Project 2: Dynamic programming

COT 4400, Summer 2023

Due July 21, 2023

1 Overview

For this project, you will develop an algorithm to find the most efficient paths across rough terrain. Designing and implementing this solution will require you to model the problem using dynamic programming, develop a method to extract the most efficient paths from your model, and implement your solution.

2 Problem Description

For this project, you will be computing the most efficient paths across a matrix that represents some region to be crossed. Each path should start at the northern (top) border of the region and make some combination of south, southeast, and southwest moves until it reaches the southern (bottom) border, and you should compute the most efficient path that ends at each location along the southern border of the region.

Each cell in the input matrix contains value representing the amount of time required to traverse that space. The total amount of time to traverse a path is the sum of time required to traverse each cell along the path; however, cells are closer to the cells immediately to their north and south than to those that are diagonal. In order to account for this, diagonal moves (southeast and southeast) should add a 40% penalty to the time required for the next cell. You do not pay this penalty on the top row of the matrix.) The "most efficient" path, of course, is the path that takes the least amount of time to cross according to this model. Note that you may use any cell on the top row as your starting point for any of your paths.

In order to compute these paths efficiently, you should compute, for every cell in the matrix, the minimum distance from that cell to the northern border of the area (when considering the diagonal movement penalty). You should develop an iterative dynamic programming algorithm to compute this value for every cell in the matrix, and then develop an algorithm to compute the most efficient traversal path for every cell on the southern border of the area based on these distances.

3 Input format

Your program should read its input from the file input.txt, in the following format. The first row of the file will contain two positive integers, *r* and *c*, specifying the number of rows and columns in the matrix, respectively. The following *r* lines of the file will contain *c* positive real numbers, indicating the traversal time for all of the locations in the matrix. The first row represents the top of the matrix, where the traversals should start, and the last line represents the bottom of the matrix, where the traversals should end. The first entry in each row represents the westernmost location and the last, the easternmost.

4 Output format

Your program should write its output to the file output.txt. You should write one line for each column in the matrix.

Each line should should start with a floating point number indicating the total length of the most efficient path ending at the bottom of a column. The second value on the line should be an integer indicating the starting column of the path, where columns are numbered 0 to $c-1$. The starting column should be followed by $r-1$ instances of the strings "S", "SE", and "SW", representing moves to the south, southeast, and southwest, respectively (r is the number of rows in the matrix). All of the entries on this line should be separated by spaces.

The first line of your output should be the most efficient path that ends at the bottom of the first column of the matrix, the second line for the second column, and so forth.

As an example, a path with a total length of 100 that started in the third column and moved south then southwest twice would be output as:

```
100 2 S SW SW
```

5 Coding your solutions

In addition to the report, you should implement an iterative dynamic programming algorithm that can solve the path finding problem described above. Your code may be in C or C++, but it must compile and run in a Linux environment.

If compiling your code cannot be accomplished by one of the commands

```
gcc -o pathfinder *.c g++ -o
pathfinder *.cpp
```

you should include a Makefile that capable of compiling the code via the make command. Your source code may be split into any number of files. Your code will not need to handle invalid input (e.g., matrix with incorrect size or negative values) nor problems with more than 1000 rows or columns.

6 Submission

Your submission for this project will be in two parts, the group submission and your individual peer evaluations.