Algorithms Programming Project

1 Problem Definition

Consider a city in Florida named Gridville that has a grid layout of $m \times n$ cells. Associated with each cell (i, j) where $i = 1, \ldots, m$ and $j = 1, \ldots, n$, Gridville architectural board assigns a non-negative number $p[i, j]$ indicating the largest possible number of floors allowed to build on that block. A developer company named AlgoTowers is interested to find the largest possible area (shaped square or rectangle) of blocks within city limits that allows a building of height at least h.

2 Algorithm Design Tasks

ALG1 Design a $\Theta(mn)$ time **Dynamic Programming** algorithm for computing a largest area **square** block with all cells have the height permit value at least h.

ALG2 Design a $\Theta(m^3n^3)$ time **Brute Force** algorithm for computing a largest area **rectangle** block with all cells have the height permit value at least h.

ALG3 Design a $\Theta(mn)$ time **Dynamic Programming** algorithm for computing a largest area **rectangle** block with all cells have the height permit value at least h.
[Hint: For gradual progress and also partial credit you might consider working on a $O(mn^2)$-time algorithm design first.]

3 Programming Tasks

Once you have the dynamic programming formulations for the algorithm design tasks, you should have an implementation for each of the following programming procedures:

TASK1 Give a recursive implementation of ALG1 using **memoization** and $O(mn)$ extra space.

TASK2 Give an iterative **BottomUp** implementation of ALG1 using $O(n)$ extra space.

TASK3 Give an implementation of ALG2 using $O(1)$ extra space.

TASK4 Give an iterative **BottomUp** implementation of ALG3 using $O(mn)$ extra space.

4 Language/Input/Output Specifications

You may use Java or C++. Your program must compile/run on the Thunder CISE server using gcc/g++ or standard JDK. You may access the server using SSH client on thunder.cise.ufl.edu. You must write a makefile document that creates an executable named **AlgoTowers**. The task is passed by an argument, e.g., when **AlgoTowers 3** is called from the terminal, your program needs to execute the implementation of TASK3.

Input. Your program will read input from standard input (stdin) in the following order:

- Line 1 consists three integers m, n, h separated by one space character.
- For the next m lines, line $i+1$ consist of n integers $p[i, 1], p[i, 2], ..., p[i, n]$ in this particular order separated by one space character.

For convenience assume that $1 \leq m, n \leq 2^{31}$, $0 \leq h \leq 2^{15}$, and $\forall i, j \;\; 0 \leq p[i,j] \leq 2^{15}$.

Output. Print four integers x_1, y_1, x_2, y_2 to standard output (stdout) separated by a space character, where (x_1, y_1) is the upper left corner and (x_2, y_2) is the lower right corner of the optimal solution region.

5 Experimental Comparative Study

You are expected to test your implementations extensively for correctness and performance. For this purpose, you should create randomly generated input files of various sizes. Then, you should do a performance comparison between Task 1 and Task 2 and between Task 3 and Task 4. For each comparison, generate a two dimensional plot of running time (y-axis) against input size (x-axis). These should be included in your report along with additional comments/observations. Feel free to do additional comparative study, e.g., you may compare do a performance comparison between Task 1 and Task 3 by adjusting the Task3 solution for square shaped regions.

6 Submission

The following contents are required for submission:

1. **Makefile**: Your makefile must be directly under the zip folder. No nested directories. Do not locate the executable file in any directory too.
2. **Source code**: should include detailed comments next to each non-trivial block of code lines.
3. **Report**: The report must be in PDF format. For each dynamic programming algorithm design task, you need to (i) state a mathematical recursive formulation expressing the optimal substructures. (ii) argue the optimal substructure (correctness). (iii) make a time and space complexity analysis. For each programming tasks, comment on ease of implementation, technicalities, performance. More importantly make sure to report on your experimental study including at least two comparative plots.
4. **Bundle**: Compress all your files together using a zip utility and submit through the Canvas system. Your submission should be named "**LastNameFirstName.zip**".

7 Grading Policy

Grades will be based on the correctness & efficiency of algorithms and the quality of the report:

- **Program 60%.** Correct/efficient design and implementation/execution. Also make sure to include comments with your code for clarity.
- **Report 40%.** Write up on your design, analysis, programming experience, and experimental study..

8 Bonus [upto %20]

For upto 20% bonus points, implement a solution for the following modified version of TASK4:

TASK5 Give an iterative **BottomUp** implementation of ALG3 using $O(n)$ extra space.

Note that TASK5 is more strict than TASK4 in terms of space constraint. Include a section in your report giving an experimental comparison on the impact of this extra space constraint on performance (e.g., speed and the size of the input program can handle).