Programming Assignment 3
Points: 500
Due: April 22, 11:59PM
Late Submission April 23, 11:59PM (25% penalty)

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistants for any questions/clarifications regarding the assignment. Your programs must be in Java, preferably Java 8.1.

All your classes must be in the **default package** (even though it is not a good programming practice). For this PA, you **may work in teams of 2**. It is your responsibility to find a team member. If you can not find a team member, then you must work on your own.

1 Dynamic Programming

You will use dynamic programming to compute *min cost vertical cut* of a matrix and to perform *string alignment*. You will design a class named `DynamicProgramming`.

1.1 Cuts

Let M be a $n \times m$ integer matrix (with n rows and m columns). Assume that rows are numbered $0, 1, 2, \cdots, n-1$ and columns are numbered $0, 1, 2, \cdots m-1$. Let $M[i, j]$ refers to the cell in ith row and jth column. A *vertical cut* V of M is a sequence of $2n$ integers $[x_0, y_0, x_1, y_1, \cdots, x_{n-1}, y_{n-1}]$ such that the following holds

1. $x_0 = 0$, $y_0 \in \{0, \cdots, m-1\}$
2. For $1 \leq i < n$, $x_i = x_{i-1} + 1$
3. For $1 \leq i < n$ $y_i \in \{y_{i-1}, y_{i-1} - 1, y_{i-1} + 1\}$

The *cost of a vertical cut* $V = [x_0, y_0, x_1, y_1, \cdots, x_n, y_n]$ is defined as

$$Cost(V) = M[x_0, y_0] + M[x_1, y_1] + \cdots + M[x_{n-1}, y_{n-1}]$$

Given a matrix, the *min-cost vertical cut*, denoted $MinVC(M)$, is a vertical cut whose cost is the smallest.

1.1.1 Your Task—Compute $MinVC$

Include the with the following public **static** method in the class `DynamicProgramming`.

`minCostVC(int[][] M)`: Returns a min-cost vertical cut. Type of this method must be `array list of integers`. Note that if M has n rows, the the returned array list has exactly $2n$ integers.

You **must** use dynamic programming paradigm to arrive at your code. For this, first define the recurrence relation. Then arrive at an iterative solution. Your code **must be iterative, not recursive and should not use use memoization**. Otherwise you will receive **zero** credit.

1.2 Alignment

Given two characters a and b, we define a function $penalty(a, b)$ as follows: if a equals b, $penalty(a, b) = 0$. If a or b equals \$, then $penalty(a, b) = 4$; otherwise $penalty(a, b) = 2$. Given two strings $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_n$ alignment cost between x and y is

$$AlignCost(x, y) = \sum_{i=1}^{n} penalty(x_i, y_i).$$

Let $x = x_1x_2, \cdots x_n$ and $y = y_1y_2 \cdots y_m$ two strings where $n \geq m$. assume that neither string has the character \$. Our task is to *align* the two strings. The alignment is done by creating a new string z formed by inserting the character \$ at $n - m$ indices in y. Note that length of z is n. Note that there are many choices for z. The goal is to find a z for which $AlignCost(x, z)$ is minimized.

We would like to write a method that gets two strings x (of length n) and y (of length m) as parameters (assume that $n \geq m$) and computes a z such that $AlignCost(x, z) \leq AlignCost(x, y)$ for every y.

1.2.1 Your Task—Compute Min Cost Alignment

Include the following `static` method in the class `DynamicProgramming`

`stringAlignment(String x, String y)` . Assume that x is a string of length n and y is a string of length m such that $n \geq m$. This method returns a string z (obtained by inserting \$ at $n - m$ indices in y) such that $AlignCost(x, z) \leq AlignCost(x, y)$ over all possible y. You may assume that length of x is at least the length of y and neither of x or y has the character \$. Note that the length of the returned string z must equal the length of x.

You **must** use dynamic programming paradigm to arrive at your code. For this, first define the recurrence relation. Then arrive at an iterative solution. Your code **must be iterative, not recursive and should not use use memoization**. Otherwise you will receive **zero** credit.

2 An Application of MinCost Vertical Cut

The notion of min-cost vertical cut has applications in image processing—in resizing images. Recall that an image is a 2-dimensional array of pixels. Each pixel has represented as 3-tuple in the RGB format. Given an image of width W and height H, it is represented by a $H \times W$ matrix (2-D array). For example an image of width 4 and height 3 is represented as the following 2-D array of pixels M:

```
[98, 251, 246]  [34, 0, 246]  [255, 246, 127]  [21, 0, 231]
[25, 186, 221]  [43, 9, 127]  [128, 174, 100]  [88, 1, 143]
[46, 201, 132]  [23, 5, 217]  [186, 165, 147]  [31, 8, 251]
```

In the above $M[i, j]$ refers to the pixel in the ith row and jth column. For example $M[2, 3]$ is $[31, 8, 251]$ and $M[0, 0]$ is $[98, 251, 246]$.

Suppose you would like to reduce the width of an image, standard techniques such as cropping may remove some prominent features of the image. Ideally, we would like to preserve the most important features of an image even while reducing the width. Suppose that width of an image is W and we would like to re-size the width $W - 1$. Thus we would like to remove one pixel from *each row* of the image, so that the width becomes $W - 1$. Which pixels should be removed from each row? Ideally, pictures that are not *important*. How do we decide the importance of a pixel? While there are several approaches, for this PA we consider the following approach to decide importance of a pixel. We define the following quantities: Given two pixels $p = [r_1, g_1, b_1]$ and $q = [r_2, g_2, b_2]$, let

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

Given a picture with width W and height H, let M the matrix that represents the image. Now, given a pixel $M[i, j]$, if $0 < i < H$, its *Yimportance* is define as below:

$$YImportance(M[i, j]) = Dist(M[i - 1, j], M[i + 1, j]),$$

if $i = 0$, then

$$YImportance(M[i, j]) = Dist(M[H - 1, j,]M[i + 1, j]),$$

and if $i = H - 1$

$$YImportance(M[i, j]) = Dist(M[i - 1, j,]M[0, j]).$$

Similarly, given a pixel $M[i, j]$, if $0 < j < W$, its *XImportance* is defined as below:

$$XImportance(M[i, j]) = Dist(M[i, j - 1], M[i, j + 1]),$$

if $j = 0$, then

$$XImportance(M[i, j]) = Dist(M[i, W - 1,]M[i, j + 1]),$$

and if $j = W - 1$

$$XImportance(M[i, j]) = Dist(M[i, 0]M[i, j - 1]).$$

Finally,

$$Importance(M[i, j]) = XImportance(M[i, j]) + YImportance(M[i, j])$$

Now, coming back to our problem of reducing the image width, we will apply the following procedure to reduce width from W to W-1.

- Compute a Matrix named I, where $I[i, j] = Importance(M[i, j])$.
- Compute Min-Cost vertical cut of I. Let it be $0, y_0, 1, y_1, \cdots, H - 1, y_{H-1}$.
- For every i, remove the pixel $M[i, y_i]$ from the image. Now the width of the image is $W - 1$.

To reduce the width from W to $W - k$, repeat the above procedure k times.

2.1 Your Task

You will use the above idea to reduce width of images. Write a class named `ImageProcessor`.

`ImageProcessor(String imageFile).` Parameter `imageFile` holds the name of the image that will be manipulated. We will use W to denote the width and H to denote the height of the image.

`reduceWidth(double x)` Returns a new `Picture` whose width is $\lceil x \times W \rceil$. Note that the type of this method must be `Picture`. Your method **must** use the algorithm described earlier to reduce the image width. And your method must use the **static method minCostVC** to compute the vertical cut. Otherwise, you will receive **zero** credit.

You are provided a class named `Picture` that will enable you to manipulate images. The `Picture` class is designed by Robert Sedgewick and Kevin Wayne. This class can manipulate `gif`, `jpg` and `png` images.

Remark. Please note that the addressing mechanism in the `Picture` class differs from the standard convention, $Pixel[x, y]$ refers to the pixel in xth column and yth row.

3 Specifications

Your program must strictly adhere to specifications. The class, package, and method names must be exactly as specified. The return types of methods must be exactly as specified. Types of parameters to the methods must be exactly as specified. Failure to follow specifications will result in a lower grade even if your program works correctly.

You should not use any external libraries/packages, except `Picture.java`. You are not allowed to modify `Picture.java`.

Your code must be well commented. Your grade will depend on : Correctness of the program, Report, Efficiency of the program. If you do not use **Dynamic Programming** for the methods `stringAlignment` and `minCostVC` you will receive **zero** credit.

4 Report

For the methods `minCostVC` and `stringAlignment` write the recurrence relation. Derive the run time of your algorithms.

5 What to Submit

- DynamicProgramming.java
- ImageProcessor
- Report
- Additional Helper Classes.