CSCB63 Assignment 3
Amortized Complexity and Disjoint Sets

Due: 11.59pm, Saturday Mar 23, 2019

Warning: Your electronic submission on MarkUs affirms that this assignment is your own work and no one else's, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters, and the Code of Student Conduct.

1. (10 marks) Consider an abstract data type that consists of a set S of positive integers upon which the following two operations can be performed:

   DELETE(S, i): Delete integer i from the set S. If $i \notin S$, there is no effect.

   SUCCESSOR(S, i): Return the successor of integer i in S, i.e. $\min\{j \in S \mid j > i\}$. If i has no successor in S, i.e. if $i \geq \max S$, then return 0. Note that it is not necessary for i to be in S.

   Initially, S is a set of consecutive integers from a to b. For example, if $a = 5, b = 10$ then initially $S = \{5, 6, 7, 8, 9, 10\}$. After some DELETE() operations, suppose that $S = \{6, 8, 10\}$ then SUCCESSOR($S, 14$) = 0, SUCCESSOR($S, 3$) = 6 and SUCCESSOR($S, 6$) = SUCCESSOR($S, 7$) = 8.

   In this question you will describe a data structure with $O(\alpha(n))$ amortized cost per operation by answering the following questions.

   (a) What is your data structure to represent S?

   (b) How do you initialize your data structure and how much time does it take to initialize it?

   (c) Provide an algorithm for DELETE that takes $\mathcal{O}(\alpha(n))$ amortized time.

   (d) Provide an algorithm for SUCCESSOR that takes $\mathcal{O}(\alpha(n))$ amortized time.

2. (10 marks) Consider our binary tree HEAP data structure. Recall that it supports INSERT and EXTRACT_MAX in $O(\log n)$ worst case time, where n is the number of elements in the PRIORITY QUEUE.

   (a) Give a potential function Φ such that the amortized cost of INSERT is $O(\log n)$ and the amortized cost of EXTRACT_MAX is $O(1)$ with respect to Φ. Justify your answer.

   (b) Prove that for any constant c, the potential function $\Phi(H) = c \times \text{size}(H)$ is not a solution to (a).

   (c) Consider a sequence of n EXTRACT_MAX operations performed on a heap H that **initially** contains n elements. Does the fact that the amortized cost of each EXTRACT_MAX operation is $O(1)$ mean that the entire sequence can be processed in $O(n)$ time? Justify your answer.

3. (10 marks) 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:

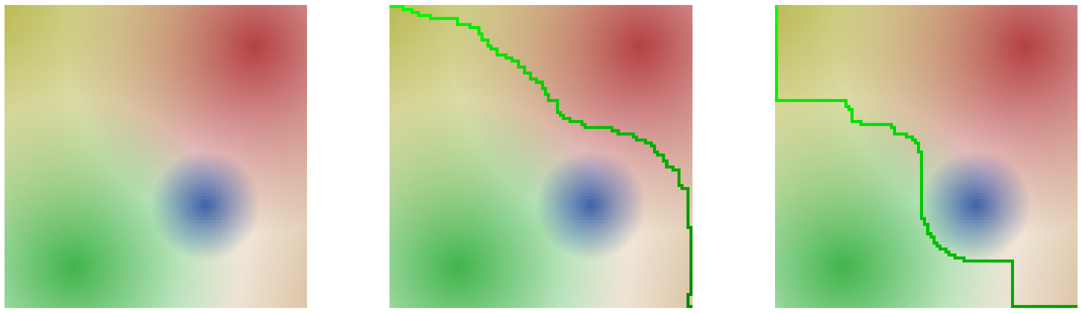

Figure 1: (Left) Input image, (Middle) Weight function 1, (Right) Weight function 2

   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 `Test.py` file.

   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:

   (i) [8 marks] 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) [2 marks] Complete the weight function `all_colour_weight()` 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.

   It also uses the Python Imaging Library (PIL) to read and write images to disk, which is already installed on all the lab computers. If you wish to work on this on your own computer, you may need to run `pip3 install Pillow`, or look up how to install it yourself. You are **not** allowed to import any additional libraries. You should implement your own data structures (if needed) beyond the basic ones Python provides.

You should be submitting **only** Marcher.py 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. But watch this: each test case gets 2 seconds only on the BV computers. Timing out is a failure. Each test case is timed separately.
- 2 marks can be earned for having a correct solution to `all_colour_weight()`. The order in which you visit the colours does **not** matter. Your function will be tested with *my* solution, 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.