E-402-STFO PROBLEMS FOR MODULE D

CREATED BY HENNING ULFARSSON

This module is concerned with the Game of Life.

You get a perfect score for this module by getting 60 points or more.

The Game of Life was invented by John Conway and became popular when an article appeared about it in Scientific American in 1970. This is not a game in the conventional sense, since there are no players. The universe of the game consists of an n-by-n grid of cells that are alive or dead. The game proceeds in rounds and is governed by the following rules:

(1) An alive cell with one or no neighbor dies (it was too lonely)
(2) An alive cell with four or more neighbors dies (overpopulation)
(3) An alive cell with two or three neighbors survives
(4) A dead cell with three neighbors resurrects

In all of the problems below we play the game on a finite grid.

Problem 1 (5 points)**.** Write a function `mDp1(A,c)` that given an n-by-n 0-1 matrix `M`, representing a state of the game, and a cell `c = (i,j)` in the matrix, outputs the status of the cell in the next round (`0` being dead and `1` being alive).

```
Input: A = matrix([[1,0,1,0],[0,0,1,0],[0,0,0,0],[1,0,1,1]])
       c = (1,1)
Run: mDp1(A,c)
Output: 1
```

Problem 2 (5 points)**.** Write a function `mDp2(A)` that given an n-by-n 0-1 matrix `M`, representing a state of the game, outputs a new 0-1 matrix representing the next state.

```
Input: A = matrix([[1,0,1,0],[0,0,1,0],[0,0,0,0],[1,0,1,1]])
Run: mDp2(A)
Output: matrix([[0,1,0,0],[0,1,0,0],[0,1,1,1],[0,0,0,0]])
```

Problem 3 (5 points)**.** Write a function `mDp3(A,k)` that given an n-by-n 0-1 matrix `M`, representing a state of the game, and an integer `k` ≥ 0 outputs a new 0-1 matrix representing the state after `k` steps. An easy way to write this function is to repeatedly call the function `mDp2`.

```
Input: A = matrix([[1,0,1,0],[0,0,1,0],[0,0,0,0],[1,0,1,1]])
       k = 3
Run: mDp3(A,k)
Output: matrix([[0,0,0,0],[1,1,1,0],[1,0,1,0],[0,1,1,0]])
```

Problem 4 (10 points)**.** Write a function `mDp4(A,k)` that given an n-by-n 0-1 matrix `M`, representing a state of the game, and an integer `k` ≥ 0 outputs a new 0-1 matrix representing the state after `k` steps. Instead of repeatedly calling `mDp2`, lets try to save ourselves some work: Note that a cell that did not change in the

Date: Updated November 6, 2013.

last round, and none of whose neighbors changed, does not change at the current round. Use this to save time by not updating the inactive regions of the matrix.

```
Input: A = matrix([[1,0,1,0],[0,0,1,0],[0,0,0,0],[1,0,1,1]])
       k = 1000000
Run: mDp4(A,k)
Output: matrix([[0,1,0,0],[1,0,1,0],[1,0,0,1],[0,1,1,0]])
```

As you can see from the previous problem there are some states which become constant after a few time steps.

Problem 5 (10 points)**.** Write a function `mDp5()` that outputs 10 states which remain unchanged. The output should be a list of 0-1 matrices.

While you were working on the previous problem you might have discovered some oscillators, states which return to them selves after a few iterations.

Problem 6 (10 points)**.** Write a function `mDp6()` that outputs 5 states which return to them selves after at most 5 iterations. The output should be a list of 0-1 matrices. Note that you are not allowed to turn in a state and a symmetry of that state (for example you can not turn in both `matrix([[1,0],[0,1]])` and `matrix([[0,1],[1,0]])`. You are also not allowed to turn in a state and another state in its cycle.

Finally, you might have discovered some states which return to a shifted version of them selves after a few iterations.

Problem 7 (6 points)**.** Write a function `mDp7()` that outputs 2 states which return to a shifted version them selves after at most 5 iterations. The output should be a list of 0-1 matrices.

Instead of playing the game on a finite grid we can imagine that the left end of the grid is joined with the right end, and the top of the grid is joined with the bottom. This creates a torus, and if you've played the game Asteroids, you are familiar with the concept.

Problem 8 (10 points)**.** Write a function `mDp8(A,k)` that given an n-by-n 0-1 matrix `M`, representing a state of the game, and an integer `k` ≥ 0 outputs a new 0-1 matrix representing the state after `k` steps; when the universe is a torus. Use the idea of inactive regions from problem 4 to save yourself some unnecessary work.

```
Input: A = matrix([[0,0,0,0,0],[0,0,0,0,0],[0,0,1,1,1],[0,0,1,0,1],[0,0,1,1,1]])
       k = 1
Run: mDp8(A,k)
Output: matrix([[0,0,0,1,0],[0,0,0,1,0],[0,0,1,0,1],[1,1,0,0,0],[0,0,1,0,1]])
```

We can also play the game on a Klein bottle, where top is flipped before being joined with the bottom, so cell $(0, i)$ is attached to cell $(n-1, n-1-i)$.

Problem 9 (10 points)**.** Write a function `mDp9(A,k)` that given an n-by-n 0-1 matrix `M`, representing a state of the game, and an integer `k` ≥ 0 outputs a new 0-1 matrix representing the state after `k` steps; when the universe is a torus. Use the idea of inactive regions from problem 4 to save yourself some unnecessary work.

```
Input: A = matrix([[0,0,0,0,0],[0,0,0,0,0],[0,0,1,1,1],[0,0,1,0,1],[0,0,1,1,1]])
       k = 1
Run: mDp9(A,k)
Output: matrix([[0,1,0,0,0],[0,0,0,1,0],[0,0,1,0,1],[1,1,0,0,0],[0,0,1,0,1]])
```

Some very advanced implementations of Game of Life have been developed, see e.g., http://en.wikipedia.org/wiki/Hashlife.

School of Computer Science, Reykjavik University, Menntavegi 1, 101 Reykjavík, Iceland
E-mail address: henningu@ru.is