Introduction

You are a member of a team developing a mars rover. Your job is to develop the path-finding system and it is crucially important that you minimise energy consumption. Given a digital elevation map of the terrain, you must identify the path between two points of least energy cost.

Assignment Specification

For this assignment, you will be given a digital elevation map (DEM) and an equation for calculating energy use in moving from one elevation to another adjacent elevation. To simplify the task you consider only the discrete positions and elevations represented by the DEM and will consider only moves in the four cardinal directions (NSEW).

You will be provided with code that contains some code to get you started.

```
int** make_dem(int size, int roughness);
```

This code generates a 2D array of ints representing the heights of a square area of the terrain. The size should be one more than a power of 2 (e.g. 9, 17, 33, 65) and roughness values around 4xsize work well. You will also notice that the code sets a random seed – this can be set to a fixed value for testing purposes (to ensure that you always get the same map).

```
int cost_funcA(int diff);

int cost_funcB(int diff);
```

These are two different cost functions for calculating the energy use from moving between two adjacent points in the DEM with the given height difference. The first will be used for part A and will always produce a positive number. The second is used for part B and includes a term for regenerative breaking – hence it can generate a negative value when going downhill.

There are also two functions for printing DEMs and markers:

```
void print_2D(int** array2D, int size);

void print_2D_ascii(int** array2D, int size);
```

These functions take a DEM where heights are expected to be in the range 0-99. Any negative value will be printed as a marker. You should use these functions to visualise your solution (setting the height of positions that are on your path to -1). You can use either print function. The first function gives more detail, but the second is a bit easier to interpret. (**NOTE: these function print with the y axis going across the page and the x axis going down**)

Your first task is to use these functions to generate an adjacency list graph representation of the terrain. Start by generating a DEM, then use the cost functions to build the weighted

graph. The vertex number of a given `x,y` position in the map should be calculated as `x*size+y`.

So, for example, the following DEM:

49 51 51 56 57
46 47 50 52 54
45 47 48 50 49
46 46 47 47 47
45 45 45 45 47

Would result in an edge from vertex 0 to vertex 5 (in bold) with weight calculate using `cost_funcA(46-49)`.

You must use the following data structures to represent the graph as an adjacency list:

```
typedef struct edge{
    int to_vertex;
    int weight;
} Edge;

typedef struct edgeNode{
    Edge edge;
    struct edgeNode *next;
} *EdgeNodePtr;

typedef struct edgeList{
    EdgeNodePtr head;
} EdgeList;

typedef struct graph{
    int V;
    EdgeList *edges;
} Graph;
```

Part A

For part A you will use `cost_funcA` which always produces positive edge weights to generate the adjacency list.

You should then implement Dijkstra's algorithm based on the pseudocode on Wikipedia (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm). While a priority-queue implementation would be best for our scenario, you should instead use a simple search to find the nearest vertex.

Your submission must contain a main function that:

- generates a DEM
- converts it to an adjacency list using `const_funcA`
- calculates the shortest paths from vertex 0 (top left in the DEM)
- reconstructs the path to the last vertex (bottom right in the DEM)
- creates a *copy* of the DEM with the heights changed to -1 for any vertex on the path
- prints the DEM to show the path

Part B

For part B you will use `cost_funcB`, which may produce negative edge weights (but no negatively weighted cycles, to build a new adjacency list.

You should then implement the Floyd-Warshall algorithm based on the pseudocode on Wikipedia (https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm). Your solution must follow this pseudocode as closely as possible (note however, that in our graph specification the vertex labels start at 0).

Your submission must contain a main function that:

- includes all of the code from part A
- calculates a new adjacency list using `const_funcB`
- calculates all shortest paths using Floyd-Warshall
- reconstructs the path from the first to last vertex (top-left to bottom-right)
- creates a *copy* of the DEM with the heights changed to -1 for any vertex on the path
- prints the DEM to show the path

Part B does NOT have to be in a separate project.

A Note on Style

I have not specified file names and code structure. You should endeavour to make your code as general, and hence reusable, as possible. Code should be organised into functions and files that make reuse easy. Functions and data structures should be properly commented so that another programmer will know how to use them.

Assignment Submission

Assignments will be submitted via MyLO (an Assignment 2 dropbox will be created). You should use the following procedure to prepare your submission:

- Make sure that your project has been thoroughly tested using the School's lab computers
- Choose "Clean Solution" from the "Build" menu in Visual Studio. **This step is very important** as it ensures that the version that the marker runs will be the same as the version that you believe the marker is running.
- Quit Visual Studio and zip your entire project folder along with a completed assignment cover sheet
- Upload a copy of the zip file to the MyLO dropbox

History tells us that mistakes frequently happen when following this process, so you should then:

- Unzip the folder to a new location
- Open the project and confirm that it still compiles and runs as expected
    - If not, repeat the process from the start

KIT205 Data Structures and Algorithms: Assignment 2

Synopsis of the task and its context

This is an individual assignment making up 15% of the overall unit assessment. The assessment criteria for this task are:

1. Implement code to construct an adjacency list representation of a graph
2. Implement Dijkstra's algorithm to find shortest paths
3. Implement Floyd-Warshall algorithm to find shortest paths

Match between learning outcomes and criteria for the task:

Unit learning outcomes	
On successful completion of this unit…	***Task criteria:***
On successful completion of this unit, you will be able to:	
1. Transform a real world problem into a simple abstract form that is suitable for computation	1-3
2. Implement common data structures and algorithms using a common programming language	1-3
3. Analyse the theoretical and practical run time and space complexity of computer code in order to select algorithms for specific tasks	-
4. Use common algorithm design strategies to develop new algorithms when there are no pre-existing solutions	-
5. Create diagrams to reason and communicate about data structures and algorithms	1-3

Criteria	HD (High Distinction)	DN (Distinction)	CR (Credit)	PP (Pass)	NN (Fail)
	You have:	**You have:**	**You have:**	**You have:**	**You have:**
1. Implement code to construct an adjacency list representation of a graph ***Weighting 30%***	• Correctly used the provided functions to generate an adjacency list from the DEM • Used correct data structures • Written code that is clear and easy to follow • Graph memory is freed when program terminates	• Correctly used the provided functions to generate an adjacency list from the DEM • Used correct data structures • Written code that is clear and easy to follow	• Correctly used the provided functions to generate an adjacency list from the DEM • Used correct data structures	• Used the provided functions to generate an adjacency list with some errors • Used correct data structures	• No serious attempt or incorrect data structures
2. Implement Dijkstra's algorithm to find shortest paths ***Weighting 35%***	• Correctly implemented function to find path distances using Dijkstra • Correctly implemented function to construct shortest paths • Used correct data structures • Written code that is clear and easy to follow • Distance and path memory is freed when no longer needed	• Correctly implemented function to find path distances • Correctly implemented function to return shortest paths • Written code that is clear and easy to follow • Used correct data structures	• Correctly implemented function to find path distances • Correctly implemented function to return shortest paths • Used correct data structures	• Implemented functions with some errors • Correct data structures used	• No serious attempt or incorrect data structures
3. Implement Floyd-Warshall algorithm to find shortest paths ***Weighting 35%***	• Correctly implemented function to find path distances using Floyd-Warshall • Correctly implemented function to return shortest paths • Used correct data structures • Written code that is clear and easy to follow • Distance and path memory is freed when no longer needed	• Correctly implemented function to find path distances • Correctly implemented function to return shortest paths • Written code that is clear and easy to follow • Used correct data structures	• Correctly implemented function to find path distances • Correctly implemented function to return shortest paths • Used correct data structures	• Implemented functions with some errors • Correct data structures used	• No serious attempt or incorrect data structures