# Beijing-Dublin International College
# COMP1004J - Introduction to Program Construction 1
# Assignment: Mountain Paths

## 1 Assignment Details

**Due date:** $11^{th}$ of December 2020
**Worksheet Weight:** 20%
**Submission Requirements:** Submit a zip file on moodle containing the following:

- A zip file containing the source code files that completes the task
- A pdf report (maximum of one page)

## 2 Topic

In this assignment you will read a set of topographic (land elevation) data into a 2D list and write some functions to compute paths through the mountains.

There are many contexts in which you want to know the most efficient way to travel over land. When travelling 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 the bottom of a map to the top.

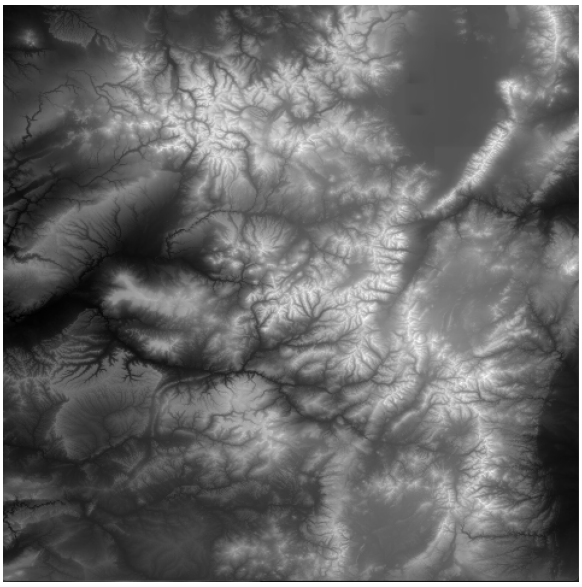


Figure 1: Visual representation of the Colorado data

Figure 1 show a representation of one of the input files that are a part of the assignment. The height at each point on the map is used to determine the colour, the tallest point on the map will be coloured white, and the lowest point will be coloured in black. All other points in between are shaded between these colours.

## 3 Input Files

The assignment will include the following files:

- `wicklow.mp` - A 100 by 100 grid
- `beijing.mp` - A 200 by 200 grid
- `mecca.mp` - A 300 by 300 grid
- `colorado.mp` - A 480 by 480 grid
- `init.py` - Code to create variables to represent the Wicklow map file as a 2D list

### 3.1 Data Input Format

The elevation is specified by a file containing integer values. The first line contains two integer values telling us the width (number of columns) and the height (number of rows) remaining in the file. Each of the remaining height lines in the file will each contain width integer values representing the elevation at one point on the map.

```
1 10 5
2 1  2  3  4  5  6  7  8  9  10
3 11 12 13 14 15 16 17 18 19 20
4 21 22 23 24 25 26 27 28 29 30
5 31 32 33 34 35 36 37 38 39 40
6 41 42 43 44 45 46 47 48 49 50
```

(a) Example of Data layout

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

(b) How this data is laid out in the Viewer

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (0,4) | (1,4) | (2,4) | (3,4) | (4,4) | (5,4) | (6,4) | (7,4) | (8,4) | (9,4) |
| (0,3) | (1,3) | (2,3) | (3,3) | (4,3) | (5,3) | (6,3) | (7,3) | (8,3) | (9,3) |
| (0,2) | (1,2) | (2,2) | (3,2) | (4,2) | (5,2) | (6,2) | (7,2) | (8,2) | (9,2) |
| (0,1) | (1,1) | (2,1) | (3,1) | (4,1) | (5,1) | (6,1) | (7,1) | (8,1) | (9,1) |
| (0,0) | (1,0) | (2,0) | (3,0) | (4,0) | (5,0) | (6,0) | (7,0) | (8,0) | (9,0) |

(c) The x and y coordinates of each square in the viewer

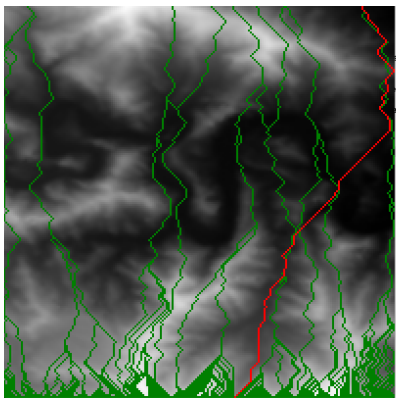
Figure 2: Layout of data and representation in the Viewer

Figure 2a shows an example of the layout of each of the files. The first line of map data (line 2) contains all of the values where the y coordinate is 0. This means that this will be at the top of the file, but on the bottom when visualised using the Topology Viewer.

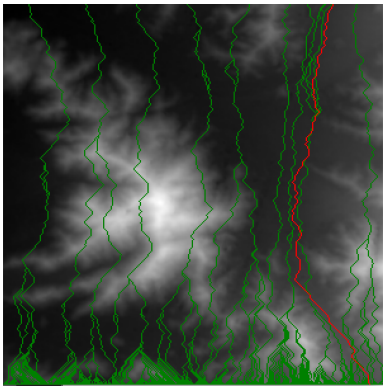
Figures 2b and 2c show where these values will be shown when visualised and the coordinates that we should use when we are calculating and outputting the calculated routes though the mountains.

It can be particularly troublesome to calculate the routes because, the indexes of the 2D list may be in the opposite order to the coordinates in output. I would suggest that you plan and test your algorithm on a very small example to make sure that you are representing the data correctly.

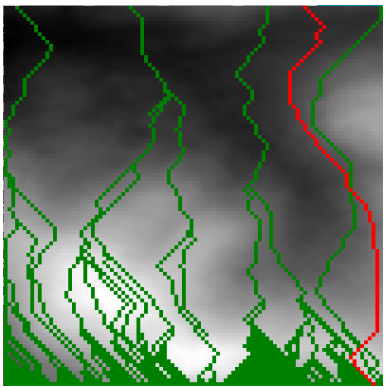
Figure 3 shows 3 of the input files from the assignment shown using the topographic map viewer. The greedy algorithm routes are shown in green and the best route with the lowest change in elevation is shown in red. This is an example of what you should see when you have completed the assignment, however, as there is a random element to the algorithm the results will not always be identical.



(a) Mountains west of Beijing



(b) Mountains South East of Mecca



(c) Mountains in Wicklow

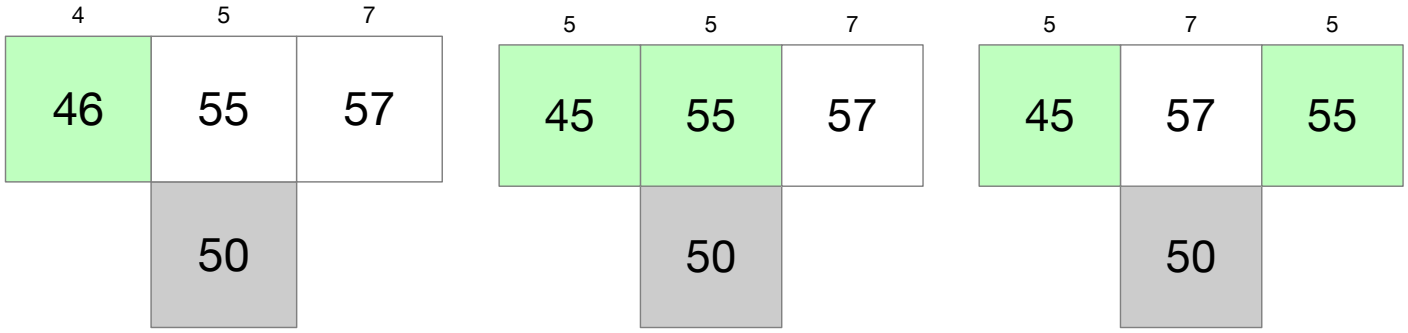
Figure 3: Different sized input files for Assignment

# 4 A Greedy Walk

A "greedy" algorithm is one in which, in the face of too many possible choices, you make a choice that seems best at that **moment**. For the largest map we are dealing with there are too many possible paths you could take starting from one bottom of the map and taking one step forward until you reach the top.

Since our map is in a 2D grid, we can envision a "walk" as starting in some in some cell at the bottom-most edge of the map (row 0) and proceeding forward by taking a "step" into one of the 3 adjacent cells in the next row up (row 1). Our "greedy walk" will assume that 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. In the case of a tie with the forward position, you should always choose to go straight forward. In the case of a tie between the two non-forward locations, you should choose randomly where to go.




(a) Single Choice (b) Equal with Center (c) Equal Sides

Figure 4: Choices when navigating the mountains

Figure 4a shows the easiest circumstance to decide in. In this case of the three squares in front of us, we choose the one with the smallest change in elevation. It should be noted that we are not concerned about the sign of the elevation change, we use the absolute value when we are calculating our next step and also when calculating the total elevation change in a path.

Figure 4b shows how we choose when there are several options with the same change in elevation. If one of the options is the square straight ahead, then we choose the square straight ahead.

Figure 4c shows how we choose when there are two options with the same change in elevation, but neither of these options are the square straight in front of us. In this case, we choose randomly between the two options. This can be done by generating a random number and choosing based on the value of the number.

# 5 Assignment Requirements

The following are the requirements of the assignment:

1. Write the code to read the data as described above from a file and store it in a 2D list
2. Implement the greedy algorithm described above to find a path from the bottom of the map to the top when starting from a particular column (value of x) in the map and output the generated route and its total elevation change
3. Use the greedy algorithm to each column (value of x) in the map and output the total elevation change and route to a file for each starting position (every value of x)
4. Find which of the results generated above has the smallest total elevation change and output this and the route to a file

# 6 Data Output Format

In order for your calculated results to be displayed correctly, they must be in the correct format. Additionally, the data must be stored in a text file with the extension `".rt"` E.g. `"Beijing.rt"`. If you do not name your files correctly, the system will not open them.

## 6.1 A Single Route File - The best Result

The information for a single route must be specified on a one line. The line should contain the total elevation change followed by the x coordinates of all steps in the route. You should not include the y coordinates as the system can calculate these automatically, instead you should submit only the x values starting from the bottom of the image (the top of the file).

```
1 36 4 3 2 1 0
```

This shows an example of the format for a single route across the example data above. The route then the y coordinates are included is (4, 0) → (3, 1) → (2, 2) → (1, 3) → (0, 4). The total elevation change for this route is 36, you can see the calculations in the following table:

| Coordinate | Elevation | Change | Total |
|---|---|---|---|
| (4, 0) | 5 | 0 | 0 |
| (3, 1) | 14 | +9 | 9 |
| (2, 2) | 23 | +9 | 18 |
| (1, 3) | 32 | +9 | 27 |
| (0, 4) | 45 | +9 | 36 |

### 6.2 All Routes File

All of the calculated routes should be specified in a single file. Each line of the file should be in the same format as above. The following file gives an example of the routes generated for the example data file.

```
40 0 0 0 0 0
39 1 0 0 0 0
38 2 1 0 0 0
37 3 2 1 0 0
36 4 3 2 1 0
36 5 4 3 2 1
36 6 5 4 3 2
36 7 6 5 4 3
36 8 7 6 5 4
36 9 8 7 6 5
```

## 7 Optimal Solution

One of the ways to get a top grade in this assignment is to also implement an algorithm and algorithm to find the optimal solution. As the greedy algorithm makes choices on only the nearest squares it can sometimes choose a path that is good in the short term, but bad in the long term. Finding the best route can not be done by checking all of the possible routes (this would take billions of years) so we must use a smarter algorithm to achieve the same task.

To complete this task, you will have to do some research. You can consider the following potential approaches to solving the problem:

- The Floyd–Warshall algorithm
- Dynamic programming, similar to the solution of Project Euler - Problem 67

## 8 Additional Approaches

An alternative way to get a top grade would be to adapt a well known path finding algorithm to work for the mountain example and find paths with the least elevation change. The following are potential algorithms that could be implemented:

- Dijkstra's Algorithm
- A* Search Algorithm
- D* Algorithm

These are graph algorithms and will take a little bit of planning and thinking in order to adapt them to this problem.

## 9 Marking Scheme

| Grade | Expectation |
|---|---|
| D | Correctly implement the greedy walk algorithm for finding the path with the least total elevation change. This should function of the test list that is supplied. |
| C | All of the above and implement code to read the map data from the files in their current format and output the results in the correct format too. The program should ask the user to enter the name of the map file they wish to analyse. |
| B | All of the above and calculate the greedy path for every starting coordinate on the map and output the results. Additionally, you should output separately the path with the smallest total elevation change. |
| A | Implement the an algorithm to find the optimal route through the map, or implement one of the additional approaches. |

### 9.1 Additional Considerations

Achieving the given grades is not guaranteed, the following factors will also be considered when determining your grade:

- Design - Your code should be procedural in nature and broken into functions appropriately
- Names - Variables should be named sensibly
- Code Formatting - Code should be well formatted and indented correctly
- Comments - You code should be lightly documented (explaining intent of the algorithms)
- Report - The contents and quality of the report you submit

# 10 Randomness

In the greedy walk algorithm, you are required to make a random choice when both left and right have the same elevation change and it is less than the elevation change straight ahead. To get make a random choices in python you can use the random module. There are many functions that can be used to generate a random number, but I will give an example of an easy way to make a choice between two options.

```
import random

choice = random.choice( (True, False) )

if choice == True:
  print("Go Left")
else:
  print("Go Right")
```

We import the random module and make use of a function called `choice`. The choice function chooses between the items in a sequence passed as a parameter. In this case I have passed a tuple containing the boolean values True and False, but this could easily be replaced by the strings `"Left"` and `"Right"`. After the function chooses one of the options, we then check which one was chosen using an if statement and execute the correct code.

# 11 Report

Along with the code, you must submit a pdf document describing the code you have attempted and implemented. The report should be a maximum of 1 page and describe the following:

- Your name and UCD student number
- The modules you have broken your code into and the purpose of each
- The functions you implemented and what they do (short description)
- The algorithms you have implemented (or attempted to implement) and if they have worked successfully