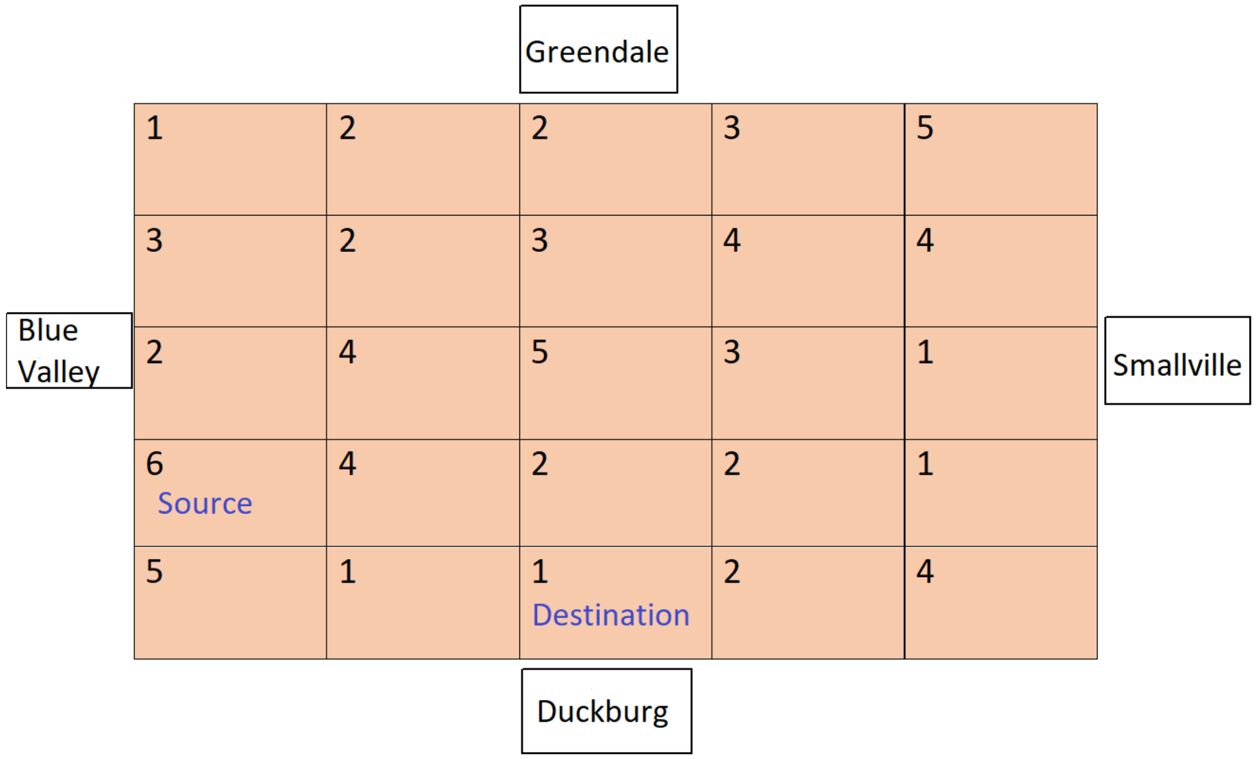
Graph Problem

Introduction

Consider a rectangular city (let’s say star_city) partitioned into a grid of square cells as shown in the image.



1	2	2	3	5
3	2	3	4	4
2	4	5	3	1
6 Source	4	2	2	1
5	1	1 Destination	2	4

Each of the borders of the star_city are surrounded by different cities- Greendale at top, Smallville at right, Duckburg at bottom and Blue Valley at left.

Each square cell represents the altitude of the land area (1 lowest and 10 highest).

A person can climb down from a higher altitude area to a lower altitude area. So, a person from a particular cell can reach to its adjacent cell (up, left, right, bottom) only if the adjacent cell is of equal or lower altitude.

Also, all the cells on the border are connected to the respective border city. That means a person can reach a particular neighboring city from any of the cells lying on the border for that city.

We have to find-

1. Any 1 route (if exists) from the cell marked Source to the cell marked Destination.
2. Any 1 route (if exists) from Blue Valley to Smallville

Housekeeping points

- This is a minimal example and may not follow some standard practices
- The focus is on the main flow, with minimal error handling. Errors in validation logic should be handled appropriately though.

Program Organization

This section will contain the information related to the different files available in the project.

1. ***class System:*** This class will contain the implementation related to the configuration and creation of adjacency matrix for the given graph.

    a. ***config_system:*** This method accesses the csv file available and plots the graph using matrix. This will be used to find the path between different nodes and cities.
    b. ***get_neighbours:*** This method will check if the given row and column is within the city limit and not going outside the plotted boundary of the matrix.

2. ***city_data.csv:*** This file contains the list of so that are available for you to push the data in the appropriate data structure. Please do not make any changes into this file. This file does not require any changes.
3. ***sample_output.txt:*** This is a sample output file. Once your code is in working state and all the methods are implemented properly, your output should appear like this.

Problem Statement

1. ***find_route:*** This method will be used to find the path between source and destination. More detail about the graph is mentioned below.
2. ***Bluevalley_to_Smallville_route:*** This method will find the route between two cities based on the condition provided in the problem statement.

We are given a matrix **star_city[][]** with **m rows and n columns** such that star_city[i][j] represents the altitude of the land area present at the i^{th} row and j^{th} column.

1. Cells at the top of the matrix are connected to Greendale. So star_city[0][0], star_city[0][1], star_city[0][2].. are connected to Greendale

2. Cells at the right of the matrix are connected to Smallville. So star_city[0][n-1], star_city[1][n-1], star_city[2][n-1].. are connected to Smallville
3. Cells at the bottom of the matrix are connected to Duckburg. So star_city[m-1][0], star_city[m-1][1], star_city[m-1][2].. are connected to Duckburg
4. Cells at the left of the matrix are connected to Blue Valley. So star_city[0][0], star_city[1][0], star_city[2][0].. are connected to Blue Valley

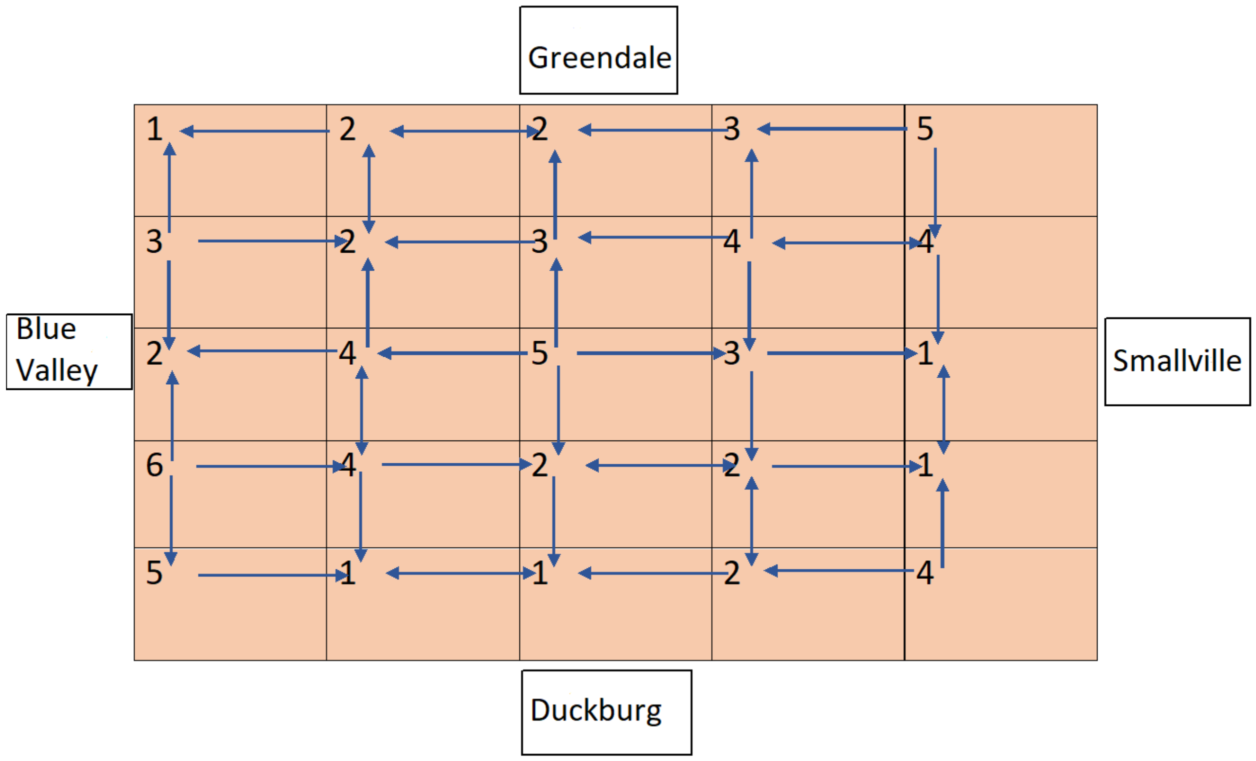
A person can travel from a particular cell star_city[i][j] to its adjacent cell (top, right, bottom, left) only if the **altitude of star_city[i][j] is greater than or equal to the altitude of the adjacent cell**.

Return-

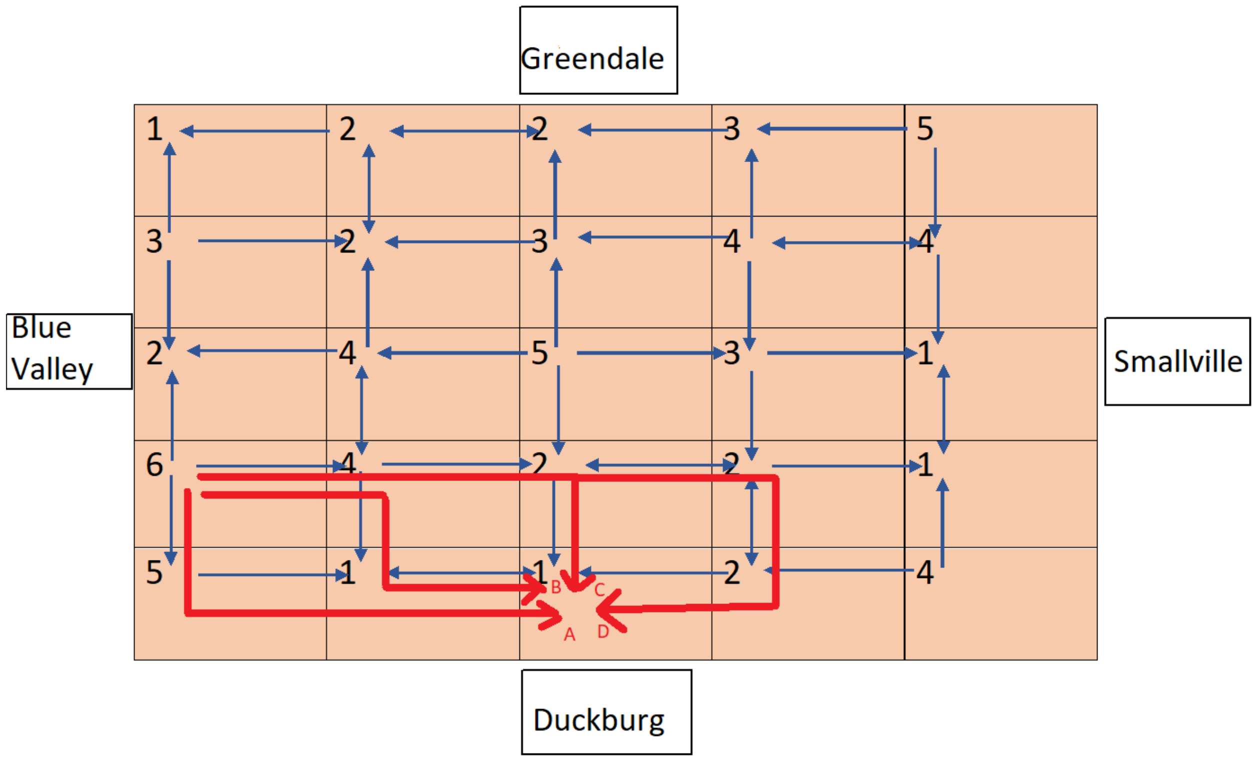
1. A 2D list containing all grid coordinates which can be traversed to reach the Destination cell from the Source cell.
2. A 2D list containing all grid coordinates which can be traversed to reach Smallville from Blue Valley.

Sample

As per the above image all the possible routes are marked as follows-



1. To reach from Source cell to Destination all possible routes A,B,C,D are marked in the following image-



As we need only 1 route, so the output can be either of the following lists-

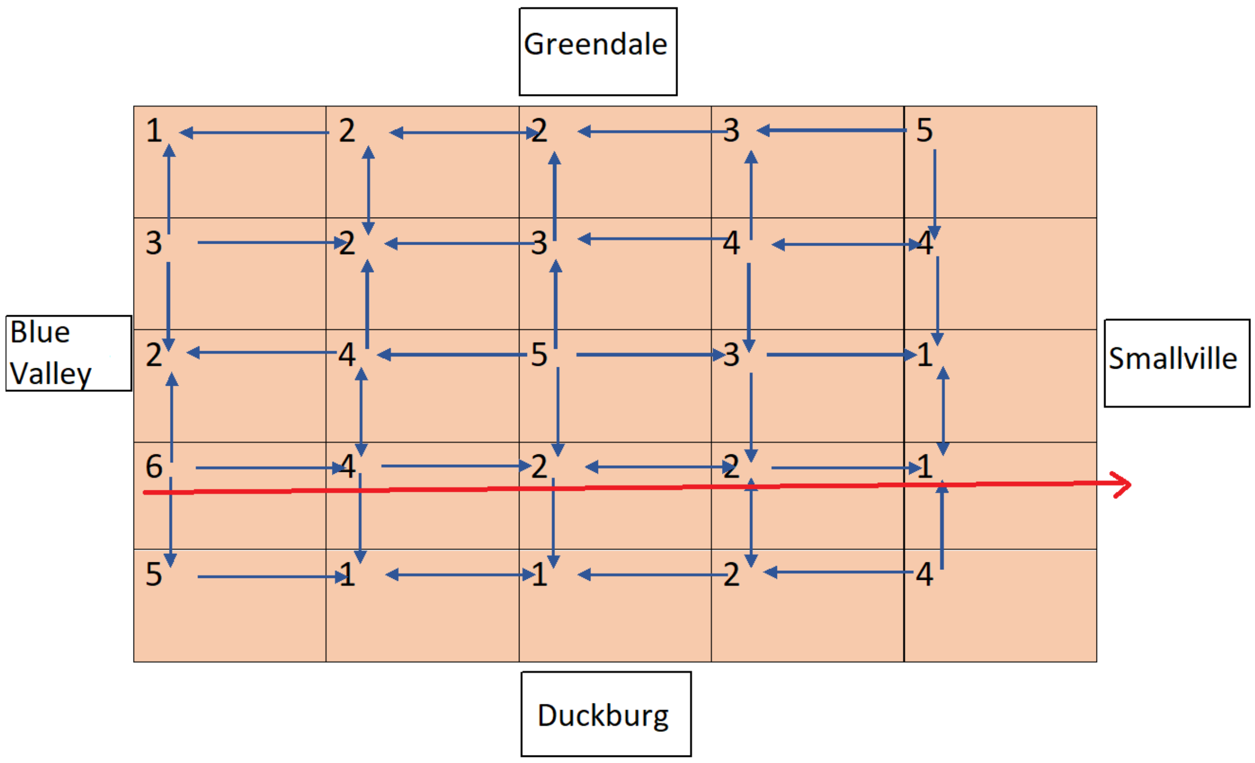
A- [[3][0], [4][0], [4][1], [4][2]]

B- [[3][0], [3][1], [4][1], [4][2]]

C- [[3][0], [3][1], [3][2], [4][2]]

D- [[3][0], [3][1], [3][2], [3][3], [4][3], [4][2]]

2. To reach Smallville from Blue Valley, there is only 1 route as shown in the image-



So, the output should be-

[[3][0], [3][1], [3][2], [3][3], [3][4]]

Evaluation Rubric

Total Project Points: **20**

- Basic compilation without errors (**10%**) : **2 Points**
- Correctness:
    - Problem statement - 1 (**60%**) : **12 Points**
    - Problem statement - 2 (**30%**) : **6 Points**

Program Instructions

1. Download the zipped folder named **C03-P02-Graph-Problem.zip**, and unzip it on your local machine. Go into the directory named **C03-P02-Graph-Problem**

2. Make sure that you have Python 3.6, or higher, installed. At your command prompt, run:
   ```
   $ python --version
   Python 3.7.3
   ```

   If not installed, install the latest available version of Python 3.

3. To run the code in the source code folder, run the following command:
   ```
   $   python3   C03-P02-Graph-Problem.py   (On   many   Linux/Mac
   platforms)
             OR
   $ python C03-P02-Graph-Problem.py           (On      Windows/Mac
   platforms)
   ```

   In any case, one of these two commands should work.

4. Alternatively, you could install a popular Python IDE, such as PyCharm or Visual Studio Code, and select a command to build the project from there.

5. You will be making very frequent changes into the C03-P02-Graph-Problem.py python files. The idea is to complete both the scripts so that it satisfies all the requirements. Once you have completed the changes in the code and it is executed without any error. Zip the folder as **C03-P02-Graph-Problem.zip** & now it is ready for submission.