Homework 7

Mountain Paths - Part III

In this homework, you will extend part II to paint all paths starting on each row. As you do this, you will keep track of which row has the shortest path. Once you have identified the shortest path, you will color it a different color.

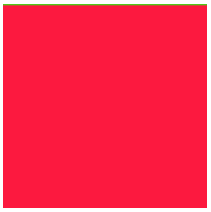
Requirements

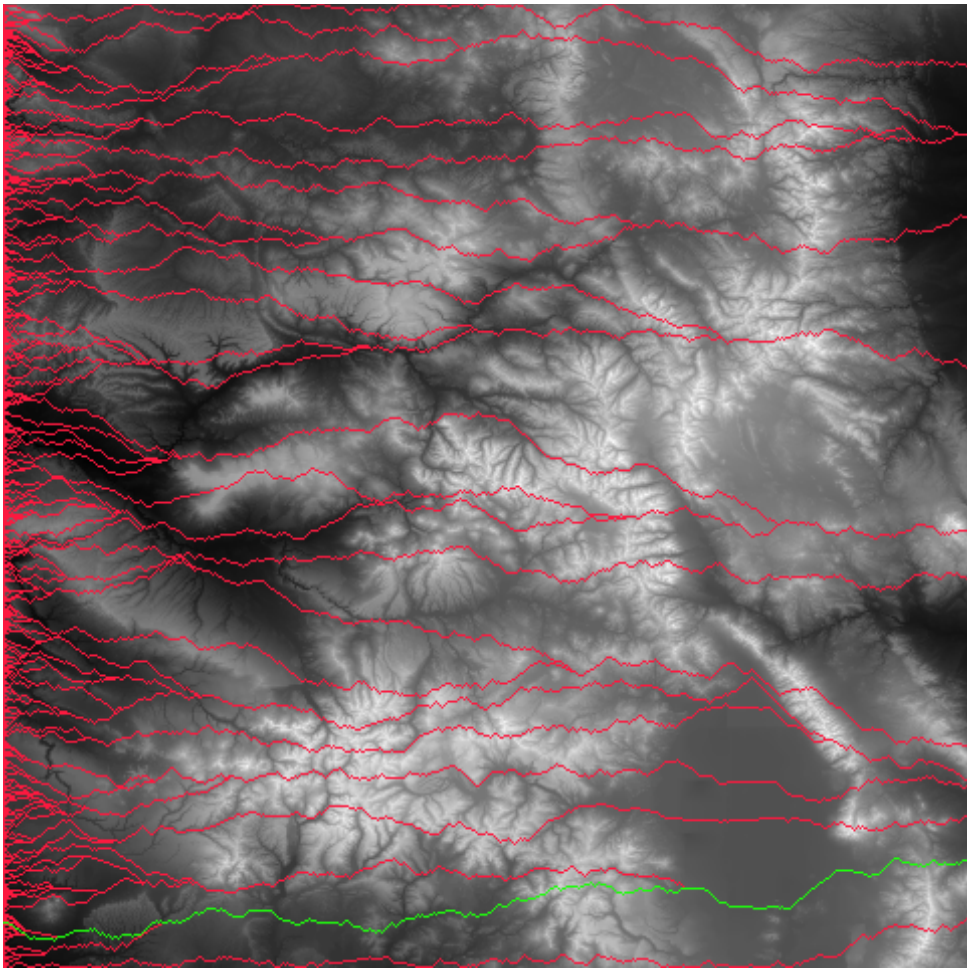
You implemented (as required by Part I) code that read the topographical data from an input file, manipulated it to identify the appropriate gray scale value to display it, and generated an output file following a simple format to represent images.

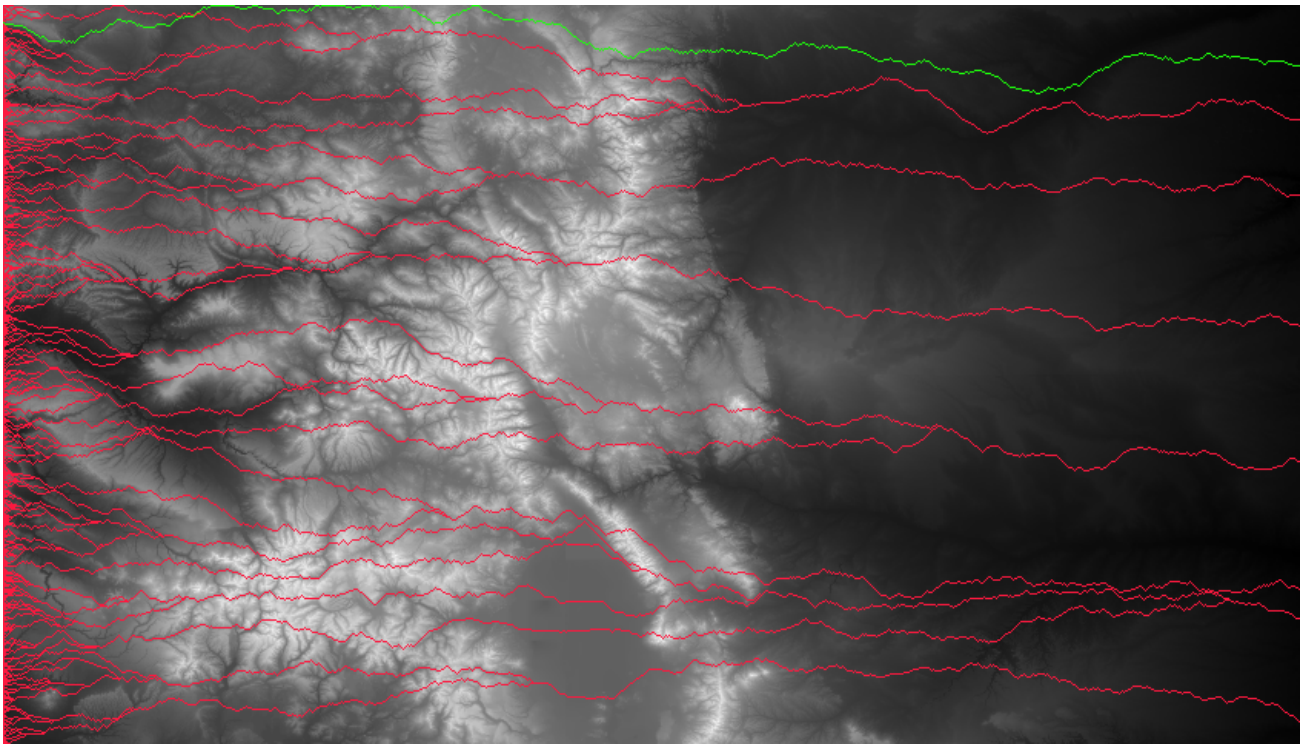
Then you created a function (as required by Part II) to compute path a path for a row following the policy of “path of least resistance”.

Now, rather than only painting three paths, you will paint the paths for each row. As you do that, keep track of which path is the shortest. Once you have painted all of the paths, you will paint the shortest path again with a different color.

The output file generated by your program will produce a visualizationsimilar to the ones depicted below. These are based on the sample input files from part I.







Program Flow

Your program starts by executing the first 3 steps from part I:

1.Read the data into a 2D array

2.Find min and max elevation to correspond to darkest and brightest color, respectively

3.Compute the shade of gray for each cell in the map

In part I, you completed the assignment by producing the output file (image file in the PPM format) and used a tool to look at the visualization of your output.

For part III, your new steps are:

4.Instead of painting 3 paths as you did in part II, loop through each row and calculate the distance and paint the path red [RGB(252,25,63)] using the function you created in part II (`colorPath()`). During the loop keep track of which row has the shortest path. If more than one path is the shortest, then use row with the lowest index.*Note: The autograder will still check if the function matches the requirements from part II.*

5.Paint the shortest path green [RGB(31,253,13)] .

For extra credit: (NOT REQUIRED)

See details below. If you do not do this, your grade will not be affected negatively.

6.Read in two integers for row and column respectively.

7.Call a function that will paint the shortest path to the edge of the map going north, south, east, and west using a greedy algorithm. The color will be aqua [RGB(19,254,253)] .

Function prototype given below.

Then you conclude by carrying out steps you already implemented in part I:

8.Produce the output file in the specified format (PPM)

9.Use an online free tool to convert your PPM file into a JPG file

Extra Credit

For extra credit, after printing all paths and the shortest one, read two integers representing row and column from input.

Note: to prevent your program from hanging in the regular cases, you will need to check if there is still input in cin, if You will then use a greedy algorithm as we did in Part II to paint four paths from the row, column location. Note, that your tiebreakers will vary according the the direction. Prefer forward over other directions, and forward and right after that.

Direction	Tiebreaker 1	Tiebreaker 2
north	N	NE
south	S	SW
east	E	SE
west	W	NW

The function signature is:

```
int colorPath(const vector
```

This is similar to before, but will also specify which col to start on.

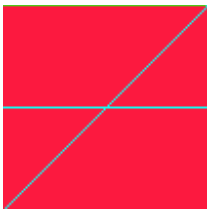
To prevent your program from hanging when testing regular test cases, you will need to see if there is more data in cin. If there is then you can get the row and column and proceed to call the extra credit function. Your code will look something like:

```
if(cin >> ec_row >> ec_column) {  //ec for extra credit!

  //call function to paint paths for point

}
```

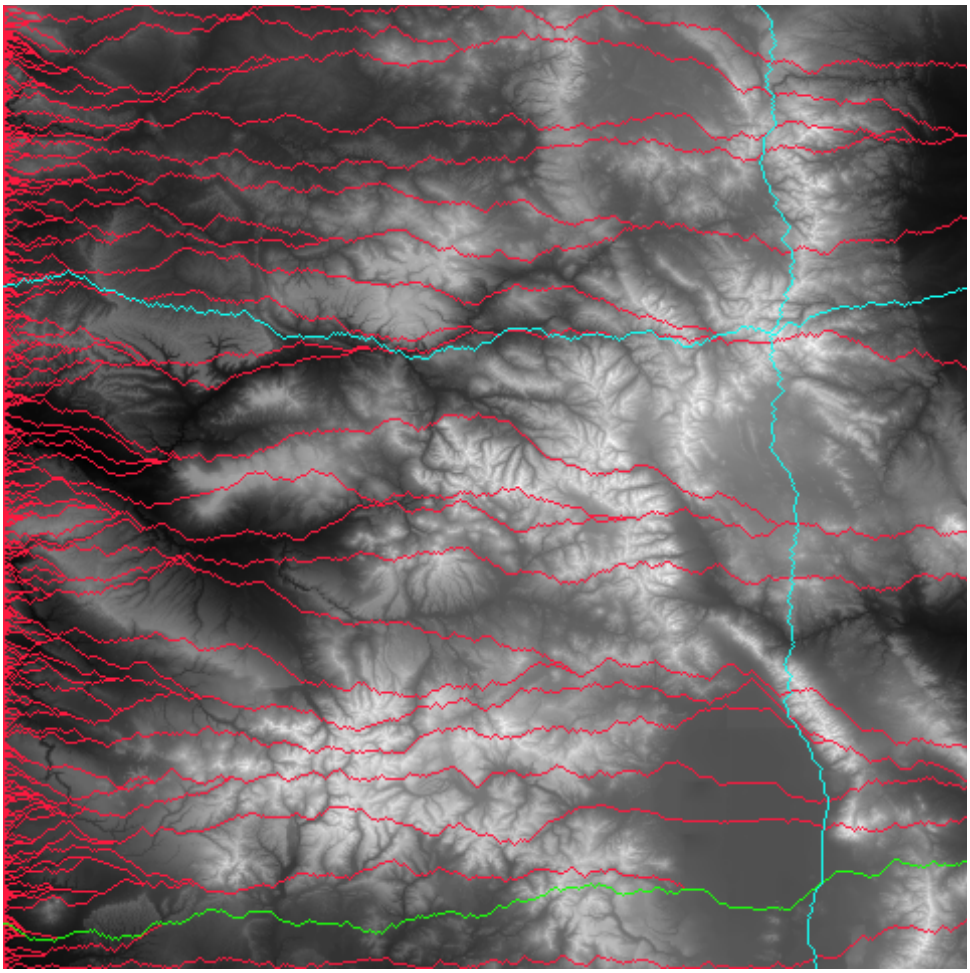
So when testing, after you type in the filename also input the row and column on the same line before you hit enter. As long as you are not using getline() to get the filename, this will work.

Sample outputs below are based on the sample files from part one with the following (row, column) values.

(50,50)



(164, 382)



(164, 388)