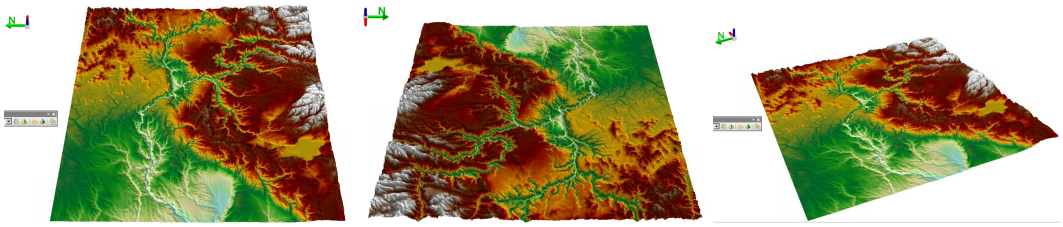
Assignment 1

Please read this document very carefully. Follow instructions exactly. If you have any questions please post them to MS Teams or ask during office hours.

This assignment is due Oct. 16th, by 11:59pm

Consider a section of the Earth's surface from a birds-eye-view. The elevation may vary greatly across this section. See the Figures below as an example. If one were interested in analysing this area based on elevation, one could represent the section of Earth as a matrix of numerical values, where each cell of the matrix would indicate the elevation of the Earth at that particular location. For example, an area the size of 1 square kilometer of Earth could be encoded as a 1000x1000 matrix, where each cell is the elevation of a single square meter; here, the value at cell (500,500) of the matrix, would be the elevation at approximately the middle of the original 1 square kilometer section.



For this assignment, you will implement several methods which will allow someone with such a matrix of values to perform meaningful analysis on the area of the Earth's surface the matrix represents. You are given a file, `Assignment1.java`, with four incomplete methods. For this assignment, **you're required to complete these methods**. A description regarding the intended behaviour of these methods is given later in this document. **DO NOT RENAME THE FILE, CLASS, OR METHODS.** If you do, you risk getting 0 on this assignment do matter how much work you put into the actual code.

For the purposes of this assignment we will use the following definitions.

1. An *elevation map* is of the type `int[][]`, and moreover, the length of an elevation map equals the length of all elements within the elevation map. An elevation map will only contain positive numbers. An example of an elevation map is:

   ```
   valid_map = [[1,2,3],[4,5,6],[7,8,9]]
   ```

   It may be more intuitive to view the map as:

   ```
   valid_map = [[1,2,3],
                [4,5,6],
                [7,8,9]]}
   ```

   The following three examples are not elevation maps (note the length of all the arrays in each):

   ```
   invalid_map1 = [[1,2,3],[4,5],[6,7,8,9]]
   invalid_map2 = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
   invalid_map3 = [[]]
   ```

2. A *cell* is of the type `int[]` and has a length of 2. All values in a cell will be greater than or equal to 0. Within an elevation map m, we say cell $[i, j]$ as a shorthand for $m[i][j]$. We also say cell $[i, j]$ is adjacent to cell $[n, m]$ if and only if $[n, m]$ equals one of:

$[i+1, j+1], [i+1, j], [i+1, j-1], [i, j-1], [i, j+1], [i-1, j+1], [i-1, j]$, or $[i-1, j-1]$.
This can be seen visually below.

[i – 1, j – 1]	[i – 1, j]	[i – 1, j + 1]
[i, j – 1]	[i, j]	[i, j + 1]
[i + 1, j - 1]	[i + 1, j]	[i + 1, j + 1]

3. Within an elevation map m, cell $[i, j]$ is a *sink* if for all adjacent cells $[n, m]$, $m[i][j] \leq m[n][m]$. With the physical interpretation of an elevation map in mind, water would collect in sinks, since there is no less elevated area in the immediate vicinity for the water to flow to.

Functions

You are required to implement all of the functions below. Pay attention to parameters of each function, for example, if it is said an input will be an elevation map, you can trust your function will never be tested on input which isn't an elevation map. For further examples of how these functions are intended to operate, view the docstrings of the starter code for this assignment.

1. `findPeak(int[][]) -> int[]`
   The first parameter is an elevation map, m. Returns the cell which contains the highest elevation point in m. **For the purposes of us testing this function, you may assume that all values of m are unique (no two locations have equal elevations).**

2. `isSink(int[][], int[]) -> boolean`
   The first parameter is an elevation map, m, the second parameter is a cell, c. Returns true if and only if c is a sink in m. Note if c does not exist in m (the values are outside m's dimensions), this function returns false. See the previous section for the definition of a sink.

3. `findLocalSink(int[][], int[]) -> int[]`
   The first parameter is an elevation map, m, the second parameter is a cell, c, which exists in m. Returns the local sink of c. A local sink of c is the cell which water would flow to if it started at c. Assume if the current location isn't a sink, water will always flow to the adjacent cell with the lowest elevation. **You may also assume for the purposes of us testing this function, that all values of m are unique (no two locations have equal elevations).**

   For example, if the elevation map is

   ```
   [[1, 2, 3],
    [9, 8, 7],
    [4, 5, 6]]
   ```

   and the cell is [1, 1] the function should return [0, 0], however, if the cell is [2, 2] the method should return [2, 0].

4. `rotateMap(int[][]) -> void`
   The parameter is an elevation map, m. Under the interpretation that the top of m is north, the function mutates m such that the top of m would now be viewed as east. See the following as an example:

```
[[1, 2, 3],
 [9, 8, 7],
 [4, 5, 6]]
```

should become

```
[[3, 7, 6],
 [2, 8, 5],
 [1, 9, 4]]
```

Submitting and Grading

This assignment will be submitted electronically via Avenue. This assignment is worth 14% of your final grade. Grading is done automatically. That is, a program calls your method, passes it certain arguments, and checks to see if it returns the expected output. The assignment is broken down as follows:

- findPeak: 20%
- isSink: 20%
- findLocalSink: 30%
- rotateMap: 30%

Your assignment is only graded on correctness. For any one function, if you pass n of the m tests we run on that function, your grade for that function will be n/m. Your assignment is not graded on efficiency. Your assignment is not graded on style, eg. comments.

Good luck!

Academic Dishonesty Disclaimer

All of the work you submit must be done by you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. The department uses software that compares programs for evidence of similar code.

Please don't copy. The TAs and I want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

Never look at another assignment solution, whether it is on paper or on the computer screen. Never show another student your assignment solution. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in involve students who have never met, and who just happened to find the same solution online. If you find a solution, someone else will too.