1. Adjacent numbers

Description
Given a n*m matrix, please judging whether there exists two or more adjacent number with the same value in a same row or a same column.

Input
The first line will be an integer T (1<=T<=20), which is the number of test cases. For each test case, the first line will be two integer n m (0<=n<=100, 0<=m<=100) means the number of rows and columns in this two-dimensional array. The next n rows have m integers per row, which is the element in each row, and the size of each integer element is in a range [-2^30, 2^30].

Output
For each test case, if exists adjacent numbers, your program is expected to output "true", otherwise, "false".

Sample Input
```
3
3 3
1 2 2
3 4 5
5 6 7
10 10
6 -6 -10 1 0 2 -9 -10 8 -4
8 9 -1 3 -1 -9 5 -9 -8 -2
7 -2 -9 -5 -3 3 -9 -7 -9 -5
-10 1 -9 -6 3 -10 -9 9 -8 -7
4 -1 -7 -2 -10 -7 9 -5 6 2
0 -3 -2 -5 0 2 -4 -9 -5 0
6 5 -3 -9 -3 3 -7 -6 -9 2
6 -5 8 2 9 -10 -10 -1 6 4
-4 0 -6 7 7 6 -7 -7 -6 -2
1 6 -8 4 -7 0 5 4 -1 -3
2 2
1 3
2 4
```

Sample Output
```
true
true
false
```

2. Processing two-dimensional arrays

Description:
There is a two-dimensional array that stores integers, and the number of elements in each row is not equal. We can do the following for the array:
k r: means to add k to all elements of the rth row (k is not equal to 0)
0 r: means to remove all 0 in the rth row

Input:
The first line will be an integer T (1<=T<=10), which is the number of test cases. For each test case, the first line will be two integer n m (1<=n, m<=100) means the number of rows in this two-dimensional array and the number of operations. The next n rows are the two-dimensional array, and the first element of each row represents the number of elements in this row. The last m rows have two integers per row, which means the operation to be performed.

Output:
For each test case, output the processed two-dimensional array, separated by a space.

Sample Input:
```
2
3 3
3 1 2 3
3 2 -1 4
5 1 5 5 5 5
1 0
1 1
0 1
3 3
2 2 3
3 -1 -2 -3
3 -3 4 5
-2 0
3 2
0 0
```
Sample Output:
```
2 3 4
3 5
1 5 5 5 5
1
-1 -2 -3
0 7 8
```

3. Center two-dimension array

Description:
Given a two-dimension array with different 2nd dimension lengths, center the two-dimension array based on the maximal 2nd dimension length and print it.

Input:
The first line will be an integer T (1≤T≤20), which is the number of test cases.
For each test case:

- The first line will be an integer n (1≤ n ≤20), which means the how many rows in the tow-dimensional array.
- The second line will be the n integers, for each integer, means the length of each row, and these n integers are all odd number or even number.
- The following n lines, for each line will be the data (from 0 to 9) in each row.

Output:

Print the centered two-dimension arrays from the first test case to the last the test case, and we use only '\n' to separate each test cases.
For each test case:

- There is only '\n' to separate each rows.
- We need to make sure that the intermediary number in all rows should be printed in a vertical line. (You can refer to the sample output)

For each rows, there is only a single space ' ' to separate each element.

Sample Input:

```
2
4
10 10 10 14
3 3 3 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0
8
1 3 3 3 3 3 5 1
0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0 0 0
0
```

Sample Output:

```
   3 3 3 3 3 3 3 3 3 3
   2 2 2 2 2 2 2 2 2 2
   1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0
  0 0 0
  0 0 0
  0 0 0
  0 0 0
  0 0 0
0 0 0 0 0
   0
```

4. Pascal's triangle

Description:

In mathematics, Pascal's triangle is a triangular array of the binomial coefficients. The entry in the nth row and kth column of Pascal's triangle is denoted as (n, k). For example, the unique nonzero entry in the topmost row is (0, 0) = 1. With this notation, the construction of the pascal's triangle paragraph may be written as: (n, k) = (n -1, k -1) + (n – 1, k) for any non-negative integer n and any integer k between 0 and n, inclusive.

Pascal's triangle determines the coefficients which arise in binomial expansions. For an example, consider the expansion:
(x + y)^2 = 1•x^2•y^0 + 2•x^1•y^1 + 1•x^0•y^1
The nth row of Pascal's triangle is the binomial coefficients of (x + y)^n.
(x + y)^n = a_0•x^n + a_1•x^(n-1) •y + ... + a_n-1•x•y^(n-1) + a_n•y^n
The entry is a_i = (n, i)

You need to print the nth row of Pascal's triangle.

Input:

The first line will be an integer T (1<=T<=150), which is the number of test cases. For each test case, there will be one line of an integer n (1<=n<=33) means the nth row needed to print.

Output:

For each test case, output entries of nth row of Pascal's triangle, separated by a space.

Sample Input:

```
4
4
3
2
1
```

Sample Output:

```
1 3 3 1
1 2 1
1 1
1
```

5. Sudoku

Description

Sudoku is a famous mathematical game in which players fill numbers 1-9 in a 9 by 9 square. The square satisfies that every row and every column contain 1-9 only once, for example, a column contains [2 3 6 4 9 8 1 5 7] is valid but a column contains [1 1 3 6 8 4 2 9 7] is invalid. Specially, the square is divided into 9 "small squares", that every small square has size 3 by 3. Of course, every "small square" also contains 1-9 only once. The picture is a valid sudoku (Notation that small squares are divided by red line).

2	9	3	7	1	5	4	8	6
8	6	1	2	4	9	5	3	7
7	4	5	8	6	3	1	9	2
6	7	8	9	2	1	3	4	5
1	3	9	5	7	4	2	6	8
4	5	2	6	3	8	7	1	9
9	2	4	3	8	7	6	5	1
3	8	6	1	5	2	9	7	4
5	1	7	4	9	6	8	2	3

After the player fills all the blanks in the sudoku, he wants to know whether he has finished it correct. So, the player turns to you to design a program to verify the correction of sudoku.

Input

The question will contain several test cases, the integer T (1<=T<=20) in the first line indicates the number of test cases.
For each test cases:
There will be 9 lines and every line has 9 integers, all the integers are from 1 to 9, split by a space character.

Output

Each test cases have only one line for output. If the sudoku is valid, output "YES", if not output "NO".

Sample input

```
2
9 7 8 3 1 2 6 4 5
3 1 2 6 4 5 9 7 8
6 4 5 9 7 8 3 1 2
7 8 9 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
8 9 7 2 3 1 5 6 4
```

```
2 3 1 5 6 4 8 9 7
5 6 4 8 9 7 2 3 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
```

Sample output

```
YES
NO
```

6. Spiral Matrix [bonus]

Description:
Given a matrix of m x n elements (m rows, n columns), output an array that contains all elements of the matrix in clockwise spiral order, starting from the given position.

Input:
The first line will be an integer T (1<=T<=10), which is the number of test cases.
For each test case, the first line will be two integers m and n (0 < m, n <= 20).
Then there will be m lines, each line contains n integers, representing the matrix.
After the matrix, the next line will be two integers x and y (0 <= x < m, 0 <= y < n), which is the initial row and column position that the spiral path begins from.

Output:
For each test case, output the array that contains all elements of the spiral path and there is only a single space ‘ ’ to separate each element.

Sample Input:

```
3
3 4
1 2 3 4
4 5 6 7
7 8 9 0
1 2
3 3
1 2 3
4 5 6
7 8 9
1 1
3 3
1 2 3
4 5 6
7 8 9
0 0
```

Sample Output:

```
6 7 0 9 8 5 2 3 4 7 4 1
5 6 9 8 7 4 1 2 3
1 2 5 4 3 6 9 8 7
```

7. Spatial Convolution [bonus]

Description

Convolution is a widely-used signal processing operation, and nowadays convolutional neural network is becoming a tremendously popular research interest. In this problem, you are asked to convolve some images with given kernels.

A greyscale digital image x consists of many pixels organized in rows and columns. The brightness of a pixel x[i][j] is discretized and usually represented by an integer ranging from 0 to 255. A two-dimensional kernel, or filter, is a small matrix indicating how to combine a pixel with its neighbors in the image to get a processed value. (Mathematically, a kernel is defined on a infinitely extended plane. When a kernel is given as a matrix with finite entries, you should consider any undefined value as 0.)

The discrete convolution of an image x and a kernel h can be calculated using the following formula:

$$(x \star h)[i,j] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} x[u,v]\, h[i-u, j-v]$$

We may think about the formula in this way: To calculate the convolution sum at $[i,j]$, we first flip the kernel h in both dimensions, i.e. rotate it 180° . Then, place the flipped kernel h' on the original image such that the center of h' covers $x[i,j]$, do pointwise multiplication on the overlapped area, and sum up the products. The sum would be the convolution sum at $[i,j]$.

To blur images, people usually use Gaussian kernels. Figure 1 may help you understand how a 2D Gaussian kernel looks like. Figure 2 depicts how ***(x*h)[2, 3]*** is calculated. In figure 2, the overlapped area is surrounded by the red rectangle. To obtain a blurred image, we need to calculate every ***(x*h)[i, j]*** in an analogous manner, and thus the rectangle should be shifted many times to cover every ***x[i, j].*** In this problem, when $\boldsymbol{h}'$ is placed on the borders or at the corners of x, covering an area where some values of **x** is undefined, you should substitute them with their nearest defined values respectively.

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

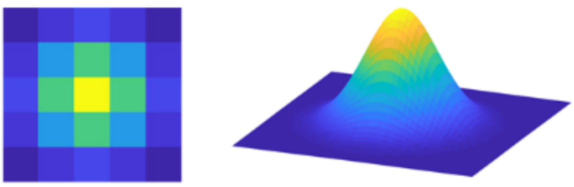
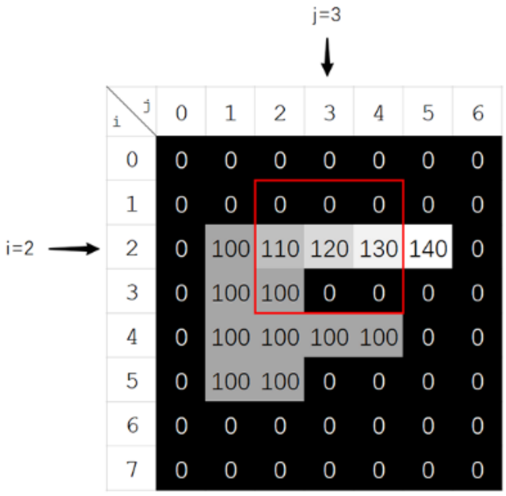


Fig.1 Two-dimensional gaussian kernels



$$h \longrightarrow h' = h$$

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$\frac{1}{16}\left(\begin{array}{c} 0\times 1+0\times 2+0\times 1\\ +110\times 2+120\times 4+130\times 2\\ +100\times 1+0\times 2+0\times 1\end{array}\right)=66.25$$

$$(x\star h)[2,3]=66.25\approx 66$$

Fig.2 Calculating $(x \star h)[2,3]$

Different convolution kernels lead to different processing effects. You are encouraged to visit https://en.wikipedia.org/wiki/Kernel_(image_processing) to see more convolution kernels. If you want to learn more about filtering operators for image processing, you can refer to Section 3.2 of *Computer Vision: Algorithms and Applications* by *R. Szeliski.* (Available on Springer: https://link.springer.com/content/pdf/10.1007%2F978-1-84882-935-0_3.pdf)

Input

The first line of the input will be an integer T $(1 \le T \le 10)$, indicating the number of test cases.
For each test case:
The first line is an odd integer M $(1 \le M \le 9)$, followed by an $M{\times}M$ matrix in the next M lines, which is the convolution kernel. Every entry of the convolution kernel has at most four decimal places. (A convolution kernel can contain negative values.)
Then come two integers H and W $(1 \le H, W \le 100)$, the height and width of the image. Each of the next H lines consists of W integers ranging from 0 to 255, representing the pixel values of the grayscale image to process.

Output

For every test case, output the calculated spatial convolution in H lines, each of which contains W integers. You should round your results to the nearest integers. Pick a closest boundary value for a result that is smaller than 0 or greater than 255. You are supposed to right-align your output as the output example does. (Hint: use "%3d" as format string and add one more space between numbers.)

Sample Input

```
3
3
0.0625 0.1250 0.0625
0.1250 0.2500 0.1250
0.0625 0.1250 0.0625
8 7
0   0   0   0   0   0   0
0   0   0   0   0   0   0
```

```
0 100 110 120 130 140   0
0 100 100   0   0   0   0
0 100 100 100 100   0   0
0 100 100   0   0   0   0
0   0   0   0   0   0   0
0   0   0   0   0   0   0
5
0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04
1 1
100
3
-1 0 1
-2 0 2
-1 0 1
3 3
9 8 7
0 1 0
7 9 8
```

Sample Output

```
  0   0   0   0   0   0   0
  6  19  28  30  33  26   9
 19  58  74  66  65  51  18
 25  76  90  68  51  32   9
 25  75  88  63  38  13   0
 19  56  63  38  19   6   0
  6  19  19   6   0   0   0
  0   0   0   0   0   0   0
100
  2   6   4
  0   1   4
  0   0   4
```