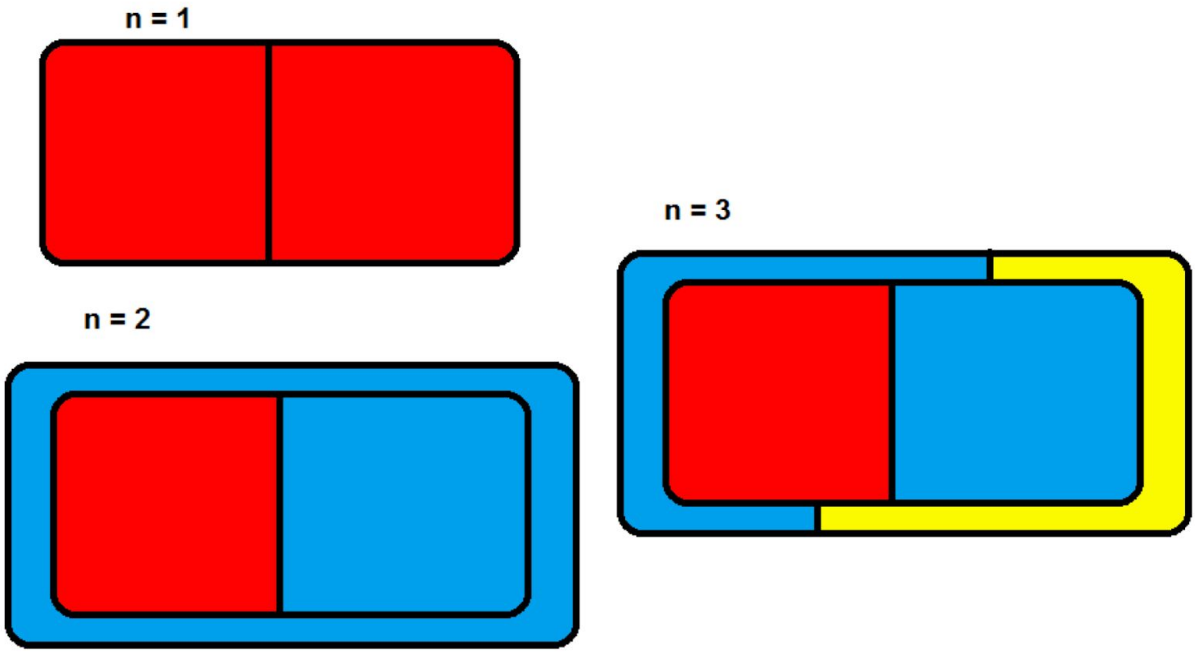
Declan Bartlett, Adam Elder, Josh Tell, Robert Young
COSC326
Etude 10 - Colouring Maps
9/4/17

Colouring Maps

This etude was a practical application of the 4-colour theorem, which was the first major theorem to be proved using a computer. It states that, ”given any separation of a plane into contiguous regions, producing a figure called a *map*, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same colour”.

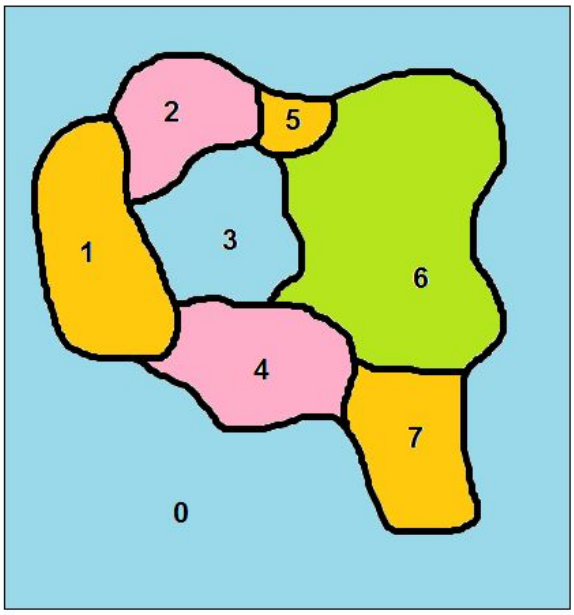


As shown by the above images, if 1 colour is used then the problem cannot be solved. Obviously because if there is more than 1 region then there will be a clash. Our images also demonstrate that 2 colours would also not work. 3-colours can work for simpler maps, and if the water did not count, then New Zealand could be coloured in only 3 colours. However, if one region is surrounded by an odd number of other regions that touch each other, 4 colours are required. An example of this is shown in our n = 3 image, as each region is surrounded by 3 other regions that are all touching each other.

Our Program

Our map-colouring program, ‘ColouringMaps.java’, takes a line of input for each country in the map to be coloured. Each line consists of a sequence of numbers.The first number on each line represents the country to dedicate the line to and the following numbers represent this

country's neighbouring territories. The ordering of the numbers following the first one doesn't matter. For instance, a country with no neighbours would just feature one number in its input line. A sample colouring of a make-believe island is displayed below:



INPUT

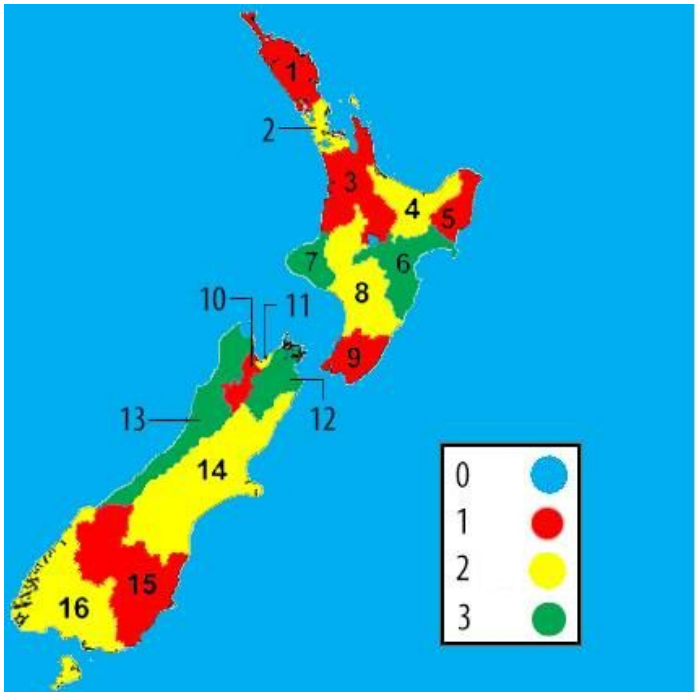
```
0 1 2 5 6 7 4
1 0 2 3 4
2 0 1 3 5
3 1 2 5 6 4
4 0 1 3 6 7
5 0 2 3 6
6 0 5 3 4 7
7 0 4 6
```

Looking at the eighth line of the sample input, we can see that by writing '7 0 4 6', we mean 'This is country 7. It neighbours countries 0, 4, and 6, so it can't be the same colour as any of those'. The input lines for countries 0, 4, and 6 also feature 7 in their neighbouring territories. Country 3 was unfortunately coloured the same as the surrounding ocean (Country 0), but if we wanted to prevent this we could feature 0 in country 3's input line and vice versa. However, this isn't guaranteed to work with four colours in all cases, as 0 and 3 aren't actually neighbours so the algorithm could be thrown off in some cases.

Our program takes a recursive approach to colouring the map. Starting with country 0, a temporary colour is picked that doesn't match a neighbouring country's colours. If a colour could be found, the program would move on to the next country in the input. If a colour couldn't be found, the program would move back a step and find a new colour for the previous country in the input before continuing onward.

Upon successfully colouring the last country in the input, the program would output a list of all of the colour assignments in the form 'Region x: c' where x is the country's number and c is a numerical representation of the colour between 0 and 3. If the program fails to colour the first country in the input after exhausting all possible options, it will print "NO SOLUTION" to the screen. However, the only time this should ever happen would be for invalid inputs, as explained by the first section of this report.

Colouring New Zealand



INPUT

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 0 2
2 0 1 3
3 0 2 4 6 7 8
4 0 3 5 6
5 0 4 6
6 0 4 5 3 8
7 0 3 8
8 0 3 6 7 9
9 0 8
10 0 11 12 13 14
11 0 10 12
12 0 10 11 14
13 0 10 14 15 16
14 0 12 10 13 15
15 0 13 14 16
16 0 15 13
```