Ali Kafel
Collaborator: Matt Jaccino

Homework 1

CSE 3500
February 17, 2019

Implementation

Consider a picture as a table of pixels, n being the height of the table and m being the width. Let i index the columns of the table and j index the rows of the table. Each (i,j) coordinate of this table represents a pixel on the image. Each pixel has an energy attached to it and this represents the cost of removing this pixel. In order to resize this image in a window we must remove a series of pixels in a vertical path called a seam, S. But we must remove the seam that has the lowest sum of energies from it's pixels. The best was way to solve this problem is to use dynamic programming. In order to do that we must define a function $S_n(j)$, representing the lowest energy seam that ends at (n, j). To compute this function we must assume we have the solution for the n-1 rows, $S_{n-1}(j)$ and pick the minimum $S_{n-1}(j)$ from our starting point. Thus we can define a recurrence relation as follows:

$$S_n(j) = \left\{ \begin{array}{ll} S_{n-1}(j-1), & \text{or } \infty \text{ if } j = 1 \\ S_{n-1}(j) & \\ S_{n-1}(j+1) & \text{or } \infty \text{ if } j = m \end{array} \right\} + e(n, j)$$

and use this to model our DP and our non-DP algorithms.

Algorithm 1: DP Algorithm

```
import imagematrix

class ResizeableImage(imagematrix.ImageMatrix):
    def best_seam(self, dp=True):

        n = self.height
        m = self.width

        pixel_energies = {}  # stores energies of all pixels computed
        for j in range(n):
            for i in range(m):
                pixel_energies[(i, j)] = self.energy(i, j)

        if dp == True:  # dp alg

            seams = {}  # stores seams computed and their energies, key = pixel(i, j)

            for i in range(m):  # top row of pixels
                seams[(i, 0)] = (self.energy(i, 0), [(i, 0)])

            def energy_from_seams((i, j)):  # key for getting energy of a seam
                return seams[(i, j)][0]

            for j in range(1, n):  # dp alg
                for i in range(m):
                    if i > 0:
                        uleft = (i-1, j-1)
                    else:
```

```
                    uleft = (0, j-1)
                u = (i, j-1)
                if i < m-1:
                    uright = (i+1, j-1)
                else:
                    uright = (m-1, j-1)
                seamuppix = min(uleft, u, uright, key=energy_from_seams)
                seamup = seams[seamuppix][1]
                seamupen = seams[seamuppix][0]
                newseamup = seamup + [(i, j)]
                newseamupen = seamupen + pixel_energies[(i, j)]
                seams[(i,j)] = (newseamupen, newseamup)

        minseam = []  # finds best seam by iterating through seams dictionary
        minseamen = float("inf")  # initialize min energy as high as possible
        for i in range(m):
            if seams[(i, n-1)][0] < minseamen:
                minseam = seams[(i, n-1)][1]
                minseamen = seams[(i, n-1)][0]

        return minseam
```

Algorithm 2: Recursive Non DP Algorithm

```
def best_seam(self, dp=True):

    n = self.height
    m = self.width

    pixel_energies = {}  # stores energies of all pixels computed
    for j in range(n):
        for i in range(m):
            pixel_energies[(i, j)] = self.energy(i, j)

    if dp == False: # non dp, recursive alg

        def get_seam_energy(seam_tup):  # key for getting seam energy
            return seam_tup[0]

        def seam_builder(pixel):  # builds a seam given a certain pixel
            i = pixel[0]
            j = pixel[1]
            if j == 0:
                return (pixel_energies[(i,j)], [(i,j)])
            else:
                if i > 0:
                    uleft = (i-1, j-1)
                else:
                    uleft = (0, j-1)
                u = (i, j-1)
                if i < m-1:
```

```
                uright = (i+1, j-1)
            else:
                uright = (m-1, j-1)

            ulseam = seam_builder(uleft)
            useam = seam_builder(u)
            urseam = seam_builder(uright)

            bestchoice = min(ulseam, useam, urseam, key=get_seam_energy)
            builden = bestchoice[0] + pixel_energies[(i, j)]
            buildseam = bestchoice[1] + [(i, j)]

            return (builden, buildseam)

    minseam = []  # finds min seam by iterating through bottom pixels
    minseamen = float("inf")
    for i in range(m):
        currseam = seam_builder((i, n-1))
        currseamen = currseam[0]
        if currseamen < minseamen:
            minseamen = currseamen
            minseam = currseam
    return minseam
```

Benchmarking

Using Python's time module, I recorded the time it took to run our best seam function of certain size images. I ran our algs on increasing size images multiple times and made line scatter plots for each alg. The Y axis is the runtime, the X axis being the size of the picture. Each point on the scatter plot is an average runtime for the alg for that size picture. Then I made a line of best fit for the points in the plot representing the runtime of the alg as the size or the picture increases. The non DP alg took very long to run on large pictures so I don't have that many data points for it. The following code was used to run the benchmarks:

Algorithm 3: Benchmarking

```
 # Importing
import time
import matplotlib.pyplot as plt
from resizeable_image import ResizeableImage

def average(l):
    total = 0
    for num in l:
        total += num
    return total/len(l)
```

```
#dp benchmark
times =[]
pics = ['small pic 1', 'small pic 2', 'medium pic 1', 'medium pic 2', 'large pic 1']
#small pic1
image = ResizeableImage('sunset_small.png')
sm1times = []
for i in range(10):
    t0 = time.time()
    seam = image.best_seam()
    t1 = time.time()
    tf = t1 - t0
    sm1times.append(tf)
sm1average = average(sm1times)
times.append(sm1average)
```

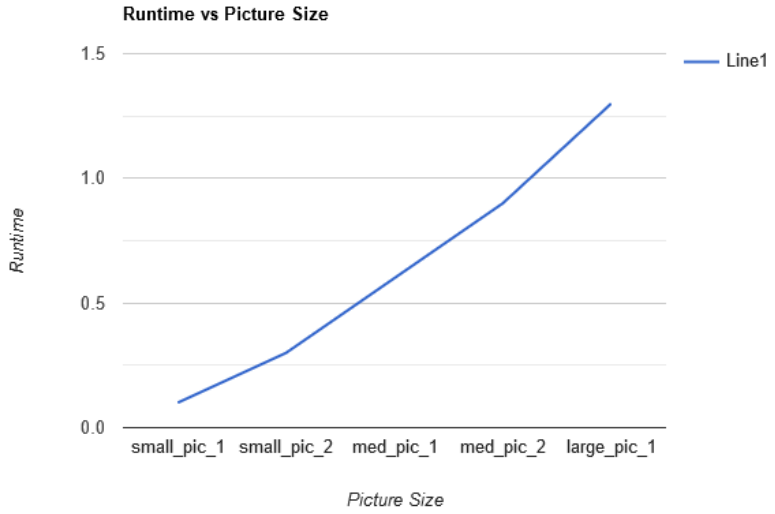

Figure 1: DP Alg

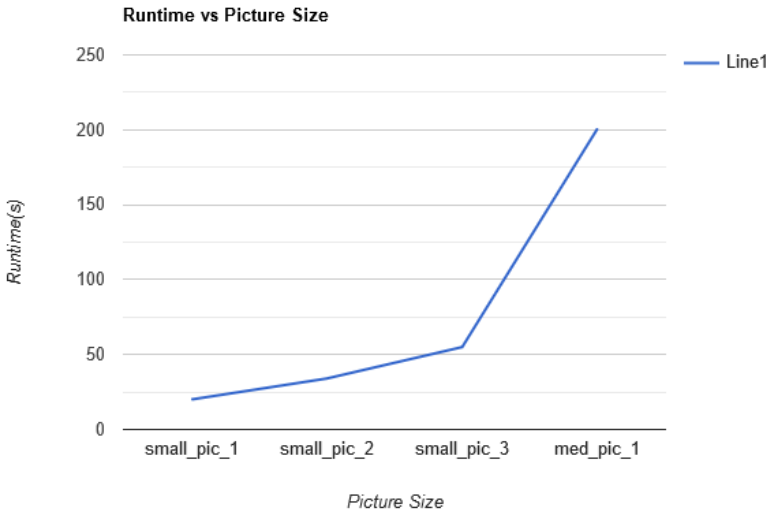

Figure 2: Non-DP Alg

The plots are placed on the last page. As you can see the DP alg grows a lot slower than the non DP alg for runtime as the size of the picture grows. And that matches the theoretical expectations as well.

Analysis

1. The recurrence relation for the horizontal seam (using the same variables as defined in the implementation section) can be defined as followed:

$$S_m(i) = \left\{ \begin{array}{ll} S_{m-1}(i-1), & \text{or } \infty \text{ if } i = 1 \\ S_{m-1}(i) & \\ S_{m-1}(i+1) & \text{or } \infty \text{ if } i = n \end{array} \right\} + e(i,m)$$

2. Given the recurrence relation:

$$S_m(i) = \left\{ \begin{array}{ll} S_{m-1}(i-1), & \text{or } \infty \text{ if } i = 1 \\ S_{m-1}(i) & \\ S_{m-1}(i+1) & \text{or } \infty \text{ if } i = n \end{array} \right\} + e(i,m)$$

   The set of possible seams does grow exponential to m because we are examining three different seams at each level of the table and applying the relation.

3. Let n be the height of the DP table and m be the width of the DP table. Then the asymptotic time complexity is O(n*m). This is because the worst case scenario would be that the best seam is at the end of the table.

4. Yes, the time complexity of both algs make sense according to the benchmarking. The DP alg had a sort of linear, straight curve when looking at it's plot of runtimes as expected theoretically. And the Non DP has an exponential curve when looking at it's plot of runtimes as expected theoretically.