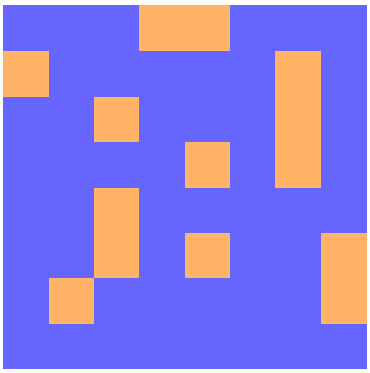
COSC201 Assignment 1: Counting the seas

Due: 11:59 p.m. Friday, April 1, 2022

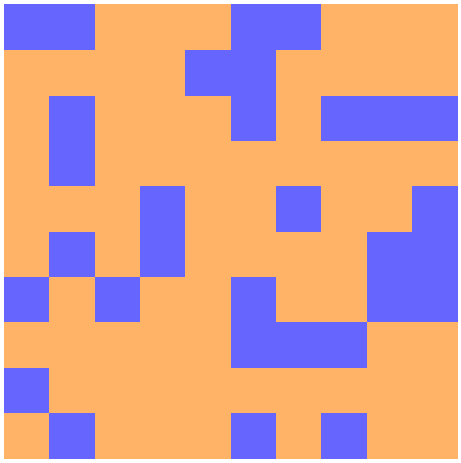
Introduction

Imagine a square world consisting of cells each of which is either land or water. A computational cartographer's paradise if you will. Anyhow, it turns out that a number of such worlds exist of varying sizes and varying proportions of land to water. Your continuing mission has been to search out these strange worlds and count how many distinct bodies of water there are on each of them.

Here's a world that you discovered recently – its dimensions are 8×8 and it has only 14 land cells.



Obviously, in this world there is only one giant sea. On the other hand, another world with a 10×10 grid was also recently discovered:



This world is dominated by land, and there are 10 seas. You may disagree, and think there are 15 seas, but water cells are considered to be part of the same sea if they meet along an edge **or** at a corner.

Exploration is a complicated and expensive business and the thoughts of your superiors have turned to simulation – that is, to creating imaginary worlds of this type in order to try and understand how the number of seas changes depending on the size of the underlying grid and the ratio of water to land.

Raw materials

You will be provided with a number of Java classes. These include:

- All the union-find implementations we've looked at.
- A class `Map.java` that represents maps of the type we're interested in. This class includes a number of convenience methods for various tasks. For instance:
  - indexing the cells of a map from 0 in the top left corner and then increasing from left to right and top to bottom (so, in the 10×10 grid the first row contains cells 0 to 9, the second 10 to 19, the third 20 to 29, and so on);
  - recognising the type of each cell (by index or coordinates);
  - identifying all the neighbours of a given cell (by index).

Using these classes, you are asked to produce some more code and to conduct and report on some experiments.

Code submission (5 points)

The code you submit for this assignment will be a single class called `MapAnalyser`. Skeleton code for that class will be provided and its Javadoc will specify the requirements in detail. However, these and the required underlying algorithms are also discussed briefly below.

`MapAnalyser(Map m)`

The constructor for a `MapAnalyser` requires a `Map` instance. All of its methods refer to this underlying `Map`.

`countSeas()`

This method returns an integer which is the total number of seas of the underlying `Map`. To compute this number you must be able to identify the seas, i.e., which water cells belong to the same seas. This is exactly the kind of task that the union-find framework is designed for.

Specifically, given a map you can initialise a `UnionFind` instance whose size is the total number of cells in the map. Then, you can iterate once over the cells of the map – whenever you find a water cell, you can do a union operation between it and its water neighbours. At the end of this process, two water cells belong to the same sea if and only if they are in same set of the partition (i.e., if and only if they have the same `find` value).

There are a number of options about how to use this idea to count seas (you can do it as you construct the final state of the union-find instance, or after you have done so). The choice is up to you.

`seaSize(int r, int c)`

This method just returns the total number of cells belonging to the same sea as the cell in row `r` and column `c`. If this cell is land rather than water it should just return the value 0.

Written submission (5 points)

Please submit the answers to the following questions as a single **PDF** document with filename of the form `314159Pi.pdf` where `314159` is replaced by your ID number and `Pi` is replaced by your surname. There are no formal requirements for the format of your submission, but presentation, spelling and grammar are all important elements which will account for roughly 40% of the marks for this part of the assignment.

The written answers to questions one and two could be a couple of paragraphs each (or a bit longer) along with supporting data. A single paragraph (and possibly some pseudocode) is enough to answer question three.

1. The efficiency of the various union-find instances affects how large a square map can be analysed. A reasonable length of time for a single computation might be something on the order of a second. Conduct experiments using `UF1`, `UF2`, `UF3`, and `UF4` to determine rough values for that limit on the hardware you are using when the probability of a cell being water is 0.5. Describe those experiments (including the number of repetitions) and report on their results (a table or chart would be appropriate). Comment briefly on whether the outcomes match our theoretical discussions of the four different algorithms and how changing the water probability might affect their performance.
2. Consider 10×10, 100×100, and 1000×1000 maps. Conduct experiments to try and determine the proportion of water needed on each so that on average there are two or fewer seas. Describe those experiments (including the number of repetitions - this should be large enough that you have confidence in the results you're claiming) and report on their results.
3. Suppose that we also wanted to count the islands. How would you do that - and what changes to the code might be needed?