1 Introduction

Arrays form the primary data storage in many applications such as image processing, combinatorial optimization and computer graphics. In this THE, you are given three tasks that relate to these areas.

2 Tasks

2.1 One-Dimensional Convolution (20 pts)

Convolution is a standard operation used in image processing to detect local patterns in an image. In this task, you will implement a convolution operation on a one-dimensional image represented as an integer array.

You will be given:

- An one-dimensional image I of size n, represented as a list of integers $I_1, \ldots, I_n$.
- A convolution *kernel* K of size $m \leq n$, represented as a list of integers $K_1, \ldots, K_m$.

You are expected to produce:

- A one-dimensional output image O of size $l = n - m + 1$, as a list $O_1, \ldots, O_l$.

O is computed as follows: Imagine I and K as one-dimensional horizontal grids. Place K on I such that K_1 and I_1 are aligned. Multiply each element of I with the element of K aligned with it. (Skip those which are not aligned with any of element of K). Sum the results up and store in O_1. To compute O_2, move K one element to the right and apply the same procedure. Repeat until K's last element falls out of I, to produce the whole O. (Such a procedure is sometimes referred to as a *sliding window* procedure, as K seems to slide over I.)

Input format will be:

$n\ m$
$I_1\ I_2\ \ldots\ I_n$
$K_1\ K_2\ \ldots\ K_m$

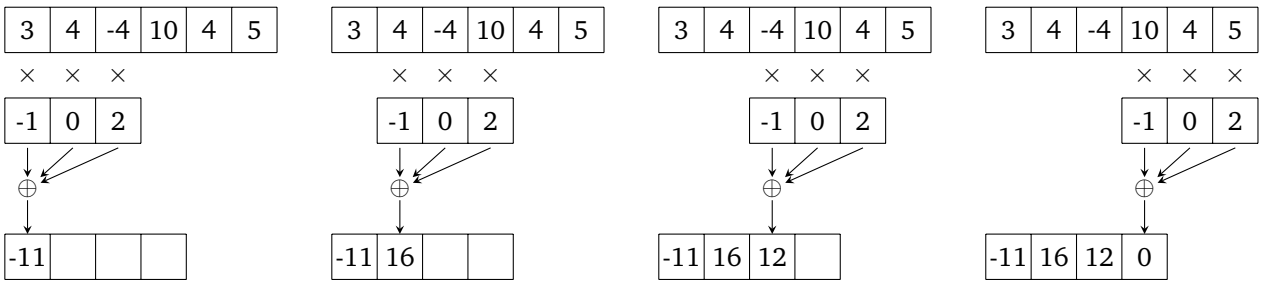
Output format will be:

$O_1\ O_2\ \ldots\ O_l$

It is guaranteeed that $1 \leq m \leq n \leq 100$ and all integers in I and K are between -1000 and 1000.

Example Input	Example Output
6 3 3 4 -4 10 4 5 -1 0 2	-11 16 12 0
5 4 10 50 30 20 40 -2 -1 1 2	0 -30

Visual Explanation for the First Input



2.2 Mean Optimization under Constraints (30 pts)

Recall that the *arithmetic mean* of a list $(a_1, \ldots, a_k)$ of numbers is $\frac{\sum_{i=1}^{k} a_i}{k}$.

You are given *square matrix* and you are supposed to find a *square shaped window* (submatrix) in it such that the arithmetic mean of the elements inside the window is maximized. Of course, the largest element in the matrix would already form a 1×1 window that maximizes the mean. Therefore, the problem also includes a *constraint*: The size of the window (i.e., its number of rows/columns) should be more than or equal to a given *minimum size*.

If there are multiple square windows that satisfy the constraint and maximize the mean, you are supposed to select one based on the following *tie-breaker* rules:

- Choose the largest one (in size).
- If there are multiple largest ones, choose the leftmost one.
- If there are multiple leftmost ones, choose the highest one.

The concepts left and high refer to the *position of the window in the input*.

Input format will be:

$$
\begin{array}{ccc}
n & m & \\
x_{1,1} & \cdots & x_{1,n} \\
\vdots & \ddots & \vdots \\
x_{n,1} & \cdots & x_{n,n}
\end{array}
$$

where n is the size of the matrix, m is the minimum size of the window to search, and $x_{r,c}$ is the matrix's entry in its rth row and cth column. All entries are space-separated. It is guaranteed that $1 \leq m \leq n \leq 30$ and all numbers in the matrix are between 0 and 1000.

Output format will be:

$$
\begin{array}{ccc}
r & c & s
\end{array}
$$

where r is the row number and c is the column number of the top-left corner of the optimizing window, and s is its size.

Example Input	Example Output
7 3 5 1 4 5 4 3 5 4 6 3 7 8 2 9 6 2 1 5 7 4 1 2 5 9 1 4 2 8 5 7 5 9 6 4 2 4 6 2 3 4 5 4 4 5 4 4 3 6 2	2 2 4

Example Input	Example Output
6 2 87 62 15 75 35 71 27 24 28 22 35 20 13 79 28 82 25 85 11 11 24 39 37 11 27 88 14 85 11 75 31 26 10 28 19 95	3 2 3

In the first example, the shown window uniquely maximizes the mean while staying within the size constraint. It has a mean of 5.3125. In the second example, there are 5 different windows that maximize the mean within the constraints: `1 1 2 / 1 4 3 / 3 2 3 / 3 4 3 / 5 5 2`. Based on the tie-breaker rules `3 2 3` is selected, with a mean of 50.0.

Side note: While it is probably not necessary for this problem, a technique to avoid floating-point numbers is applicable here. The mean here is the division of two integers and a decimal in general. One can, however, represent such a decimal indirectly as $\frac{a}{b}$ where a and b are integers. When a comparison is needed between two such decimals, one can use "cross-multiplication". For instance, $\frac{a}{b} < \frac{c}{d}$ if $ad < bc$.

2.3 Painter's Algorithm (50 pts)

The *painter's algorithm* is a method occasionally used in computer graphics to render images that consist of objects in various depths. The main idea of painter's algorithm is simple: Draw the objects in furthest to nearest order, so that nearer objects are not later hidden by further objects. In this task, you will implement the painter's algorithm on a simple scenario.

The objects you need to draw will be circles at various positions and of various radii. You are supposed to draw these circles onto a *canvas*, which you will represent as a character grid with n columns and m rows.

The bottom-left corner of the grid represents the coordinate $(1, 1)$. The top-right is (n, m). To more easily describe the math, please imagine that each grid cell (*pixel*) represents the geometric point at its center.

You will be given k circles. The ith circle:

- Is identified by an uppercase English letter l_i, which will be used to draw the circle on the canvas,
- Has its center at (x_i, y_i),
- Has radius r_i,
- Is at depth d_i.

A pixel (x, y) is *contained* by the ith circle, if its Euclidean distance to the circles center is less than or equal to r_i. In other words, it satisfies the inequality:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2} \leq r_i$$

This inequality involves a square root and therefore requires floating-point computation (which may be unreliable). (It also requires you to include the "math.h" header, which is forbidden.) However, notice that the same inequality can be written as:

$$(x - x_i)^2 + (y - y_i)^2 \leq r_i^2$$

which avoids the square root, keeps the computation in the integer domain, and therefore is more reliable.

When drawing the image, deeper circles are supposed to be *hidden by shallower ones*. At any pixel, one should see the letter that corresponds to the circle with the lowest depth among the ones that contain the pixel. If no circle contains a pixel, one should see '-' (dash) in that pixel. To draw the correct image, we suggest that you use the painter's algorithm, i.e., process circles in the order of decreasing depth.

When the canvas is simply drawn with a single character per pixel, due to the aspect ratio of the fonts, the circles look more like ellipses. Therefore, we ask you to draw each pixel with two characters instead (the same character repeated), so that the circles will look more like circles.

Additional Statistics

In addition to the image, some inputs may ask for additional statistics. When asked for additional statistics, your program is supposed to report, for each circle, the number of pixels that:

- Belong to the circle on the canvas. (We will refer to this as C_i for the ith circle.)
- Are contained by the circle on the canvas but is hidden by another circle. (We will refer to this as H_i for the ith circle.)
- Would be contained by the circle on an infinite-size canvas but falls outside the given canvas. (We will refer to this as F_i for the ith circle.)

Whether your program should output additional statistics is given as an integer S in the input (1 for yes, 0 for no statistics). *Half of the inputs for grading will not ask for statistics.*

Input & Output

Input format will be:

$$\begin{array}{l} n\ \ m\ \ k\ \ S \\ l_1\ \ x_1\ \ y_1\ \ r_1\ \ d_1 \\ l_2\ \ x_2\ \ y_2\ \ r_2\ \ d_2 \\ \vdots \\ l_k\ \ x_k\ \ y_k\ \ r_k\ \ d_k \end{array}$$

The input guarantees that

- $1 \leq n, m \leq 50$,

- $1 \leq k \leq 26$,
- The centers of all circles fall onto the grid,
- The radii of all circles will be between 1 and 50 (inclusively).
- The depths of the circles will be positive integers that fit into an `int`.

Output format will be:

$$
\begin{array}{cccc}
p_{1,m} & p_{2,m} & \cdots & p_{n,m} \\
\vdots & \vdots & \ddots & \vdots \\
p_{1,1} & p_{2,1} & \cdots & p_{n,1} \\
C_1 & H_1 & F_1 & \\
C_2 & H_2 & F_2 & \\
\vdots & & & \\
C_k & H_k & F_k &
\end{array}
$$

where $p_{x,y}$ is the character pair representing pixel (x, y). The pixels should not be space separated. All other fields should be space separated. The statistics C_i, H_i, F_i should *only be printed if $S = 1$*.

Example Input

```
30 20 4 1
C 10 9 8 700
E 12 10 4 200
N 16 14 7 400
G 25 20 8 900
```

Corresponding Output

```
------------------------NNNNNNNNNNNNNNGGGGGGGGGGGGGGGGGGGGGG
----------------------NNNNNNNNNNNNNNNNNNGGGGGGGGGGGGGGGGGGGG
--------------------NNNNNNNNNNNNNNNNNNNNNNGGGGGGGGGGGGGGGGGG
------------------NNNNNNNNNNNNNNNNNNNNNNNNNGGGGGGGGGGGGGGGGG
------------CCCCCCNNNNNNNNNNNNNNNNNNNNNNNNNNGGGGGGGGGGGGGGGG
--------CCCCCCCCCCNNNNNNNNNNNNNNNNNNNNNNNNNNGGGGGGGGGGGGGGGG
------CCCCCCCCCCNNNNNNEENNNNNNNNNNNNNNNNNNNNGGGGGGGGGGGGGGG
------CCCCCCCCCCCCEEEEEEEEEENNNNNNNNNNNNNNNNGGGGGGGGGGGG----
----CCCCCCCCCCCCEEEEEEEEEEEEEENNNNNNNNNNNNNN----GG----------
----CCCCCCCCCCCCEEEEEEEEEEEEEENNNNNNNNNNNNNN----------------
----CCCCCCCCCCEEEEEEEEEEEEEEEENNNNNNNNNN----------------
--CCCCCCCCCCCCCCEEEEEEEEEEEEEENNNNNNNNNN----------------
----CCCCCCCCCCCCEEEEEEEEEEEEEENNNNNNNN----------------
----CCCCCCCCCCCCCCEEEEEEEEECCNNCC----------------
----CCCCCCCCCCCCCCCCCCEECCCCCCCCCC----------------
------CCCCCCCCCCCCCCCCCCCCCCCCC----------------
------CCCCCCCCCCCCCCCCCCCCCCCCC----------------
--------CCCCCCCCCCCCCCCCCCCCC----------------
------------CCCCCCCCCCCCC----------------
----------------CC----------------
117 80 0
49 0 0
117 31 1
68 28 101
```

3 Submission & Rules

- Submit a separate C source file (with given name in ODTÜClass/VPL system) for each of the tasks.

- We will use VPL system for submission (which will be case for lab exams as well.) You can use the system directly as an editor or work locally and upload your source files to the system later.
- **Run** and **debug** modes in VPL system are disabled since we have 4 main functions exist in submission files. You can directly use **evaluate** mode in VPL to run all tasks at once and see your grade. You can evaluate your submission infinitely many times.
- Late submission ***IS NOT*** allowed. **Please, do not ask for any deadline extensions**.
- In your solutions, you should ***NOT*** use any library functions other than those in *"stdio.h"*.
- We will compile your code with the following command:

```
gcc yourCode.c -o yourExecutable -g -std=c17 -Wall -Wextra -Wpedantic
```

- If you are working on a command prompt, you can likely feed input files to your program instead of typing inputs. This way, you can test your inputs faster. For instance, you can type:

```
> ./yourExecutable < yourInput.txt
```

- Your grade will be given by an auto-grader upon submission. The grade will be determined by running your code against several input files including, but not limited to the test cases given to you as an example. It is possible to get a partial grade from each task.
- You should stick to the output format described in the tasks. Your program's output should not contain any unnecessary characters or whitespace. (The only exception to this is that you may optionally print a newline character at the end of your output.) It is your responsibility to check the correctness of the output in terms of whether it contains any invalid invisible characters. This also means you are not supposed to print any prompts for input.
- Your programs should terminate with an exit code of 0.
- Your programs should be reasonably well-written such that they do not run slowly. The time limit we impose in the auto-grade is more than enough with respect to the tasks that you are assigned; however, if you have a bug or a code that is extremely slow, your program may exceed the time limit and you will not receive any points.
- Grade from the auto-grader is ***NOT*** final. We may re-evaluate your submissions in black-box or white-box manner and adjust your grade. Submissions that do not reasonably attempt to solve the tasks may lose points.
- As usual, you are expected to complete your solutions in ***academic integrity***. We have zero-tolerance policy for cheating. People involved in cheating will receive a grade of zero and may be prosecuted according to the university regulations. *Sharing code (in any amount) with each other or using third party code is strictly forbidden.* Be aware that there are "very advanced tools" that can detect this type of cheating.

Good Luck!