;width:40px;height:40px;\"></td></tr>";
        } else {
            echo "<td style=\"background-image:url('bracket_right.png');background-
repeat:repeat-y;width:40px;height:40px;\"></td></tr>";
        }
        $i++;
    }
    echo "</table>";
    if (!$locked) { echo "<input type=\"submit\" value=\"Next ->\" />"; }
    echo "</form>";
}
function arrayHexColours() { //Since n < 11, max of 3 colours. More can be added ('z'
is reserved)
    return Array('a' => 'fbff21', 'b' => '12cbff', 'c' => 'f94db3');
}
function generateNullMatrix($n=3) {
    if (!is_int($n) || $n<3) { return false; }
    for ($i1=0;$i1<$n;$i1++) {
        for ($i2=0;$i2<$n;$i2++) {
            $return["$i1"]["$i2"] = 0;
```

```
        }
    }
    return $return;
}
function populateFirstRowMatrix($matrix) {
    if (!isSquareMatrix($matrix)) { return false; }
    foreach ($matrix[0] as &$item) {
        $item = 1;
    }
    return $matrix;
}
function parsePostMatrix() {
    $matrix = @$_REQUEST['postmatrix'];
    if (!is_array($matrix)) { return false; }
    foreach ($matrix as &$row) {
        if (is_array($row)) {
            foreach ($row as &$item) {
                if ($item==1) { $item = 1; } else { $item = 0; }
            }
        }
    }
    return $matrix;
}
function NOTvalidateMatrix($matrix) { //false is good
    if (!isSquareMatrix($matrix)) { return 1; }
    foreach ($matrix as $r => &$row) {
        foreach ($row as &$item) {
            if ($r==0) {
                if ($item!=1) { return 2; }
            } else {
                if ($item!=0 && $item!=1) { return 3; }
            }
        }
        if (array_sum($row)<1) { return 4; }
    }
    return false;
}
function permanentMatrix($matrix) {
    if (!isSquareMatrix($matrix)) {
        return false;
    } else {
        return permanentMatrix_do($matrix);
    }
}
function permanentMatrix_do($matrix) { //no error checking to save on resources
    $n = count($matrix); $ans = 0;
    if ($n==1) {
        return $matrix[0][0];
    } elseif ($n==2) {
        return (($matrix[0][0]*$matrix[1][1])+($matrix[0][1]*$matrix[1][0]));
    } else {
        for ($i=0;$i<$n;$i++) {
            $ans = $ans + $matrix[0][$i]*permanentMatrix_do(getSubMatrix($matrix, 0, $
i));
        }
    }
    return $ans;
}
```

```
function getSubMatrix($matrix, $i, $j, $maintain=false) { //no error checking to save
on resources
    $sr=0; $sc=0;
    foreach ($matrix as $r => $row) {
        if ($r!=$i) {
            $c=0; $sc=0;
            foreach ($row as $c => $item) {
                if ($c!=$j) {
                    if ($maintain) {
                        $submatrix[$r][$c] = $item;
                    } else {
                        $submatrix[$sr][$sc] = $item;
                    }
                    $sc++;
                }
            }
            $sr++;
        }
    }
    return $submatrix;
}
function nonzeroCase($matrix) {
    if (NOTvalidateMatrix($matrix)) { return false; }
    $map = Array(); //map: person |-> slice
    $map = nonzeroCase_loop($matrix, $map);
    ksort($map);
    return $map;
}
function nonzeroCase_loop($matrix, $map) { //no error checking to save on resources
    foreach ($matrix as $key => $row) {
        $order[$key] = array_sum($row);
    }
    $colsum = sumSum($matrix,3); $colsum = current($colsum);
    asort($order, SORT_NUMERIC); reset($order);
    $person = key($order);
    asort($colsum, SORT_NUMERIC); reset($colsum);
    foreach ($colsum as $slice => $value) {
        if ($matrix[$person][$slice] == 1) {
            $i = $person; $j = $slice;
            break;
        }
    }
    $a = Array($i => $j);
    $map = $map + $a;
    $c = count($matrix);
    if ($c>1) {
        $submatrix = getSubMatrix($matrix, $i, $j, true);
        $map = nonzeroCase_loop($submatrix, $map);
    }
    return $map;
}
function displayMap($map) {
    if (is_array($map)) {
        ksort($map);
        echo "<br><br><table style=\"border-
collapse:collapse;\"><tr><td style=\"border: 1px solid black; border-
style: none solid solid none; padding: 5px 20px 5px 5px; float:right;\">Person</td>";
        foreach ($map as $person => $slice) {
```

```
            echo "<td style=\"border: 1px solid black; border-style: ";
            if ($person==0) { echo "none none solid none"; } else { echo "none none so
lid solid"; }
            echo "; padding: 5px 20px;\"><samp>A<sub>"; echo $person+1; echo "</sub></
samp></td>";
        }
        echo "</tr><tr><td style=\"border: 1px solid black; border-
style: none solid none none; padding: 5px 20px 5px 5px; float:right;\">Slice</td>";
        foreach ($map as $person => $slice) {
            echo "<td style=\"border: 1px solid black; border-style: ";
            if ($person==0) { echo "none none none none"; } else { echo "none none non
e solid"; }
            echo "; padding: 5px 20px;\"><samp>S<sub>"; echo $slice+1; echo "</sub></s
amp></td>";
        }
        echo "</tr></table>";
        return true;
    } else {
        echo "Error with map.<br>";
        return false;
    }
}
function sumSum($matrix, $v) {
    if (!isSquareMatrix($matrix)) { return false; }
    $multiplier = count($matrix);
    foreach ($matrix as $i => $row) {
        $rowsum[$i] = array_sum($row);
        foreach ($row as $j => $item) {
            $colsum[$j] = @$colsum[$j] + $item;
        }
    }
    $summed = false;
    foreach ($matrix as $i => $row) {
        foreach ($row as $j => $item) {
            if ($v==1) {
                $summed[$i][$j] = $item/$rowsum[$i];
            } elseif ($v==2) {
                $summed[$i][$j] = round((($rowsum[$i]+$colsum[$j])/$rowsum[$i])+($item
*$multiplier),3);
            } elseif ($v==3) {
                $summed[$i][$j] = $colsum[$j];
            }
        }
    }
    return $summed;
}
function resetMatrixKeys($matrix) {
    foreach ($matrix as &$row) {
        $row = array_values($row);
    }
    return array_values($matrix);
}
function zeroCase($matrix) {
    if (NOTvalidateMatrix($matrix)) { return false; }
    $map = Array();
    return zeroCase_loop($matrix, $map);
}
function zeroCase_loop($matrix, $map) { //no error checking to save on resources
```

```
    $summed = sumSum(sumSum($matrix,1),2); if (!$summed) { return false; }
    foreach ($summed as $i => $row) {
        $max[$i] = max($row);
    }
    arsort($max); $taken = Array(); $top = current($max);
    foreach ($max as $key => $item) {
        if ($item == $top) {
            $nmax[$key] = $item;
        }
    }
    $max = $nmax;
    foreach ($max as $i => $num) {
        $row = $summed[$i];
        $pos = array_keys($row, $num);
        $done = false;
        foreach ($pos as $item) {
            if (!in_array($item, $taken)) {
                $sharemap[$i] = $item;
                $taken[] = $item;
                $done = true;
                break;
            }
        }
        if (!$done) {
            asort($row);
            foreach ($row as $j => $item) {
                if (!in_array($j, $taken)) {
                    $sharemap[$i] = $j;
                    $taken[] = $j;
                    break;
                }
            }
        }
    }
    $submatrix = $matrix;
    foreach ($sharemap as $i => $j) {
        $submatrix = getSubMatrix($submatrix, $i, $j, true);
    }
    $map[] = $sharemap;
    $permanent = permanentMatrix(resetMatrixKeys($submatrix));
    if ($permanent==0) {
        $map = zeroCase_loop($submatrix, $map);
    } else {
        $ilabs = array_keys($submatrix); reset($submatrix); $jlabs = array_keys(curren
t($submatrix));
        $tmap = nonzeroCase(resetMatrixKeys($submatrix));
        foreach ($tmap as $i => $j) {
            $smap[$ilabs[$i]] = $jlabs[$j];
        }
        $map[-1] = $smap;
    }
    return $map;
}
function highlightMatrix($matrix, $map, $with='z') {
    if (!is_array($map)) { return false; }
    if (!isSquareMatrix($matrix)) { return false; }
    foreach ($map as $person => $slice) {
        $matrix[$person][$slice] = $with . $matrix[$person][$slice];
```

```
    }
    return $matrix;
}
function doPage($p) {
    if ($p=='first') {
        echo "Please enter the number of people you wish to share the cake between: <f
orm action=\"?second\" method=\"post\"><input name=\"n\" size=\"1\" value=\"\" autocom
plete=\"off\" /> <input type=\"submit\" value=\"Next ->\" /></form>";
    } elseif ($p=='second') {
        $n = (int) @$_REQUEST['n'];
        if ($n==1) {
            echo "The cake is all yours! Click <a href=\"?first\">here</a> to try agai
n.";
        } elseif ($n==2) {
            echo "This is the 'You cut, I choose' principle. Click <a href=\"?first\">
here</a> to try again.";
        } elseif ($n>10) {
            echo "You have chosen to share the cake with <samp>$n</samp> people. This
script is limited to <samp>10</samp> or fewer people to save on server resources. Clic
k <a href=\"?first\">here</a> to go back.";
        } elseif ($n>2) {
            echo "You have chosen to share the cake between <samp>$n</samp> people. We
 are defining <samp>A<sub>i</sub></samp> and <samp>S<sub>j</sub></samp> to be the <sam
p>i<sup>th</sup></samp> person and <samp>j<sup>th</sup></samp> slice for some <samp>i,
j &#8712; &#8484;<sup>+</sup></samp>. The person cutting the cake should be randomly d
ecided and assigned <samp>A<sub>1</sub></samp>, the other people can then be assigned
in a way of your choice. Please enter a <samp>1</samp> in the box corresponding to <sa
mp>A<sub>i</sub></samp> and <samp>S<sub>j</sub></samp> if the <samp>i<sup>th</sup></sa
mp> person deems the <samp>j<sup>th</sup></samp> slice as acceptable and a <samp>0</sa
mp> otherwise. Any entries other than <samp>0</samp> or <samp>1</samp> will be convert
ed to <samp>0</samp>. Please fill out the matrix below in the manner described above:"
;
            displayMatrix(populateFirstRowMatrix(generateNullMatrix($n)), false, "thir
d");
        } elseif (!isset($_REQUEST['n'])) {
            echo "Error. Click <a href=\"?first\">here</a> to restart.";
        } else {
            echo "Please try again and enter a positive integer. Click <a href=\"?firs
t\">here</a> to try again.";
        }
    } elseif ($p=='third') {
        $matrix = parsePostMatrix(); $n = @count($matrix);
        if (!isSquareMatrix($matrix)) {
            if ($n>2 && $n<8) {
                echo "No sqaure matrix was found in the post data. Click <a href=\"?se
cond&n=$n\">here</a> to try again.";
            } else {
                echo "Error. Click <a href=\"?first\">here</a> to restart.";
            }
        } else {
            $valid = NOTvalidateMatrix($matrix);
            if (!$valid) {
                $a = permanentMatrix($matrix);
                if ($a!=0) {
                    $map = nonzeroCase($matrix);
                    echo "You entered the following matrix:"; displayMatrix(highlightM
atrix($matrix, $map), true);
                    echo "<br>As the permanent of this matrix is <samp>$a</samp>, ther
```

```
e is a simple fair solution. The output of which is highlighted in the above matrix an
d displayed in the table below:";
                    displayMap($map);
                    echo "<br><br>Click <a href=\"?first\">here</a> to start again.";
                } else {
                    echo "You entered the following matrix:";
                    $sharemap = zeroCase($matrix); $tmatrix = $matrix; $colours = arra
yHexColours(); $keys = array_keys($colours);
                    if (count($keys) < (count($sharemap)-1)) {
                        foreach($sharemap as $key => $map) {
                            if ($key<0) {
                                $tmatrix = highlightMatrix($tmatrix, $map, 'z');
                            } else {
                                $tmatrix = highlightMatrix($tmatrix, $map, 'a');
                            }
                        }
                    } else {
                        $i=0;
                        foreach ($sharemap as $key => $map) {
                            if ($key<0) {
                                $tmatrix = highlightMatrix($tmatrix, $map, 'z');
                            } else {
                                $tmatrix = highlightMatrix($tmatrix, $map, $keys[$i]);
                            }
                            $i++;
                        }
                    }
                    displayMatrix($tmatrix);
                    echo "<br>As the permanent of this matrix is <samp>0</samp>, there
 isn't a simple fair solution. However, there is still a fair method of sharing which
is explained here.<br><br>";
                    asort($sharemap, SORT_NUMERIC);
                    foreach ($sharemap as $key => $map) {
                        if ($key<0) {
                            echo "Firstly, give out the following slice(s) to people a
s displayed in the table below (also <span style=\"background-
color:#08ff00;padding:2px 6px;\">highlighted</span> above):";
                            displayMap($sharemap[-1]);
                            echo "<br><br>";
                        } else {
                            echo "Now, recombine the following slices to re-
share between the people displayed in the table below (also <span style=\"background-
color:#{$colours[$keys[$key]]};padding:2px 6px;\">highlighted</span> above):";
                            displayMap($sharemap[$key]);
                            echo "<br><br>";
                        }
                    }
                    echo "Click <a href=\"?first\">here</a> to start again.";
                }
            } else {
                echo "The matrix you entered wasn't valid. This is because ";
                if ($valid==1) { echo "it wasn't square (don't really get how you mana
ged that one...)."; }
                elseif ($valid==2) { echo "the top row didn't contain all <samp>1</sam
p>."; }
                elseif ($valid==3) { echo "some elements of the matrix weren't <samp>0
</samp> or <samp>1</samp>."; }
                elseif ($valid==4) { echo "the sum of at least one row wasn't <samp>1<
```

```
/samp> or greater (any non <samp>0</samp> or <samp>1</samp> entries are converted to <
samp>0</samp>)."; }
                echo "<br /><br />Please try again:";
                displayMatrix($matrix, false, "third");
            }
        }
    } else {
        echo "Page '$p' not found. Click <a href=\"?first\">here</a> to restart.";
    }
}
?>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<link rel="shortcut icon" href="favicon.ico" />
<title>Fair Cake Division Script</title>
<style type="text/css">
html {
    font-family: Arial;
}
</style>
</head>
<body>
<img src="cake.png" /><h1 style="display:inline;">Fair Cake Division Script</h1> <h3 s
tyle="display:inline;"><sup>(<?php $v = @trim(file_get_contents("version.txt")); echo
$v; //version.txt contains version info ?>)</sup></h3> ~ click <a href="?first"><small
>HERE</small></a> to restart and click <a href="source.php" target="_blank"><small>HER
E</small></a> for source code<br><br>
Written by <a href="http://rowanparker.com/" target="_blank">Rowan Parker</a> as part
of a mathematics project for <a href="http://sheffield.ac.uk/" target="_blank">The Uni
versity of Sheffield</a>. The project was entitled <i>How to Fairly Cut a Cake</i>. Th
e final essay will be linked <a>here</a> at a later date. This script has been tested
in Firefox, Chrome and the Android browser but should function in other browsers as we
ll (turn JavaScript off if you have any issues).<br><br>
For the purposes of this script, we are defining <samp>A<sub>i</sub></samp> and <samp>
S<sub>j</sub></samp> to be the <samp>i<sup>th</sup></samp> person and <samp>j<sup>th</
sup></samp> slice for some <samp>i,j &#8712; &#8484;<sup>+</sup></samp>.
<hr><br>
<?php
dopage($page);
?>
</body>
</html>
```