Homework 9

15-210: Parallel and Sequential Data Structures and Algorithms (Fall'12)

`tag` — Version 1.0 **due:** *Fri, Nov 30th @ 11:59pm*

Disclaimer:
We will not grade non-compiling code.

1 Introduction

In this assignment you will explore applications of dynamic programming in the form of a short programming task and two relatively easy dynamic programming problems. The programming task involves dynamic image resizing using the seam carving technique which was only recently discovered in 2007.

1.1 Submission

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

```
/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn9/
```

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

```
/afs/andrew.cmu.edu/course/15/210/bin/check/09/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 `hw09.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. This file should be included at the top of your `.tex` file with the command `\input{<path>/defs.tex}`.

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

```
SeamFinder.sml
original.jpg
seamcarved.jpg
```

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.

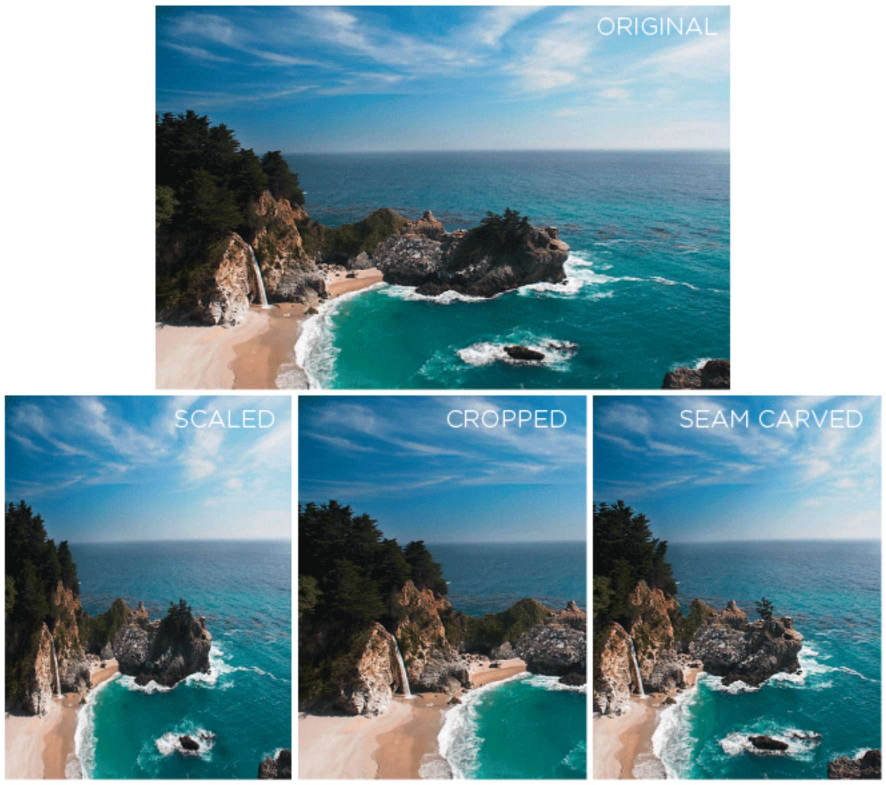
1.2 Style

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. **Code is Art.** Code *can* and *should* be beautiful. Clean and concise code is self-documenting and demonstrates a clear understanding of its purpose.
2. **If the purpose or correctness of a piece of code is not obvious, document it.** Ideally, your comments should convince a reader that your code is correct.
3. **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.
4. **Use the check script and verify that your submission compiles.** We are not responsible for fixing your code for errors. If your submission fails to compile, you will lose significant credit.

2 Seam Carving

Seam Carving is a relatively new technique discovered for "content-aware image resizing" (Avidan & Sheridan, 2007). Traditional image resizing techniques involve either scaling, which results in distortion, or cropping, which results in a very limited field of vision.



The technique involves repeatedly finding ‘seams’ of least resistance to add or to remove, rather than straight columns of pixels. In this way the salient parts of the image are unaffected. It’s a very simple idea, but very powerful. You will work through some simple written problems and then implement the algorithm yourself to use on real images. For simplicity, we will only be dealing with horizontal resizing. To motivate this problem, watch the following SIGGRAPH 2007 presentation video which demonstrates some surprising and almost magical uses of this technique:

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

2.1 Logistics

2.1.1 Representation

Images will be imported by a Python program that uses image libraries which may not be compatible with your local machine. Make sure to use one of the Andrew machines to run your program. The imported `image` type is defined as

```
type pixel = { r : real, g : real, b : real }
type image = { width : int, height : int, data : pixel seq seq }
```

2.1.2 Gradients

To find a seam of least resistance, we define the following: for any two pixels $p = (r_1, g_1, b_1)$ and $q = (r_2, g_2, b_2)$, the *pixel difference* δ is the sum of the differences squared for each RGB value:

$$\delta(p,q) = (r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2$$

Then, for a given pixel $p_{i,j}$, the *gradient* $g(i,j)$ is defined as the square root of the sum of its difference from the pixel on the right and its difference from the pixel below it.

$$g(i,j) = \sqrt{\delta(p_{i,j}, p_{i,j+1}) + \delta(p_{i,j}, p_{i+1,j})}$$

This leaves the right-most column and bottom row undefined. For an image with height n and width m, we define $g(n-1, j) = 0.0$ for any j, and $g(i, m-1) = \infty$ for any i. Using the gradient as a cost function, we can now develop an algorithm to compute a lowest-cost seam for any image.

2.2 Specification

Task 2.1 (10%). Implement the function

```
generateGradients :  image -> gradient seq seq
```

in `SeamFinder.sml` which converts a raw `image` type into a 2D sequence of gradients $g(i,j)$ as defined above. Your implementation should have work in $O(nm)$ and constant span.

2.2.1 Seam Finding Algorithm

Let's work through a simple example. Suppose we have a 4x4 image with the following table of gradient values already computed (ignoring the right-most column of ∞'s and the bottom row of zeroes):

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 2.2 (7%). A valid vertical seam must consist of m pixels if the image has height m, and each pixel must be *adjacent* in the sense that if the seam at row i is at column j, the seam at rows $i-1$ and $i+1$ are limited to columns $j-1$, j, or $j+1$. The cost of a seam is $\sum_{p_{i,j} \in \text{seam}} g(i,j)$. Let $m(i,j)$ be the minimum cost to get from row 0 to $p_{i,j}$. Fill in the table for $m(i,j)$.

i ↓ j →	**0**	**1**	**2**	**3**
0		3		
1	9			
2				
3				

Task 2.3 (3%). What is the lowest cost vertical seam in this image?

Task 2.4 (10%). For the general case, write the mathematical definition of $m(i,j)$ and describe a bottom-up dynamic programming algorithm to find the minimum cost seam. What is the asymptotic work and span of this algorithm? Use a Θ bound.

Task 2.5 (25%). Using this insight, implement the function

```
findSeam :  image -> int seq
```

in `SeamFinder.sml` which finds the lowest-cost seam in the given image, where the seam is represented as an ordered sequence of column indices going from the top row of the image to the bottom row.

2.2.2 Testing

As usual, you will be required to test your code. But it's going to be a little more fun this time! In the handout directory we've given you a few sample images, as well as headshots of the entire course staff to play with (`/hw/09/sml/images/` and `/hw/09/sml/images/staff/`). After completing the above tasks, you should be able to perform the following in the REPL:

```
$ smlnj
Standard ML of New Jersey v110.73 [built: Wed Aug 24 12:35:24 2011]
- CM.make "sources.cm";
[autoloading]
...
[New bindings added.]
val it = true : bool
- SeamCarve.removeSeamsFile ("images/cove.jpg", "images/cove-50.jpg", 50);
val it = () : unit
```

This takes the image `images/cove.jpg`, removes 50 seams from it, and stores the resulting image in `images/cove-50.jpg`. You should compare your results with the given `cove-50.jpg`, `cove-100.jpg`, and `cove-200.jpg` samples. For reference, our solution removes 100 seams from `cove.jpg` in < 8 seconds. Removing more seams from higher resolution images will naturally take longer.

Task 2.6 (5%). Find and submit an image that has an interesting seam-carved result. You should submit the original as `original.jpg` and the altered version as `seamcarved.jpg`. Points will be for awarded for creativity, humor, and self-expression. Dupes will be frowned upon, so be unique!

3 Written Problems

3.1 Subsequence Counting

Suppose you want to count the number of times that a sequence of letters S appears as a subsequence of a string s. The subsequence does not necessarily have to be contiguous. For example, the sequence $\langle k, t, y \rangle$ occurs twice in the string "kitty".

Task 3.1 (2%). How many ways does the string "abbabab" contain the sequence $\langle a, b \rangle$?

Task 3.2 (2%). How many ways does the string "abbabab" contain the sequence $\langle a, b, a \rangle$?

Task 3.3 (8%). Let S_i denote the first i elements of the subsequence S and s_j denote the first j characters of the string s. Consider the subproblem of finding the number of times the prefix S_i appears as a subsequence of the prefix s_j denoted as $C(S_i, s_j)$.

Write a recursive function (in pseudocode or SML) to find $C(S_i, s_j)$ in terms of smaller subsequences of S and smaller substrings of s. Include the base case $C(\emptyset, s_j)$, $0 \le j < |s|$. That is to say, how many times does the empty sequence appear in the first j characters of s? What is the final answer?

Task 3.4 (5%). Since this is a dynamic programming solution, the recursive call structure is a directed acyclic graph (DAG). Place a bound on the number of vertices (subproblems) in your DAG as well as the longest path in the DAG, in terms of $|S| = n$ and $|s| = m$. What is the work and span of your algorithm? Use Θ notation and explain why.

3.2 Word Breaking

Suppose you are given a sequence of characters $S = \langle c_0, c_1, c_2, \cdots, c_{n-1} \rangle$ that has no spaces or punctuation. You want to determine if you can parse the string into a sequence of words, given a dictionary of valid words. For example, "winterbreakisalmosthere" can be broken into "winter break is almost here". If the sequence has n characters, then are $n-1$ places in which to put a space or not and so there are 2^{n-1} ways to insert spaces. You are to develop a dynamic programming algorithm to break up the sequence into words. You may assume that words are bounded by length k and that you can determine if a word w is in a dictionary in $O(|w|)$ work and span.

Task 3.5 (2%). A greedy algorithm would find a word at the beginning of the sequence, then repeat on the rest of the sequence. Give an example where using a greedy method would not find a valid segmentation of the sequence when one exists.

As is often the case with dynamic programming solutions, instead of finding where to put spaces, you will first consider the decision problem to simplify the problem space: Can you break the character sequence into a sequence of valid words? Later, you will reconstruct where the spaces can go.

Task 3.6 (3%). In the previous problem, we defined for you the subproblems using the notation $C(S_i, s_j)$. For this problem, you will define your own. Clearly state the subproblems and notation that your dynamic programming solution will use.

Task 3.7 (8%). Write a recursive function (in pseudocode or SML) to determine if a sequence of characters can be broken into words based on a dictionary D. Be sure to include all the base cases. Assume a word is itself a sequence of characters, and that for a word w, $D[w]$ returns `true` if w is in the dictionary and `false` otherwise.

Task 3.8 (5%). Place a bound on the number of vertices (subproblems) in your DAG as well as the longest path in the DAG, in terms of $|S| = n$ and the maximum word size k. What is the work and span of your algorithm? Use Θ notation and explain why.

Task 3.9 (5%). Briefly describe how to modify your code to return a list of locations where to insert the spaces. There may be many possible solutions — You need to return just one.