CSE 3500 **Homework 1**

Programming Assignment – Image Compression

Complete this programming assignment solo or with a partner. If you need a partner, submit the partner request form on HuskyCT before 2 days elapse. High level collaboration is encouraged but you and your partner's hand-in should represent your own understanding of the material. Please refer to the notes on programming assignment collaboration in the syllabus.

Introduction

In a 2007 paper, Avidan and Shamir describe a new method for resizing images with minimal visual disruption [1]. The paper is posted to HuskyCT and the demonstration video provides a great overview of the work. The breakthrough is an algorithm for removing the lowest *energy* vertical or horizontal *seam* in an image.

Notation and Definitions

Please note, in this assignment we will be following the conventions that (0, 0) is the upper-left pixel and that i **indexes column and** j **indexes row**. We are given a picture bitmap represented by an $m \times n$ array $B[1, ..., m, 1, ..., n]$ of pixels, where each pixel, $B[i, j]$, encodes red, green, and blue (RGB) intensities. Our goal is to compress the image by removing a single pixel from each row (or column), n (or m) in total. We do not want to distort the global structure, so if we remove a pixel from row j column i, we require that the pixels removed from rows $j - 1$ and $j + 1$ be in the column range of $[i - 1, i + 1]$, i.e. at most 1 column away from i. We call the pixels removed in this manner a *seam*.

Problem

Similar to edit distance, we define a cost for removing a pixel to be $energy(i, j)$ which corresponds to the *energy* of the pixel, or, the level of image disruption that would occur by removing pixel $B[i, j]$. Informally, the higher the energy, the more dissimilar the pixel is to its neighbors. The total cost is $T_s = \sum_{(i,j)\in s} energy(i, j)$ where s is the seam. The DP subproblem is, for each pixel (i, j), what is the lowest-energy seam that starts at the top row of the image, but ends at (i, j)?

Let $\texttt{dp}[i, j]$ be the solution to subproblem (i, j). Then, for a vertical seam, $\texttt{dp}[i, j] = \min(\texttt{dp}[i - 1, j - 1], \texttt{dp}[i, j - 1], \texttt{dp}[i + 1, j - 1]) + energy(i, j)$

Implementation

We have provided support to more easily complete this assignment in Python, but solutions in other languages will be accepted.

Download `prog_assign1.zip` from HuskyCT and unzip. You will need to install the Python Pillow library. You should be able to install it with:

```
pip install Pillow
```

or

```
easy_install Pillow
```

or download https://pypi.python.org/pypi/Pillow and from inside the archive run:

```
python setup.py install
```

The provided code assumes you are using Python3. In `resizable_image.py`, implement a function `best_seam(self, dp=True)` that returns a list of coordinates corresponding to the lowest-energy vertical seam to remove, e.g. `[(5, 0), (5, 1), (4, 2), (5, 3), (6, 4)]`.

The class `ResizeableImage` inherits from `ImageMatrix`. You should use the following components of `ImageMatrix` in your program:

- `self.energy(i,j)` returns the energy of a pixel. This takes $O(1)$ time, but the constant factor is large. If you call *energy* more than once on the same pixel, you should cache the result. *You should still cache the energy for a pixel even if* `dp == False`.
- 'self.width' and 'self.height' are the width and height of the image.

Your implementation may be either bottom-up or top-down. But either way, it must respect the argument `dp`, which indicates whether or not dynamic programming should be used. If `dp == True`, then you should either use memoization or store the subproblem values in a table for re-use. If `dp == False`, then you should naively recompute those subproblems.

- test your code using `test_resizable_image.py`, and submit `resizable_image.py` to HuskyCT.

- you can also view your code working by running `gui.py`
- you should try out your code on other images of your choosing, once it's working

Since the default behavior of `best_seam()` is to perform dynamic programming, the test suite provided will only test that variant. However, for small inputs, turning off dynamic programming should be possible. You will still be graded on the design and representation of your code.

Benchmarking

On `sunset_small.png` and several other *small* image files, and potentially on `sunset_full.png` (if the running time is not excessive), time your code both *with* and *without* dynamic programming. You should test on enough inputs to be able to make convincing plots of running time. Clearly, there will be inputs on which only the dynamic programming solution is tractable, so I expect your naive plot will cover fewer data points. Think about what it means to vary the image size. Plot the running times.

Analysis

1. What is the recurrence for a horizontal seam?
2. Assume $m > 1$. Show that the set of potential vertical seams grows in complexity at least exponential in n.
3. What is the asymptotic time complexity of the dynamic programming algorithm?
4. Does the time complexity of both the DP and non-DP algorithm make sense given the results of the benchmarking?

Grading Rubric

Your grade will be based on four components:

- Functional Correctness (40%)
- Benchmarking comparison (20%)
- Design and Representation (20%)

- Analysis of the algorithm (20%)

We still expect your code to be well-documented and well-structured.

References

[1] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM Transactions on graphics (TOG)*, volume 26, page 10. ACM, 2007.