Programming Assignment 2
Points: 500
Due: Nov 16, 11:59PM
Late Submission Nov 18, 11:59PM (5% penalty)

0 Preamble

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. You can work in group of size at most 2. It is your responsibility to find a classmate (taking this course with you) with whom you can work effectively. Your programs must be in Java, preferably Java 8.1.

1 Q1: Shortest Path Problems

We have covered single source shortest path problem and used Dijkstra's algorithm to address the problem in $O((|V| + |E|)lg(V))$ time for a graph $G = (V, E, wt)$ (V is the set of vertices, E is a set of edges and $wt : E \to \mathtt{N}$). In this assignment, you will write algorithms to solve different variants of shortest path problems. We will denote any path as π as a sequence of vertices in the path. We will denote source and destination of π as $src(\pi)$ and $dest(\pi)$. The cost of a path, $cost(\pi)$, is the sum of the weights of the edges in the path.

1. $V2V$ problem: Given vertices u and v, find the shortest path to from u to v. We will refer to such a path as $V2V(u, v)$.

$$V2V(u, v) = \texttt{argmin}\ cost(\pi \mid src(\pi) = u \text{ and } dest(\pi) = v)$$

2. $V2S$ problem: Given a vertex u and a set of vertices S, find the shortest path from u to some vertex in S. We will refer to such a path as $V2S(u, S)$.

$$V2S(u, S) = \texttt{argmin}\ cost(\pi \mid src(\pi) = u \text{ and } dest(\pi) \in S)$$

3. $S2S$ problem: Given a set of vertices S_1 and a set of vertices S_2, find the shortest path from some vertex in S_1 to some vertex in S_2. We will refer to such as path as $S2S(S_1, S_2)$.

$$S2S(S_1, S_2) = \texttt{argmin}\ cost(\pi \mid src(\pi) \in S_1 \text{ and } dest(\pi) \in S_2)$$

1.1 Your Task for Q1

You are required to design a class with the following properties:

1. Classname and constructor: `WGraph`. The constructor of this class will take as input (a String type) a file-name *FName*. It should read the file with the name *FName* from the same directory and create a graph representation from the file data (how you represent the graph will depend on you: adjacency matrix, adjacency list, list of edges, etc. The representation will also indicate what type of member variables and methods you need for the class). The file data is formatted as follows:

   (a) First line contains a number indicating the number of vertices in the graph

   (b) Second line contains a number indicating the number of edges in the graph

   (c) All subsequent lines have five numbers: source vertex coordinates (first two numbers), destination vertex coordinates (third and fourth numbers) and weight of the edge connecting the source vertex to the destination (assume direction of edge from source to destination)

   For example:

   ```
   4
   4
   1 2 3 4 10
   5 6 1 2 9
   3 4 5 6 8
   1 2 7 8 2
   ```

   Numbers in the same line are separated by a single space.

2. Your class should contain three methods with the following signatures:

   ```
   // The pre/post-conditions describes the structure of the
   // input/ouput. The semantics of these structures depend on
   // defintion of the corresponding method.

   // pre:  ux, uy, vx, vy are valid coordinates of vertices u and v
   //       in the graph
   // post: return arraylist contains even number of integers,
   //       for any even i,
   //       i-th and i+1-th integers in the array represent
   //       the x-coordinate and y-coordinate of the i/2-th vertex
   //       in the returned path (path is an ordered sequence of vertices)
   ArrayList<Integer> V2V(int ux, int uy, int vx, int vy)

   // pre:  ux, uy are valid coordinates of vertex u from the graph
   //       S represents a set of vertices.
   //       The S arraylist contains even number of intergers
   ```

```
//       for any even i,
//       i-th and i+1-th integers in the array represent
//       the x-coordinate and y-coordinate of the i/2-th vertex
//       in the set S.
// post: same structure as the last method's post.
ArrayList<Integer> V2S(int ux, int uy, ArrayList<Integer> S)

// pre:  S1 and S2 represent sets of vertices (see above for
//       the representation of a set of vertices as arrayList)
// post: same structure as the last method's post.
ArrayList<Integer> S2S(ArrayList<Integer> S1, ArrayList<Integer> S2)
```

The definitions of these methods follow the problem definitions for $V2V$, $V2S$ and $S2S$ above.

2 Q2: Image Processing

We will consider the problem of resizing images. We will proceed with the notion of a min-cost vertical cut in a 2-D matrix. We will use this notion to address the resizing problem.

2.1 Cuts

Let I 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 $0, 1, 2, \cdots m-1$. A *vertical cut* V of I 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* $V = [x_0, y_0, x_1, y_1, \cdots, x_n, y_n]$ of I is defined as

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

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

$$MinVC(I) = \texttt{argmin}\ Cost_I(V)$$

2.2 Vertical Cut and Image Resizing

Note that an image is 2-dimensional array of pixels. Each pixel has represented as 3-tuple in the RGB format. Given an image of height H and width W, it is represented by a $H \times W$ matrix (2-D array). In this problem, we will assume that H and W are both greater than 1. For example, an image with $H = 3$ and $W = 4$ 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 $M[i, j]$ ($i \in [0, 2], j \in [0, 3]$) 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 most important features of an image even after reducing the width. Consider that for an image of width W, we would like to re-size it to 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? One possible solution would be remove the pixels that are *not important*. How do we decide on importance of a pixel? There are several alternatives, in this assignment we consider the following simple method to decide the importance of a pixel.

Given two pixels $p = [r_1, g_1, b_1]$ and $q = [r_2, g_2, b_2]$, we define

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

Given a picture with width W and height H for any pixel $M[i, j]$, $0 \leq i < H$, its *YImportance* is

$$YImportance(M[i,j]) = \begin{cases} PDist(M[H-1,j], M[i+1,j]) & \text{if } i = 0 \\ PDist(M[i-1,j], M[0,j]) & \text{if } i = H-1 \\ PDist(M[i-1,j], M[i+1,j]) & \text{otherwise} \end{cases}$$

Similarly, given a pixel $M[i, j]$, $0 \leq j < W$, its *XImportance* is

$$XImportance(M[i,j]) = \begin{cases} PDist(M[i,W-1], M[i,j+1]) & \text{if } j = 0 \\ PDist(M[i,j-1], M[i,0]) & \text{if } j = W-1 \\ PDist(M[i,j-1], M[i,j+1]) & \text{otherwise} \end{cases}$$

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 $MinVC(I)$. Let it be $V = [x_0, y_0, x_1, y_1, \cdots, x_{H-1}, y_{H-1}]$. **You must compute the Min-Cost vertical cut using some variant of shortest path problems (from Q1).**
- For every x_i, y_i in the vertical cut V, remove the pixel $M[x_i, y_i]$ from the image. Now the width of the image is $W - 1$.

To reduce the width from W to $W - k > 1$, repeat the above procedure k times.

2.3 Your Task for Q2

You are required to design a class with the following properties:

1. Classname and Constructor: `ImageProcessor`. The constructor of this class takes as input (a String type) a filename FName. The file with this name will be read from the same directory to obtain the data related to pixels of some image. The file data is formatted as follows:

   (a) First line contains the height H of the image as a number

   (b) Second line contains the width W of the image as a number

   (c) All subsequent lines contains pixel values at $M[i,j]$ ($0 \leq i < H$, $0 \leq j < W$). For instance,

   ```
   3
   4
   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
   ```

   The numbers in the same line are separated by a single space. You can decide the way you want to represent the matrix M.

2. Methods

   (a) Compute Importance matrix: The matrix I capturing the importance values for each element in M

   ```
   // pre:
   // post: returns the 2-D matrix I as per its definition
   ArrayList<ArrayList<Integer>> getImportance()
   ```

   (b) Compute the reduced image (reduction in width by k) and write the result in a file

   ```
   // pre:  W-k > 1
   // post: Compute the new image matrix after reducing the width by k
   //       Follow the method for reduction described above
   //       Write the result in a file named FName
   //       in the same format as the input image matrix
   void writeReduced(int k, String FName)
   ```

3 Postscript

1. You shall use default package. Though it is not recommended in practice, you can write all your classes for each problem in one file. For the Q1, you will submit `WGraph.java` and for Q2, you will submit `ImageProcessor.java`.

   Submit the `README.txt` including the names and netids of all group members (alphabetical order by last name):

   ```
   Lastname1 FirstName1 netid1
   Lastname2 FirstName2 netid2
   ```

2. You must follow the given specifications. Method names, classnames, return types, input types. Any discrepancy may result in lower than expected grade even if "everything works".

3. In the above problems, there are several data structure/organization that are left for you to decide. Do not ask questions related to such data structure/organization. Part of the exercise to understand and assess a good way to organize data that will allow effective application of methods/algorithms.

4. In Q2, you will have to think about how to model the problem into a graph-based problem and apply your knowledge of graph algorithms to address the original problem. Do not ask questions about how to model the problem as a graph-based problem and/or what graph algorithms to use (though in this assignment, it is clear that we are looking to apply shortest path related algorithm). Part of the exercise is to understand and assess a good way to represent/reduce a problem to a known problem for which we know an efficient algorithm.

5. Start reading and sketching the strategy for implementation as early as possible. That does not mean starting to "code" without putting much thought on what to code and how to code it. This will also help in resolving all doubts about the assignment before it is too late. Early detection of possible pitfalls and difficulties in the implementation will help in reducing the finishTime-startTime for this assignment.

6. Both correctness and efficiency are important for any algorithm assignment. Writing a highly efficient incorrect solution *will* result in low grade. Writing a highly inefficient correct solution *may* result in low grade. In most cases, the brute force algorithm is unlikely to be the most efficient. Use your knowledge from lectures, notes, book-chapters to design and implement algorithms that are correct and efficient. In addition to typical correctness validation, your implementation will be tested on at least one large graph (and/or image).

7. If you would like to generate pixels from real images, you can use the following code available at `https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Picture.java.html` with documentation at
`https://algs4.cs.princeton.edu/code/javadoc/edu/princeton/cs/algs4/Picture.html`