DIGITISING PICTURES

The Square Kilometre Array is going to collect mind-boggling amounts of information. Information needs to be stored and displayed accurately to be truly useful.

One of the most amazing things that the *SKA* will have to deal with is the sheer amount of information it will generate. Unlike optical telescopes, radio telescopes don't actually record images. They record the strength of signals within a given stretch of the electromagnetic spectrum. The strength of that signal is converted to a string of numbers. That string is what gets stored, and used to make pictures.

This system is also how a digital camera works- though digital cameras detect visible light instead of radio waves.

This activity is going to explore the idea of representing pictures (in this case *text* first) as pixels- small squares that make up a picture. The colour and position of that pixel is determined by the string of numbers.

To keep it relatively simple, in this activity, we will use 1 to mean "fill the square" (you can either fill it completely, or mark it ☒) and 0 to mean "leave it empty". With that rule established, we can read strings, or create them.

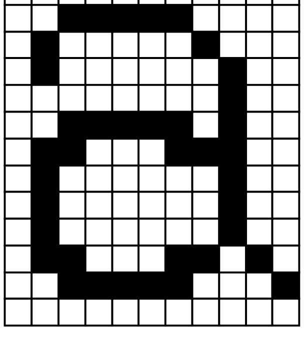
What to do

Creating strings of data

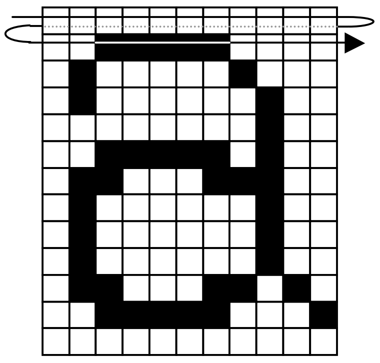
In this activity we are going to explore making our own pictures. As with the *Reading Strings* activity, we can start with text- creating your own *font*. After doing this, you may appreciate how labour intensive making fonts can be. Fortunately digital fonts aren't usually made this way any more. Instead, people use formulae to describe the shape of the letter.

What you need:

- Square paper
- Pencil
- Ruler



1. Start by setting the boundary of the letter you're going to make. If you make the boundary a prime number, it makes decoding it easier. The larger the grid, the smoother your letter will look, but the more information it will take. In this example, the sides are 11×13 = 143 pixels to make one letter!
2. Choose a letter, and draw it on your grid. In this example, choosing the lower case 'a' shows that even 143 pixels isn't many when you're trying to make curves smooth…

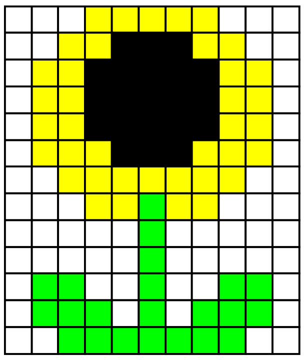


3. Starting at the top left corner, start recording what information is there. Use 1 for “filled” and 0 for “empty”. In this example, the string will start “0”. Once you’ve filled in the state of a pixel, look at the pixel immediately to its right. Record that pixel’s state and keep going.
4. When you reach the end of a row, continue at the leftmost column in the next row down. Accordingly, this string is going to start 00000000000001111…

Adding Colour

Adding more information to your picture means adding more information to the string.

In this example, we’ve got 4 possible states for a pixel to be:



- White (which we can represent by 0);
- Black (represented by 1);
- Green (represented by 2);
- Yellow (represented by 3).

Doing the same process detailed above, we can now have a string that starts 000333330000033111330003311….

The more colours that you use, the more information needs to go into the string. With this manual system, we can happily use the numbers 0-9 to represent ten different colours.

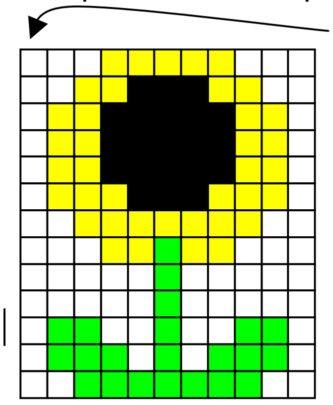
If we want to represent more than ten colours, we either need to use:

- preceding zeros- for example, 00, 01, 02, 03 which will double the length of the string;
- letters to represent colours. This is great, but only gets us to 36 colours.

A “CompuServe Graphic Interchange Format” file or .gif file uses a maximum of 256 colours for each picture. Accordingly, each pixel is labelled with a number from 0-255 (in binary, from 00000000 to 011111111). That’s why .gif files are so small, but also means they have as good picture quality as other files types.

Compressing the information

Having 133 bits of information describing such a small picture is a good start, but there are more efficient ways of storing this data. A simple way of doing this is instead of using one number per pixel, you can use pairs of numbers to describe how the picture is built up.



Starting at the top left, and moving one pixel at a time left to right, you can see that you get large chunks of repeating numbers.

Expanding on this idea, you could use the first number in the pair of numbers to describe the colour, and the second number to describe how long that number repeats.

In this example, the first pair of numbers would be 03, meaning "It's colour 0 for the next 3 squares".

The string for this picture would be 033505321332033215320232153202321532023313330337053221320821090121062202210222022301210123032702, which is only 96 'bits' of information…

This gives an excellent opportunity to look at ratios, by considering the unfortunate phrase *data compression ratios*.

We have a string of 96 numbers represent an uncompressed string of 143 numbers.

As a ratio, that works out to be 96:143, which doesn't present well. Alternative ways of representing this ratio, however, show that we've *data compression* of 96 ÷ 143 ≈ 0.671, or roughly 67%. You can also consider the space saving, by using:

1 – *data compression.*

There are pictures that will actually increase the size of the string if you use the compression technique listed above. For example, for the first 8 squares of a chessboard pattern, the 'raw' data from the first step would give a string length of 8: 01010101. Using this compression technique would give a string length of 16: 0111011101110111.

Investigation

Perhaps you can devise a better compression system? It's one of the goals that mathematicians and computer scientists strive for.

If there are only 4 possible colours for a pixel, how many combinations are there of two side by side pixels?

{00,01,02,03,10,11,12,13,14,20,21,22,23,30,31,32,33} = 4×4 = 16.

How could you represent those 16 different combinations?