`tag` — Version 1.1 ***due:*** *thu, mar 8 @ 11:59pm*

1 Introduction

In this homework, you will first prove the correctness of the Bellman-Ford algorithm for single-source shortest paths. You will then design and implement a shortest-path-based algorithm for resizing images. Finally, you'll do some exercises involving probability and expectation.

1.1 Submission

This assignment is distributed in a number of files in our `git` repository. Instructions on how to access that repository can be found at `http://www.cs.cmu.edu/~15210/resources/git.pdf`. This is how assignments will be distributed in this course.

This assignment requires that you submit both code and written answers.

Submit your solutions by placing you solution files in your handin directory, located at

```
/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn5/
```

Name the files exactly as specified below. You can run the check script located at

```
/afs/andrew.cmu.edu/course/15/210/bin/check/05/check.pl
```

to make sure that you've handed in appropriate files. Do not submit any other files.

Your written answers must be in a file called `hw05.pdf` and must be typeset. You do not have to use LaTeX, but if you do, we have provided the file `defs.tex` with some macros you may find helpful.

For the programming part, the only files you're handing in are

```
seamCarve.sml
```

These files must contain all the code that you want to have graded for this assignment and must compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

1.2 Naming Modules

The questions that follow ask you to organized your solutions in a number of modules. Your modules must be named exactly as stated in the handout. Correct code inside an incorrectly named structure, or a structure that does not ascribe to the specified signatures, *will not receive any credit*.

You may not modify any of the signatures or other library code that we give you. We will test your code against the signatures and libraries that we hand out, so if you modify the signatures your code will not compile and you will not receive credit.

1.3 The SML/NJ Build System

This assignment includes a substantial amount of library code spread over several files. The compilation of this is orchestrated by CM through the file `sources.cm`. Instructions on how to use CM can be found at on the website at `http://www.cs.cmu.edu/~15210/resources/cm.pdf`.

1.4 Style

As always, you will be expected to write readable and concise code. If in doubt, you should consult the style guide at `http://www.cs.cmu.edu/~15210/resources/style.pdf` or clarify with course staff. In particular:

1. **If the purpose or correctness of a piece of code is not obvious, document it.** Ideally, your comments would convince a reader that your code is correct.
2. **You will be required to write tests for any code that you write.** In general, you should be in the habit of thoroughly testing your code for correctness, even if we do not explicitly tell you to do so.
3. **Clearly indicate parallelism in your code.** Use the provided `par` and `par3` functions in the library structure `Primitives` for this purpose.

2 Correctness of Bellman Ford

We covered the Bellman-Ford algorithm in Lecture 11, in which we showed that on a graph with n vertices and m edges, the algorithm runs in $O(nm)$ work and $O(n \log n)$ span using an array sequence implementation. An important feature of the Bellman-Ford algorithm is that unlike Dijkstra's, it correctly computes the shortest paths even when we have negative weight edges.

Task 2.1 (15%). You will prove the correctness of the Bellman-Ford algorithm. Specifically, you'll show the following theorem:

Theorem 2.1. *Let $G = (V, E)$ be a simple, connected directed graph with a weight function $w : E \to \mathbb{R}$, which assigns to each edge $e \in E$ the weight $w(e)$ (possibly negative). Let $s \in V$ be a starting vertex. If G contains no negative weight cycles*[1]*, then the Bellman-Ford algorithm described in class correctly computes the shortest path from s to every vertex in the graph.*

Note that the Bellman-Ford algorithm presented in class terminates within n steps, because we keep a counter i and we quit when i reaches $|V|$.

(Hint: Consider walks of length i. What can we say about the shortest-path distances D in the iteration that `BF` was called with this particular value of i?)

[1] a negative weight cycle is a cycle where the sum of edge weights is negative

3 Seam Carving

Seam carving is a (relatively) new technique (2007) for resizing an image by finding and deleting the "least resistant paths" in the image. Traditional image resizing scales the image, causing objects to be stretched out or squished. In contrast, seam carving repeatedly identifies seams of pixels that can be added or removed without affecting the salient parts of the image. For a demo of seam carving, you can watch the following video:

`http://www.youtube.com/watch?v=6NcIJXTlugc`

In this assignment, you will first answer some questions about seam carving and then implement code to support it. For simplicity, we will only worry about horizontal resizing. Therefore, in this case, a seam is any vertically connected path of pixels. To resize the image, we identify the seam with the lowest cost and then remove it. In this case, the cost of a seam is defined as $\sum_{i \in rows} c(i, j_i)$ where $c(i, j_i)$ is the magnitude of the gradient of the image for the pixel chosen for row i, which is in column j_i. To create a valid seam, the pixel in the row must be attached to the one from the previous row. For example, if in row 5, $p(5,3)$ was chosen, then in row 6, we could choose either $p(6,2)$, $p(6,3)$ or $p(6,4)$. Let's do an example of a 4x4 image with $c(i, j)$ displayed in each box.

i ↓ j →	**0**	**1**	**2**	**3**
0	5	3	2	4
1	6	1	7	8
2	2	7	1	2
3	4	7	6	8

Task 3.1 (5%). Let $q(i, j)$ be the minimum cost to get from anywhere in row 0 to pixel (i, j). Fill in the table for $q(i, j)$.

i ↓ j →	**0**	**1**	**2**	**3**
0				
1				
2				
3				

Task 3.2 (3%). What is the lowest cost vertical seam in the image?

Task 3.3 (5%). Now that you've worked through a simple example, let's examine the general case for an $h \times w$ (height x width) image. How would you convert this to a shortest path problem? (Hint: The resulting graph should be a directed acyclic graph (DAG).)

Task 3.4 (2%). List the algorithms from class that can be used to solve this problem.

Task 3.5 (5%). Describe an algorithm for solving this problem that runs in $O(h \cdot w)$ work and $O(h \log w)$ span.

Task 3.6 (20%). Now you need to implement code for seam carving. In particular, you need to complete the code in the file `seamCarve.sml`. We supply the code for generating the gradients from an image as well as various functions for trying out your code on real images. Since these functions make use of external calls to some Python scripts, some of the functionality might not be compatible with all operating systems and machine configurations. In particular, Mac OS X does not come with the python imaging library preinstalled, so our scripts for reading/writing .jpg files will not work. Everything should work on the Andrew linux machines (knock on wood).

4 Probability

Task 4.1 (5%). In HW 2, you carefully analyzed the number of times insertion sort called the `swap` function for a specific input A. What is the expected number of times `swap` is called during insertion sort as a function of $n = |A|$? Assume all $n!$ permutations of $[n]$ are equally likely inputs. (Hint: Use linearity of expectations.)

Task 4.2 (10%). Let X be a non-negative random variable. That is to say, for all all possible outcomes $\omega \in \Omega$, $X(\omega) \geq 0$. Prove that for a value $a > 0$,

$$\mathbf{Pr}[X \geq a] \;\leq\; \frac{\mathbf{E}[X]}{a}.$$

Prove also that this bound is tight by giving an example of a random variable X and a real number $a \neq 0$ such that $\mathbf{Pr}[X \geq a] = \mathbf{E}[X]/a$. (Hint: what does $\mathbf{Pr}[X \geq a]$ say about $\mathbf{E}[X]$?)

Task 4.3 (10%). Let f be a non-decreasing function satisfying

$$f(n) \leq f(X_n) + \Theta(n), \qquad \text{where } f(1) = 1.$$

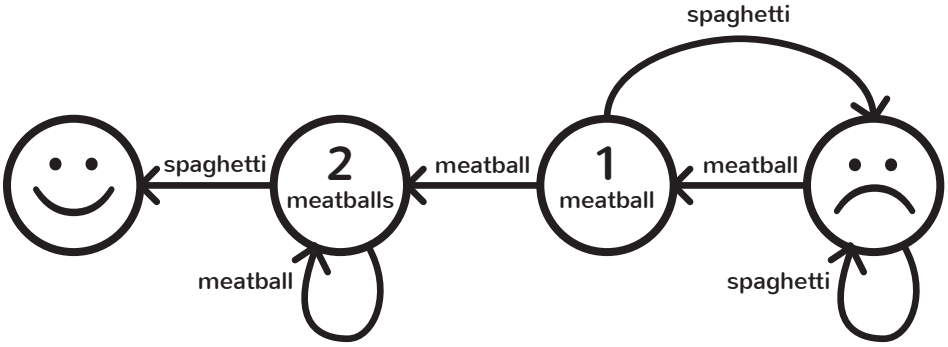
Prove that if for all $n > 1$, $\mathbf{E}\left[X_n\right] = n/2$, then $\mathbf{E}\left[f(n)\right] \in O(n)$.
(Hint: What's the probability $\mathbf{Pr}\left[X_n \geq 3n/4\right]$?)

Task 4.4 (10%). Consider the following algorithm to compute the minimum of a sequence:

```
/* Precondition: length(A) > 0 */
int min(int[] A) {
    int min = A[0];
    for(int i=1; i<n; i++)
        if(A[i] < min)
            min = A[i];
    return min;
}
```

If $A = [1,2,3]$, we assign to min only once, whereas if $A = [3,2,1]$, we assign to min 3 times. Calculate the expected number of times we assign to min as a function of n, the length of A (assume all $n!$ permutations of A are equally likely, and that the elements of A are distinct).

Task 4.5 (10%). **I put a FSM in your FSM:** In the basement of Gates, there exists a food dispenser which produces a meatball with probability $\frac{1}{3}$, and a serving of spaghetti with probability $\frac{2}{3}$, all with the push of a button. The Flying Spaghetti Monster (FSM) wishes to eat a meal consisting of two meatballs and a serving of spaghetti. However, the FSM can only consume spaghetti directly after eating two meatballs in a row due to a rare indigestion disorder.



Let X be the number of times the FSM must push the button on the food dispenser for a full meal. Specifically, X is the number of button presses required to obtain a sequence of meatball, meatball, spaghetti, in that order. What is the expected value of X? As a hint, we provide you with the above finite state machine (FSM) representing each state that the Flying Spaghetti Monster could be in after each of push of the button. You may wish to write out recurrences for each state.