# CIS 260 – Introduction to Programming
# Spring 2023
# Instructor: Dr. Jackie F. Woldering

## Programming Assignment 2
***Due at 11:59 pm on Sunday, April 16, 2023***
***Please submit your answers on Blackboard***

### The Problem
This is the second of a two-part assignment (you completed the first part in Assignment 1), where you will read a set of topographic (land elevation) data into a 2-dimensional array and write some methods to compute some paths through the mountains as well as visualize them.

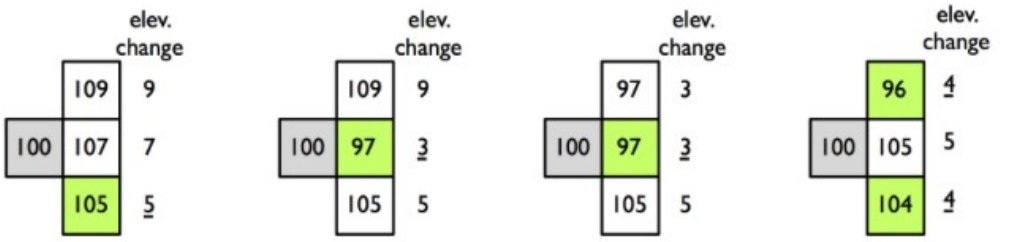
### Background
There are many contexts in which you want to know the most efficient way to travel over land. When traveling through mountains (let's say you're walking) perhaps you want to take the route that requires the least total change in elevation with each step you take – call it the path of least resistance. Given some topographic data it should be possible to calculate a "greedy lowest-elevation-change walk" from one side of a map to the other.

You've already completed the task of drawing the map. I am providing you with a complete correct solution for the map drawing, or you can use your own working MapDataDrawer class from Assignment 1. For assignment 2, I am also providing you with the complete `Driver.java` code you will use today. The only piece of work for you is to complete the `drawLowestElevPath` method in `MapDataDrawer`.

**Step 1:** Figure out the starting point of the walk across the mountain, by identifying the lowest elevation in column 0.

**Step 2:** Now the task is to begin the walk across. We are going to use a "greedy" algorithm, and take one step at a time. Since our map is in a 2D grid, we can envision a "walk" as starting in some in some cell at the left-most edge of the map (column 0) and proceeding forward by taking a "step" into one of the 3 adjacent cells in the next column over. Our "greedy walk" will assume that in your walk you will choose the cell whose elevation is closest to the elevation of the cell you're standing in. (NOTE: this might mean walking uphill or downhill).

The diagrams below show a few scenarios for choosing where to take the next step. Assume that in the case of a tie, you can go in any of the tied directions.




Case 1: smallest change is 5. Go Down.

Case 2: smallest change is 3. Go forward.

Case 3: Tie. Smallest change is 3. Forward is possible so go forward.

Case 4: Tie. Smallest change is 5. Flip a coin to go up or down.

Here is an example of a small segment of the data grid. The cells marked in green shows the greedy path.

| | | | | | |
|---|---|---|---|---|---|
| 3011 | 2900 | 2852 | 2808 | 2791 | 2818 |
| 2972 | 2937 | 2886 | 2860 | 2830 | 2748 |
| 2937 | 2959 | 2913 | 2864 | 2791 | 2742 |
| 2999 | 2888 | 2986 | 2910 | 2821 | 2754 |
| 2909 | 2816 | 2893 | 2997 | 2962 | 2798 |

Shows a portion of the data.
Greedy path shown in green.

**Detailed Instructions**

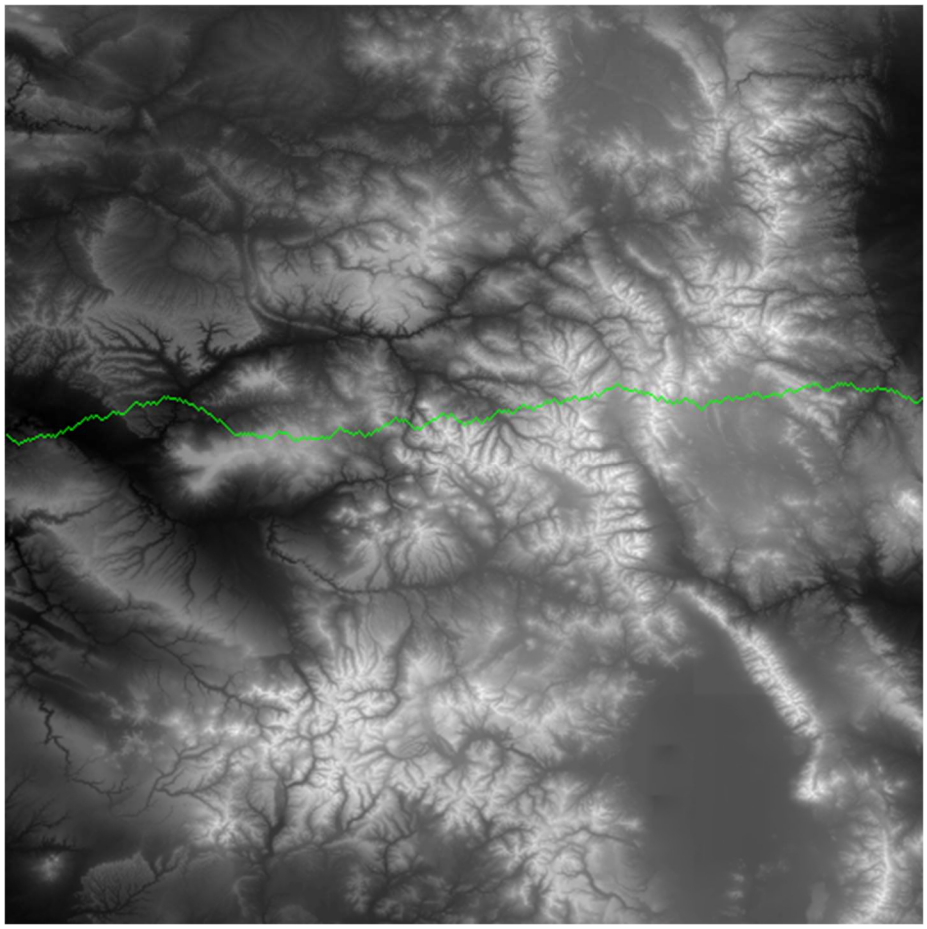
Implement the `drawLowestElevPath(Graphics, startRow)` method. Note that this method does two things: 1) it draws the path, 2) it calculates and returns to the total elevation change on that path. The method should draw a line by drawing 1x1 filled rectangles in a different color on top of the existing drawing. The path should be drawn going West-to-East, starting from the given row using the greedy lowest-elevation-change technique described in earlier pages.

You will need to do the following things in some order.

1. Starting from the given row, and column 0, color it red (already done in the Driver class).
2. Write a loop that generates every possible column across the map from 1 to 480 (you start at column 0, but column 1 is the first column you need to make choice about where to step). For each column you will decide which row to take your next "step"

to – fwd, fwd-and-up, or fwd-and-down – using the greedy choice strategy described.

3. Use a variable that keeps track of the 'current row' you're on, and update it each time you take a step forward – row may stay the same, or go up or down by 1 depending how you walk.
4. Use `Math.abs(...)` to get the absolute value of the difference between two elevations.
5. Continue finding the lowest neighboring cell and coloring it green as you go.
6. Keep a running total of the total elevation change that would be 'experienced' by a person walking this path. Since we consider an elevation change the absolute value (i.e. going 'uphill' 10 meters is the same amount of change as going 'downhill' 10 meters) this running total will be non-decreasing and will end up being a pretty large positive number
7. When you're done you should see a line tracing the lowest elevation-change path from west to east. Here is the image you should see (path may vary slightly due to equivalent choices):



**Submission Instructions**

- You will write code for all three parts in a single Java file. Please name the class `MapDataDrawer_LastName`.
- Please document your code with comments. Please explain each major action you are performing in your program.
- Please make your code *readable*. Please use the pointers I've given you in class. For example,
    - do not use single letter variable names, instead, name the variables meaningfully
    - use braces and parentheses liberally
    - use proper indentation
- Submit the java program on Blackboard.

***Grading***

- The program must compile for you to get a non-zero grade.
- The readability of your code is an important contributor to your final grade.
- Please DO NOT CHEAT! I want to see YOUR WORK, not your Google search skills. I take cheating very seriously, and will deal with cheating harshly. Do not risk it!
- This assignment is worth 5% of your final grade.