Functional and Logic Programming

Home Assignment 7 – Haskell Coding Interview Question

Due: Monday, 22.6.2019 - 23:55

Instructions

- Please create a source file called ***hw7.hs*** and put all the answers there.
- This submission is in pairs.
- The file should start with a comment which contains both of your **full names** (in English) and **IDs.**

```
-- Jimi Hendrix
-- 654321987

-- Eric Clapton
-- 654321988
```

- Make sure the file **is valid** by loading it into GHCi. A valid file will load without any errors or warnings.
- If you need a function but you don't know how to implement it - just write it's signature (name and type) and put `undefined` in the function's body.
  That way you'll be able to load the file even though it contains references to undefined names.
- When writing a function - write both the **type** and the **body** of the function.
- Be sure to write functions with **exactly the specified name** (and **type signature** - if it is provided) for each exercise.
  You may create additional auxiliary/helper functions with whatever names and type signatures you wish.
- Try to write **small functions** which perform just **a single task**, and then **combine** them to create more complex functions.

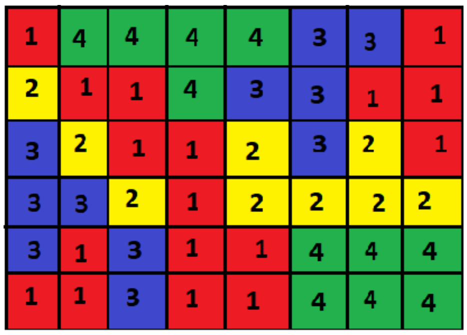
Exercise

In this assignment we will solve together a common coding interview question using Haskell. We will find the **Largest connected component on a grid.**

Given a grid with different colors in different cells, each color is represented by a different number. The task is to find the largest connected component on the grid.
A Largest component grid refers to a maximum set of cells such that you can move from any cell to any other cell in this set by only moving between side-adjacent cells in the grid. (Thus by only using Up, Down, Right, Left moves.)

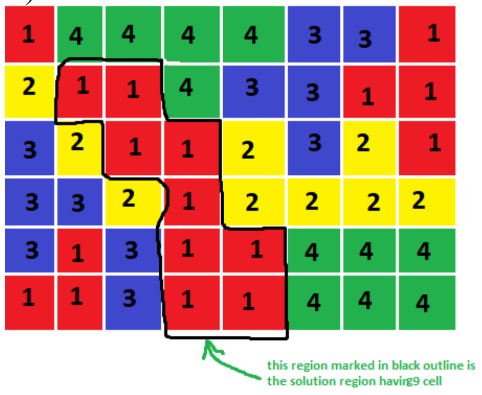
<u>Example:</u>

Input



1	4	4	4	4	3	3	1
2	1	1	4	3	3	1	1
3	2	1	1	2	3	2	1
3	3	2	1	2	2	2	2
3	1	3	1	1	4	4	4
1	1	3	1	1	4	4	4

Output (The circled CC)



1	4	4	4	4	3	3	1
2	1	1	4	3	3	1	1
3	2	1	1	2	3	2	1
3	3	2	1	2	2	2	2
3	1	3	1	1	4	4	4
1	1	3	1	1	4	4	4

this region marked in black outline is
the solution region having 9 cell

Haskell representation:

The grid representation will be:

```haskell
type Color = Int
type Index = (Int,Int)
type Cells = [(Index,Color)]
```

So the example input will be represented by:

```haskell
example = [((0,0),1),((0,1),4),((0,2),4),((0,3),4),((0,4),4),((0,5),3),((0,6),3),((0,7),1),
           ((1,0),2),((1,1),1),((1,2),1),((1,3),4),((1,4),3),((1,5),3),((1,6),1),((1,7),1),
           ((2,0),3),((2,1),2),((2,2),1),((2,3),1),((2,4),2),((2,5),3),((2,6),2),((2,7),1),
           ((3,0),3),((3,1),3),((3,2),2),((3,3),1),((3,4),2),((3,5),2),((3,6),2),((3,7),2),
           ((4,0),3),((4,1),1),((4,2),3),((4,3),1),((4,4),1),((4,5),4),((4,6),4),((4,7),4),
           ((5,0),1),((5,1),1),((5,2),3),((5,3),1),((5,4),1),((5,5),4),((5,6),4),((5,7),4)]
```

You can assume that you will be tested on grids of size 6x8 (the size of the example grid). You can assume that you will be tested only with inputs that have only one LCC.
There is no limitation for the number of colors.

Implement a function:

`getLCC ::` **Cells -> [Index]**

which receives a grid an returns a list of indexes of the LCC of the grid.

Examples:

```
*Main> getLCC example
[(1,1),(1,2),(2,2),(2,3),(3,3),(4,4),(4,3),(5,3),(5,4)]
*Main> length $ getLCC example
9
```

Note: the order of the indexes in the returned list does not matter.