Problem 1. Fountain Codes

(Time Limit: 1 seconds)

Problem Description

You want to transform a message in an *erasure* network. The term "erasure" indicates that routers in the network may drop your packet due to congestion or packet collision. To ensure that the receiver can reassemble your original message, you adopt the *fountain code* technique to improve the reliability of the transmission system.

The fountain code approach encodes your message into several encoded blocks that can be sent over the network. The receiver can reconstruct the original message as long as she/he receives sufficient number of encoded blocks. The beauty of fountain codes is that it does not matter which encoded blocks are received or in what order they are received.

Fountain codes are widely used in many network applications such as `BitTorrent` or on-demand TV.

We explain the encoding scheme in the following.

1. The original message is split into several *source blocks*. We use n and s_i to respectively indicate the number of source blocks and the i_{th} source block.
2. Pick a random number, k, between 1 and n.
3. Randomly pick k source blocks and use XOR operator to combine them together. The combined block is name an *encoded block*.
4. Transmit the encoded block along with the information about which source blocks it was constructed from.

Here is an example. Suppose the message you what to transmit is "xyz123". We break the message into 6 source blocks (i.e., each source block is 1 character's length). Thus, in the example, $n = 6$ and $s_0 = 'x'$, $s_1 = 'y'$, $s_2 = 'z'$, $s_3 = '1'$, $s_4 = '2'$ and $s_5 = '3'$.

Now that $k = 2$, we randomly choose 2 source blocks. Assume that the source blocks we picked are s_1 and s_2. We can get an encoded data by using the following procedure: $\text{data} = s_1\ XOR\ s_2 = 'y'\ XOR\ z' = 121\ XOR\ 122 = 3$. Note that 121 and 122 are the ASCII of $'y'$ and $'z'$, respectively.

Finally, we construct the encoded block $e =< 3, [1, 2] >$ and send e to the receiver. Note that in the encoded block e, 3 is the encoded data and $[1, 2]$ represents the source blocks (i.e., s_1 and s_2) from which e is constructed.

The decoding procedure is more complicated than the encoding procedure. We use two examples to explain how it works.

Example 1

Assume the receiver have got three encoded blocks e_0, e_1, and e_2. The values of the three encoded blocks are shown in Table 1. We also assume that the number of source blocks is 6 (i.e., $n = 6$). Table 2 shows the value of each source block. At the beginning, the value of every source block is initiated to ∅, indicating that the value is "unknown".

■ Table 1 The received encoded blocks.

Encoded block	Value
e_0	$< 120, [0] >$
e_1	$< 121, [1] >$
e_2	$< 123, [0,1,2] >$

■ Table 2 The value of each source block in the beginning.

Source block	s_0	s_1	s_2	s_3	s_4	s_5
value	∅	∅	∅	∅	∅	∅

The first encoded block (i.e., e_0) consists of only one source block (i.e., s_0), so we already know value of the first source block (i.e., 120). Similarly, we can decode the value of s_1 to 121.

Now we process e_2. We can see that the value of e_2 depends on s_0, s_1, and s_2. Since we already know the values of s_0 and s_1, we can xor them together and obtain the value of s_2. That is,

$$s_0 \; \boldsymbol{XOR} \; s_1 \; \boldsymbol{XOR} \; e_2 = 120 \; \boldsymbol{XOR} \; 121 \; \boldsymbol{XOR} \; 123 = 122 = 'z'$$

Therefore, we have the value of $s_2 = 122 = 'z'$. We show the value of each source block in Table 3 after the processing of e_0, e_1, and e_3.

■ Table 3 The value of each source block after e_0, e_1, and e_2 are processed.

Source block	s_0	s_1	s_2	s_3	s_4	s_5
value	$120 = 'x'$	$121 = 'y'$	$122 = 'z'$	∅	∅	∅

Example 2:

Continued from Example 1, we receive another three encoded blocks (see Table 4).

■ Table 4 The received encoded blocks.

Encoded block	Value
e_3	$< 3, [3,4] >$
e_4	$< 73, [0,3] >$
e_5	$< 1, [4,5] >$

The value of e_3 is consisted of s_3 and s_4. Since the values of s_3 and s_4 are unknown (i.e., $\varnothing$), we cannot decode e_3. We store e_3 for later use.

The value of e_4 depends on s_0 and s_3. We already know the value s_0, so we can decode the value of s_3:

$$s_0 \; \boldsymbol{XOR} \; e_4 = 120 \; \boldsymbol{XOR} \; 73 = 49 = '1'$$

We now know enough information to decode e_3 after the value of s_3 is gained. The value of s_4 can be obtained using the following procedure:

$$s_3 \; \boldsymbol{XOR} \; e_3 = 49 \; \boldsymbol{XOR} \; 3 = 50 = '2'$$

Repeating the same procedure again, we can decode e_5 and then obtain the value of s_5:

$$s_4 \; \boldsymbol{XOR} \; e_5 = 50 \; \boldsymbol{XOR} \; 1 = 51 = '3'$$

Table 5 shows the value of each source block after $e_0 \sim e_5$ are processed.

● Table 5 The value of each source blocks after $e_0 \sim e_5$ are processed.

Source block	s_0	s_1	s_2	s_3	s_4	s_5
value	$120 = 'x'$	$121 = 'y'$	$122 = 'z'$	$49 = '1'$	$50 = '2'$	$51 = '3'$

Technical Specification

- $1 \leq n$ (the number of source blocks) ≤ 1250
- $1 \leq |E|$ (the number of received encoded blocks) ≤ 500
- $1 \leq k \leq 5$
- $1 \leq$ the number of test cases ≤ 8.

Input Format

The first line is an integer which indicates the number of test cases. Each test case consists of two parts.

The first part of the test case is an integer *n* indicating the number of source blocks (i.e., the source blocks are indexed from 0 to *n*-1).

The second part of the test case contains several lines. Each line represents a received encoded block. Each encoded block contains several integers delimited by a space. The first integer of the encoded block is the encoded data. The following integers indicate the source blocks from which the encoded block is constructed.

The test case is terminated by a line containing the integer -1.

Output Format

For each test case, output the number of source blocks that are decoded. For example, in Example 1, you should output 3, since the number of decoded source blocks is 3. However, in Example 2, all six source blocks are decoded, so you should output 6.

Example

Sample Input:	**Sample Output:**
1 3 120 0 121 1 123 0 1 2 -1	3

Problem 2. Maximal Hits

(Time Limit: 2 seconds)

Problem Description

Given a square matrix with random values from 0 to 4, number 0 means you can get through it, and the numbers from 1 to 4 in the matrix represent directions heading for upward, rightward, downward, and leftward respectively. You are always asked to start from a non-zero element. The movement keeps going until it reaches a visited element or is going beyond the matrix. The number of hits is equal to the total number of non-zero elements you visited. You are asked to print the maximum number of hits among all feasible paths.

Input Format

The first line is an integer indicating the number of test cases. Each test case consists of two parts. The first part is a line which contains an integer *n* (1~100) indicating the dimension of the square matrix. The second part contains *n* lines and each line contains *n* integers to represent the elements of the square matrix.

Output Format

For each test case, output the maximum number of hits among all feasible paths in a line.

Example

Sample Input:	**Sample Output:**
1 5 3 0 0 0 0 2 3 0 0 0 0 2 3 0 0 0 0 2 3 0 0 0 0 2 3	9

Problem 3. Gold

(Time Limit: 3 seconds)

Problem Description

An ancient tale says that you can make gold from lead and mercury, but the ratio of these two elements is the key to success. Fortunately, your father knows the magic ratio and he tells you this secret. However, you are very cautious, and you want to prove this by yourself. Suppose there are several available bricks of alloys of lead and mercury for you. You want to take some of them and then mix them by the magic ratio in order to make gold. For simplicity, we use v_i to denote the i_{th} brick, whose ratio of lead and mercury is denoted as $a_i : b_i$. However, before mixing the alloys you are choosing, you need to spend w_i dollars to process brick v_i into a solution, and then get the needed portion of v_i from the solution. For example, let the magic ratio be $\mathrm{v} = 7:13, v_1 = 3:7$ with $w_1 = 1, v_2 = 5:5$ with $w_2 = 1$, and $v_3 = 10:9$ with $w_3 = 10$. If you chose bricks v_1 and v_2, we would have $(7,13) = 1.5 \cdot (3,7) + 0.5 \cdot (5,5)$ with cost $w_1 + w_2 = 1 + 1 = 2$ that you have to pay. Notice that there is no limit on the quantity of v_i that you are using, and the amount of gold that you turned is not an issue here, because you only want to prove this big idea. Finally, your goal is to minimize the cost you have to pay.

Technical Specification

- a_i, b_i and w_i are all positive integers less than 10000, for $1 \leq i \leq \mathrm{n}$.
- $1 \leq \mathrm{n} \leq 10^5$.

Input Format

The first line of the input contains an integer indicating the number of test cases. Each test case has several lines. The first line gives n, the kinds of bricks that you can use; the second line gives you the magic ratio a and b which are all positive integers. In the next n lines, each specifies three positive integers a_i, b_i and w_i, for i from 1 to *n*, accordingly.

Output Format

For each test case, please output the minimum cost in a line if it is possible to make gold from the specified bricks; otherwise output “Impossible”.

Example

Sample Input:	Sample Output:
2 1 1 2 2 1 1 2 1 2 2 1 1 1 3 1	Impossible 2

Problem 4. Matrix Transformation

(Time Limit: 1 second)

Problem Description

A matrix is a rectangular array of elements. If a matrix has *R* rows and *C* columns, we say that it's size is $R \times C$. For any matrix *X*, we use X_{ij} to denote the element at the i_{th} row and the j_{th} column. For example, in Figure 1, the size of *X* is 4×6 and $X_{32} = 11$.

Define two operations on a matrix of *R* rows as follows:

- **Reverse:**

Exchange the 1st row and the R_{th} row, exchange the 2nd row and the $(R-1)_{th}$ row, exchange the 3rd row and the $(R-2)_{th}$ row, and so on.

- **Rotate:**

Rotate the matrix 90 degrees in clockwise.

For example, in Figure 1, matrix *Y* is obtained if a Reverse operation is performed on *X*; and matrix *Z* is obtained if a Rotate operation is performed on *X*.

X

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 2 & 1 \\ 0 & 0 & 1 & 0 & 1 & -1 \\ 2 & 11 & 4 & 8 & 7 & 5 \\ -3 & -1 & 0 & 11 & 2 & -8 \end{bmatrix}$$

Y

$$\begin{bmatrix} -3 & -1 & 0 & 11 & 2 & -8 \\ 2 & 11 & 4 & 8 & 7 & 5 \\ 0 & 0 & 1 & 0 & 1 & -1 \\ 1 & 2 & 3 & 3 & 2 & 1 \end{bmatrix}$$

Z

$$\begin{bmatrix} -3 & 2 & 0 & 1 \\ -1 & 11 & 0 & 2 \\ 0 & 4 & 1 & 3 \\ 11 & 8 & 0 & 3 \\ 2 & 7 & 1 & 2 \\ -8 & 5 & -1 & 1 \end{bmatrix}$$

Figure 1.

A matrix can be transformed into another matrix by applying a sequence of Reverse and Rotate operations. Given two matrixes *A* and *B*, please compute the minimum number of operations that transform *A* to *B*.

Technical Specification

- The number of test cases is at most 10.
- The numbers of rows and columns are at most 1000.
- Each element of a matrix is an integer between −100 and 100.

Input Format

The input describes several test cases. The first line of input for each test case contains four integers R_A, C_A, R_B, C_B, indicating that A is of size $R_A \times C_A$ and B is of size $R_B \times C_B$, where $1 \le R_A, C_A, R_B, C_B \le 1000$. The following R_A lines describe the matrix A and each line contains C_A integers, where the j_{th} integer in the i_{th} line represents A_{ij}, where $-100 \le A_{ij} \le 100$. The next R_B lines describe the matrix B and each line contains C_B integers, where the j_{th} integer in the i_{th} line represents B_{ij}, where$-100 \le B_{ij} \le 100$.

The input is terminated by a line containing four zeros, which should not be processed.

Output Format

For each test case, please output the minimum number of operations that can transform A to B. If A cannot be transformed into B, output -1.

Example

Sample Input:	**Sample Output:**
4 6 6 4 1 2 3 3 2 1 0 0 1 0 1 -1 2 11 4 8 7 5 -3 -1 0 11 2 -8 1 -1 5 -8 2 1 7 2 3 0 8 11 3 1 4 0 2 0 11 -1	3 -1

1 0 2 -3 3 2 2 3 0 1 1 2 2 0 2 1 0 1 2 0 0 0 0 0	

Problem 5. Palindrome

(Time Limit: 3 seconds)

Problem Description

Given a string *S*, you can add some characters in the head of *S*. What is the minimum number of characters you need to add in order to turn *S* into a palindrome? A palindrome is a sequence of characters which reads the same backward or forward.

Technical Specification

- The length of *S* is bounded in range $[1,1000000]$.
- Each character of *S* comes from the small case English character set $a-z$.

Input Format

The first line of the input gives the number of test cases, *T*. *T* test cases follow. Each test cases will have a string *S* alone in a line.

Output Format

For each test case, output the number of characters added to the head of *S* in a single line.

Example

Sample Input:	**Sample Output:**
4 aacecaaa abcd aba abbacd	1 3 0 2