CSCB63 Assignment 2 - Coding

Due: 11:59 PM, Monday February 24, 2020

1. You've all seen the the Code Mangler, now we introduce his less-villainous cousin - the *Pixel Marcher*. Given some image, his mission is to walk from the top-left corner to the bottom-right corner. Every step he takes costs him some energy (which is determined by a weight function that is provided to you). The *Pixel Marcher* wants to find the path that takes up the least energy.

   The weight function which is provided is the key element that decides this least-energy path. For example, here is an input image with the output produced given 2 different weight functions:

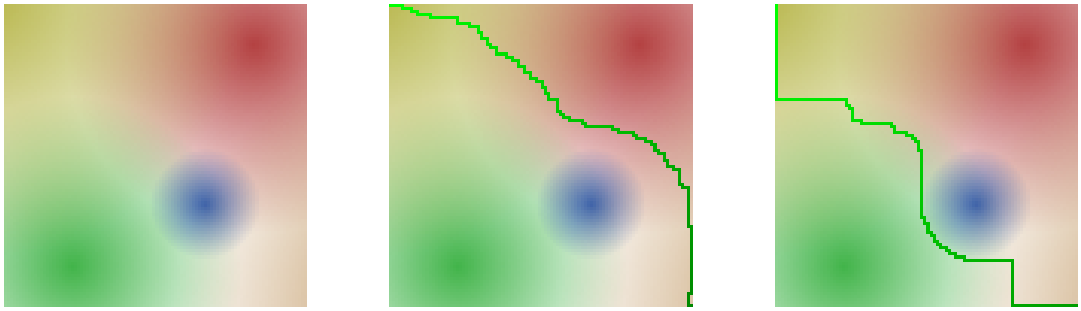

Figure 1: (Left) Input image, (Middle) Weight function 1, (Right) Weight function 2

In the above example, weight function (1) reflects how similar the pixel is to it's neighbour, and weight function (2) reflects how close the neighbouring pixel is to white. Both of these weight functions are provided to you as part of the starter code in the `TestMarcher.c` and `Driver.c` files.

Fun fact: You can use this algorithm exactly to help the Pixel Marcher solve mazes if you pick the correct weight function! Some small mazes have been provided as test cases for you.

You have two tasks (Look at `Marcher.c`):

(i) Given an input image and a weight function, complete the `findPath()` function that helps the Pixel Marcher find a least-energy path, and also the amount of energy he needs to spend to fulfill his mission.

(ii) Complete the weight function `allColourWeight()` so that:

- When `findPath()` is run with this weight function and the image `25colours.ppm`, the Pixel Marcher goes through at least 1 pixel of each of the 25 colours along the least-energy path.
- The Pixel Marcher can stay on pixels of the same colour for as many steps as needed, but once he leaves a colour, he can *never* step on any pixels of the same colour again. (He's very specific about this)
- The energy required to go between any two pixels is always **non-negative**.

The starter code provides comprehensive input specifications and requirements for both functions - so make sure you read everything before you ask any questions.

Note on implementation: We usually like to think of pixels in terms of a pair of (x, y) coordinates. However, images need to be stored in memory as linearly at the end of the day. For this reason, in this assignment we instead use the *linear index* of the pixels. The linear index is simply $x + y \cdot sx$, where sx is the width of the image.

You should be submitting **only** Marcher.c with the functions completed. Do not submit any of the other files or images. Additionally, the `expected` folder contains images displaying least-energy paths from the test cases for you to compare against.

Marking Scheme.

- Up to 8 marks can be earned by passing the test cases for `findPath()`. You may be tested on new images/weight functions. Each test case gets 1 second on the BV computers (though you will probably never reach this).
- 2 marks can be earned for having a correct solution to `allColourWeight()`. The order in which you visit the colours does **not** matter. Your function will be tested with *my* solution for `findPath()`, and must produce the correct result. There will be no part marks.
- You will receive 0 marks if your code does not compile, or fails all the test cases, with possible penalties for compiling with warnings.