For this assignment you will implement a curve finding algorithm using shortest paths.

You should turn in your source code and a brief PDF writeup. Please turn in a ZIP file via email with the subject “ENGN1610 HOMEWORK 3”.

Your writeup should include example results computed using your implementation. The writeup should also explain how you built your graphs, including the approach used for defining the weights of the edges.

**Part 1**

Write a function to find a boundary curve connecting two selected pixels in an image. Your function should take as input an image and the coordinates of two pixels. It should return an image where the curve pixels are painted white.

The figure below shows an example result.

![Example Result](image)

Your program should find curves by building an appropriate graph and computing shortest paths between pairs of points. To compute shortest paths you can use two functions provided with the assignment:

\[
\begin{align*}
\text{[dist,prev]} &= \text{shortestpaths}(E,s,k) \\
p &= \text{trace}(\text{dist,prev,v})
\end{align*}
\]

The first function computes shortest paths taking at most \(k\) steps from vertex \(s\) to every other vertex in a graph with edge set \(E\). The second function traces the shortest path ending at vertex \(v\). It returns the path as an array of vertices.
The vertices of the graph are represented by integers from 1 to the length of $E$. $E$ is a cell-array defining the graph using adjacency lists. The set of edges leaving vertex $i$ is defined by a $m \times 2$ array $E\{i\}$, where $m$ is the number of edges leaving vertex $i$. The first column of $E\{i\}$ specifies the destination vertices of the edges leaving $i$, and the second column specifies the weights of the edges.

Here is the edge structure for an example graph with 3 nodes and 4 edges:

\[ E\{1\} = \begin{bmatrix} 2 & 0.5 \end{bmatrix}; \]
\[ E\{2\} = \begin{bmatrix} 1 & 0.3; 3 & 0.8 \end{bmatrix}; \]
\[ E\{3\} = \begin{bmatrix} 2 & 0.7 \end{bmatrix}; \]

Note: When finding a curve in an $N \times M$ image you can set $k = N + M$ in practice. This may not find the absolutely shortest paths between two pixels because it excludes paths that take many twists, but that is not a problem for our application. Setting $k = N \times M$ would ensure we always find the absolutely shortest path but will be too slow.

For an $N \times M$ image you will need to define a mapping from image pixels to integers from 1 to $N \times M$ to interface with the shortest paths algorithm. You should construct a graph from an image, using 4 or 8 neighbors for each pixel. Note that the graph should be directed, so there will be two edges between each pair of connected pixels. The weight of the edges should encourage shortest paths to move along the boundaries of objects.

You can use gradient magnitudes to define weights. In this case the weight of an edge should be low if the edge goes through pixels with high gradient magnitude. The weight should be high if the edge goes through pixels with low gradient magnitude. This will encourage shortest paths to go through high gradient regions of the image.

**Part 2**

In some applications involving elongated objects instead of finding boundaries we are interested in finding centerlines. For example, consider the problem of tracing an artery in a medical image, illustrated in the result below.
We can solve this problem with a shortest path computation, but now the edge weights should attract the path towards the center of the artery instead of the boundary.

For this part of the assignment you should construct a weighted graph to find the center-lines of arteries in images like the one above. A good approach for detecting center points in an object of fixed width involves using difference-of-gaussian filters, where the standard deviations of the gaussians are selected based on the object width.

**Part 3 OPTIONAL**

Make an interactive tool to segment an object by selecting a few points along the boundary. You can use the function ginput in matlab to make an interactive function for selecting endpoints of a curve.

**Part 4 OPTIONAL**

Improve the implementation of the shortest paths algorithm to make it faster. Alternatively you can implement Dijkstra’s shortest paths algorithm. With the proper data structures Dijkstra’s algorithm is much faster than the Bellman-Ford algorithm.