Problem A
A Simplified Block Cipher

(Time Limit: 1 second)

In a block cipher, the message to be encrypted is processed in blocks of *k* bits. For example, if *k*=3, a 12-bits message is broken into four 3-bit blocks and each block is encrypted independently. A k-bit block of cleartext is mapped to a k-bit of ciphertext to encrypt a block. Table 1 shows just one possible mapping for the 3-bit block example. With this table, the message 010110001111 gets encrypted into 101000111001.

Table 1. A specific 3-bit block cipher

Input	000	001	010	011	100	101	110	111
Output	110	111	101	100	011	010	000	001

When *k*=3, the eight inputs can be permuted in 8! different ways. With only 8! mappings, brute-force attacks can quickly decrypt ciphertext by using all mappings. To thwart brute-force attacks, *k* is set as 64 in this problem. However, it is an infeasible task for a sender and its corresponding receiver to maintain a full table with 2^{64} values. To this end, a procedure as shown in Figure 1 is proposed to simulate the full table.

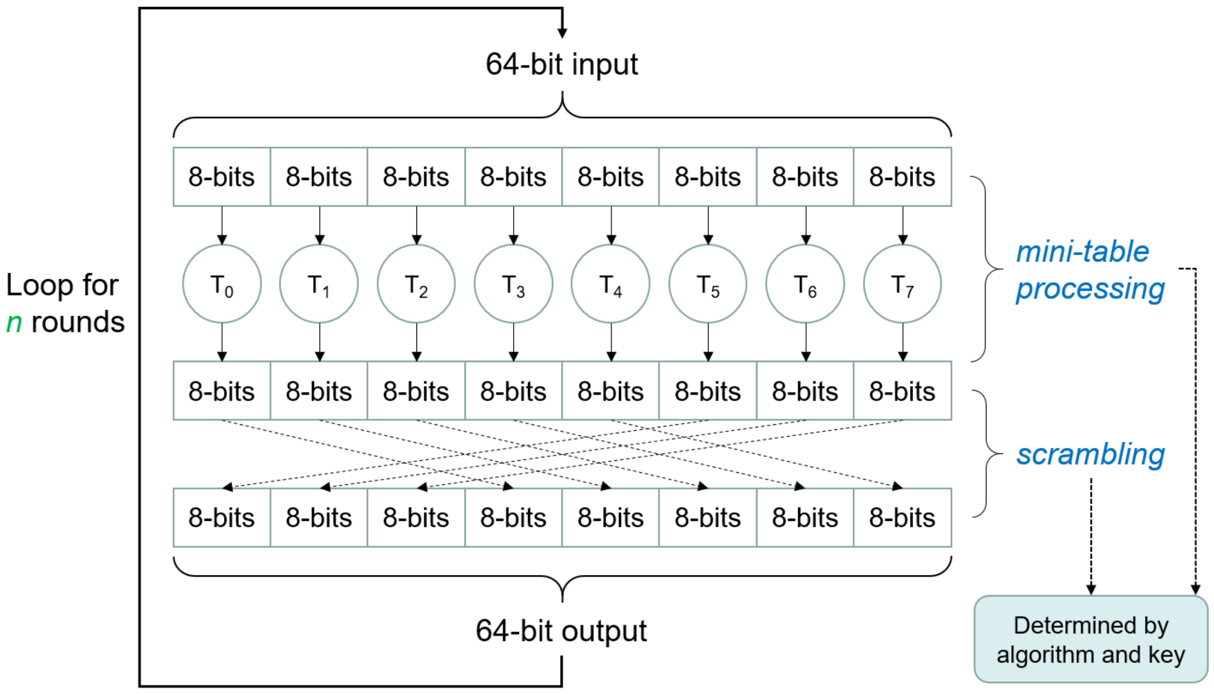


Fig. 1. A simplified block cipher.

Problem A

The proposed procedure consists of two functions, a *mini-table processor* and a *scrambler*. The mini-table processor function first breaks a 64-bit block into eight 8-bit chunks. Each chunk is processed by an 8-bit to 8-bit mini-table, which is of manageable size. Then, the positions of the 8 output chunks are scrambled to produce a 64-bit output. This output is fed back to the mini-table processor function to start another cycle. After *n* such cycles, the procedure provides a 64-bit block of ciphertext.

The eight mini-tables are determined in the following steps. First, it is assumed that there is a predetermined base mini-table with 2^8 rows, each of which has only one filed, *output*. For the *i*-th row, its output value is $0\text{xff} - i$, where *i* ranges from 0 to 0xff. Compared to Table 1, the proposed base mini-table replaced the inputs with the indexes, i.e. the 0th, 1st, 2nd, …, and 255th, as shown in Figure 2 to reduce the table size. Second, an offset value, *o*, is applied to the base mini-table to move the first *o* rows to the end of the base mini-table for generating a new mini-table. An example with *o*=2 is illustrated in Figure 2. Note that the base mini-table remains unchanged after the above operation. Third, a shared key agreed by the sender and its receiver is used to provide eight offset values for the eight mini-tables. The proposed algorithm uses a 72-bit key. The *i*-th 8-bit of the key determines the offset value for generating the *i*-th mini-table, where *i* ranges from 0 to 7. For example, if the first 64-bit of the key was 0x0000000002000200, the mini-tables T_6 and T_4 will be the same as the new mini-table in Figure 2 while the remaining six mini-tables will be identical to the base one.

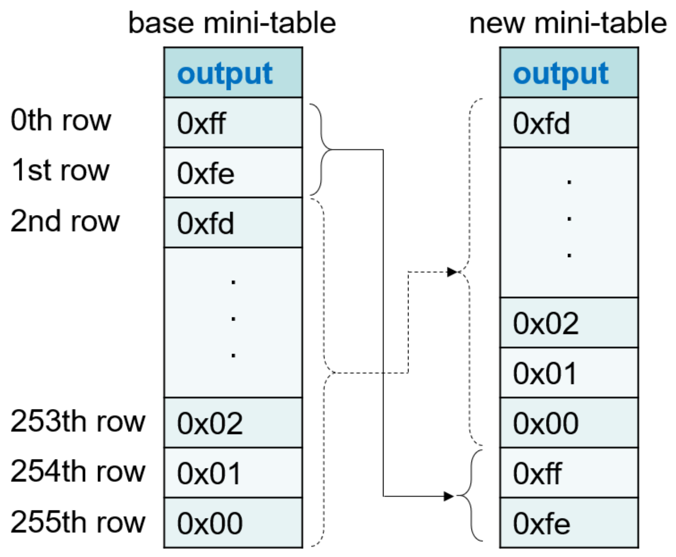


Fig. 2. An example of the generation of a new mini-table with offset value *o*=2.

In the mini-table processor function, if the value of the 8-bit input chunk is *x*, the value of the 8-bit output chunk will be the output value of the *x*-th row of the corresponding mini-table.

Problem A

Continuing with the above example, if the 64-bit input message is 0x0101010100000000, the values of the eight 8-bit output chunks will be 0xfe, 0xfe, 0xfe, 0xfe, 0xfd, 0xff, 0xfd and 0xff, respectively.

In Figure 1, assume that the 8-bit output chunks processed by mini-table T_0 and T_7 are the most significant chunk and the least significant chunk, respectively. In the scrambler function, the *s* least significant 8-bit chunks will be inserted into the positions of the *s* most significant ones. An example with *s*=3 is illustrated in Figure 1. Besides, the most significant 4-bit value of the last byte of the shared key determines the above value *s*. Then, the above procedure will repeat *n* times which is determined by the least significant 4-bit value of the last byte of the shared key. For example, if the last byte of the shared key was 0x38, *s* and *n* will be 3 and 8, respectively.

Finally, for ease of understanding, the data structure of the key in the proposed block cipher is depicted in Figure 3. Note that the 4-bit value of variable *n* will never be 0000. Besides, the most significant bit of variable *s* is useless in the proposed simple scrambler but may be useful for other sophisticated ones. Thus, the above bit will be set as 0 in this problem. In other words, the 4-bit value of variable *s* will be 0XXX where X is either 0 or 1.

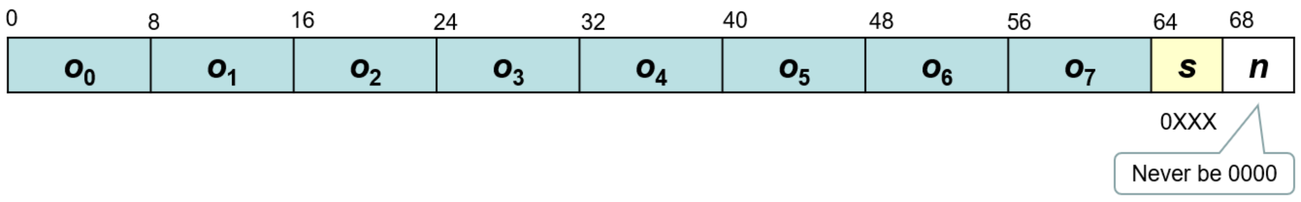


Fig. 3. The data structure of the key in the proposed block cipher.

Input Format

The first line is a decimal number indicating the number of test cases. For each test case, there is a line for the key, then followed by a line for the message to be encrypted. The key is a string containing 18 hexadecimal digits while the message is a string containing *m* hexadecimal digits. Note that a-f is used to represent the decimal values 10-15.

Output Format

For each test case, output the ciphertext in one line. Each ciphertext contains only hexadecimal digits and a-f is used to represent the decimal values 10-15. The length of each ciphertext will be the same as the length of its corresponding cleartext.

Technical Specification

- The maximum value of *m* is 2048. That is, there are at most 128 64-bit blocks of cleartext in one

Problem A

test case.

- The value of m is made by multiplying 16 by positive integers ranging from 1 to 128.
- The value of s ranges from 0 to 7.
- The value of n will not be 0.
- There are at most **10** test cases.

Example

Sample Input	Sample Output
7 0000000000200020001 0101010100000000 0000000000200020031 0101010100000000 0000000000200020032 0101010100000000 0000000000200020032 0101010100000000000123456789abcdef 94212530aa5843d901 aabbccddeeff0122 94212530aa5843d911 aabbccddeeff0122 94212530aa5843d912 aabbccddeeff0122	fefefefefdfffdff fffdfffefefefefd 01ff0200020001ff 01ff0200020001ff45658babcfef0121 c1230ef267a8bb04 04c1230ef267a8bb 6b671db7c1634014

Problem B
String to Palindrome

(Time Limit: 3 seconds)

A palindrome is a word, phrase, or sentence reads the same backward or forward. For example, radar is a palindrome because if we try to read it from backward, it is same as forward. In this problem you are asked to convert a string into a palindrome with minimum number of edit operations, including:

- Insertion: insert a single character at any position
- Deletion: delete any character from any position
- Substitution: replace any character at any position with another character

There are different ways to convert a string to palindrome. For example, to convert "abccda" you would need at least two operations if you use insertion operation only, that is, abccda → abdccda → abdccbda. But you can do it with only one replacement operation, that is, abccda → adccda. The minimum number of edit operations to convert "abccda" into a palindrome is 1.

Input Format

The first line is an integer *n* indicating the number of test cases. Each test case consists of a string to convert to palindrome. The input for each test case consists of a string containing lower case letters only. Each test case occupies exactly one single line, without leading or trailing spaces.

Output Format

For each test case, print the minimum number of edit operations needed to turn the given string into a palindrome on a single line.

Technical Specification

You can safely assume that the length of this string will not exceed 1000 characters.

Problem B

Example

Sample Input	Sample Output
6	
onlevel	2
rotation	2
coloring	4
redumsirismurded	2
ididdudi	1
tacocopa	2

Problem C
Valley in a Sequence of Numbers

(Time Limit: 3 seconds)

Anne is fond of numbers. One day she was given a sequence of 32-bit integers $a_1, a_{2,} a_3, \ldots, a_n$ where $a_{n-1} = a_n$ and $1000 \geq n \geq 2$ She was thinking of re-ordering these numbers into a new sequence $b_1, b_{2,} b_3, \ldots, b_n$ such that $b_1 \geq b_n \geq b_2 \geq b_{n-1} \geq b_3 \geq b_{n-2} \geq \cdots \geq b_{n/2} \geq b_{n/2+1}$ for an even n or $b_1 \geq b_n \geq b_2 \geq b_{n-1} \geq b_3 \geq b_{n-2} \geq \cdots \geq b_{(n+1)/2+1} \geq b_{(n+1)/2}$ for an odd n. For example, given a sequence of integers 5, 2, 3, 4, 5, 1, 1 Anne would like to re-order this sequence of numbers into 5, 4, 2, 1, 1, 3, 5. Note that the last two numbers in the given sequence are the same. This signals an end of a given sequence.

Input Format

The first line gives the number of test cases. It is then followed by the sequence of integers for each test case. The numbers are separated by whitespace(s). The input of each test case ends when two consecutive numbers read from standard input are the same. The input of a test case may take several lines. There are at most 20 test cases.

Output Format

For each test case, Anne should print out the re-ordered sequence of integers. Each line should contain 30 numbers except the last line which could contain at most 30 numbers.

Example

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

Problem D
Correctness of Multiplication

(Time Limit: 3 seconds)

The multiplication result of two unsigned integers in a CPU is not always correct. For example, in a 32-bit CPU, $2147483647 \times 2147483647 = 1$ is not the same as natural number multiplication. The reason why $2147483647 \times 2147483647 = 1$ is because the result of multiplication instruction is the result of natural number multiplication modulo by $2^{32} = 4294967296$.

For example, in an n-bit CPU, there are 2^n unsigned integers representing 0 to $2^n - 1$. Let $\otimes$ be the unsigned integer multiplication operator in an n-bit CPU, and $\times$ be the natural number multiplication. We have $u \otimes v = u \times v \ mod \ 2^n$.

$\otimes$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	4	6	8	10	12	14	0	2	4	6	8	10	12	14
3	0	3	6	9	12	15	2	5	8	11	14	1	4	7	10	13
4	0	4	8	12	0	4	8	12	0	4	8	12	0	4	8	12
5	0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11
6	0	6	12	2	8	14	4	10	0	6	12	2	8	14	4	10
7	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9
8	0	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8
9	0	9	2	11	4	13	6	15	8	1	10	3	12	5	14	7
10	0	10	4	14	8	2	12	6	0	10	4	14	8	2	12	6
11	0	11	6	1	12	7	2	13	8	3	14	9	4	15	10	5
12	0	12	8	4	0	12	8	4	0	12	8	4	0	12	8	4
13	0	13	10	7	4	1	14	11	8	5	2	15	12	9	6	3
14	0	14	12	10	8	6	4	2	0	14	12	10	8	6	4	2
15	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Take a 4-bit CPU as an example. You can write a program to get the above multiplication table. We have $u \otimes v = u \times v$ in gray cells, but $u \otimes v \neq u \times v$, otherwise. So, the correctness of n-bit unsigned integer multiplication $CM(n)$ is defined as the number of the cases $u \otimes v = u \times v$ in the multiplication table divided by $2^n \times 2^n$. So, we have $CM(4) = \frac{76}{2^4 \times 2^4} = 0.29687\cdots$. Moreover, we have $CM(32) = 0.0000000056659665\cdots$. $CM(32)$ is a very small number. But usually, we have

Problem D

no deep sense of this observation. That is because usually we only deal with multiplications of small numbers for $u, v \in \{0,1,2,3,\cdots,m-1\}$. The correctness $CM(n,m)$ is thus redefined as the number of the cases $u \otimes v = u \times v$ with $u, v \in \{0,1,2,3,\cdots,m-1\}$ divided by m^2. Therefore, we have $CM(n) = CM(n, 2^n)$. According to the above table, we can get $CM(4,4) = \frac{16}{4\times4} = 1$, and $CM(4,10) = \frac{52}{10\times10} = 0.52$. Now giving you three integers n, m, and d, please round $CM(n,m)$ down to d decimal places. For examples, you have to print 1.000 for $(n,m,d) = (4,4,3)$, 0.5 for $(n,m,d) = (4,10,1)$, and 0.296 for $(n,m,d) = (4,16,3)$.

Input Format

The first line is an integer indicating the number of test cases. Each test case consists of 3 integers n, m, and d.

Output Format

For each test case, please round $CM(n,m)$ down to d decimal places.

Technical Specification

- There are at most 10 test cases.
- $1 \le n \le 42$
- $1 \le m \le 2^n$
- $1 \le d \le 128$

Example

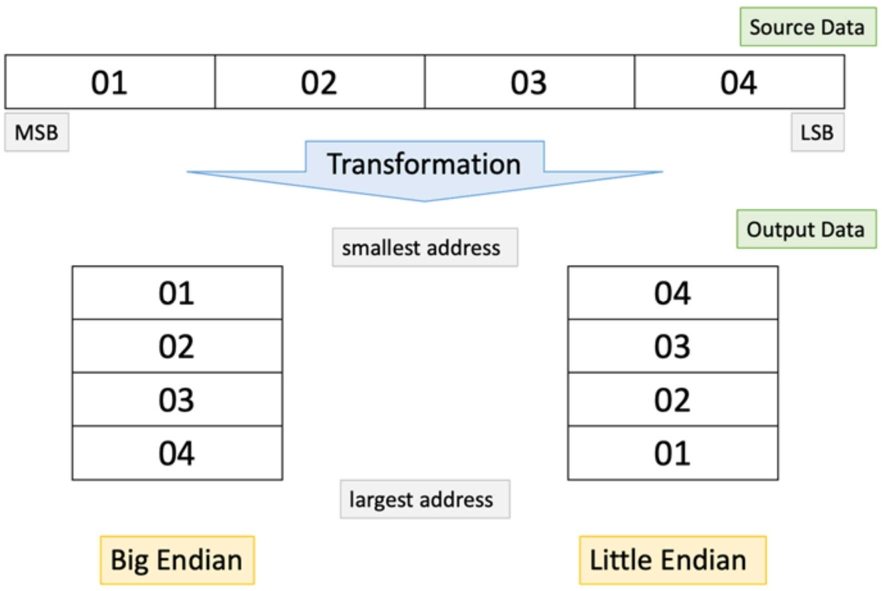
Sample Input	Sample Output

4
4 4 3
4 10 1
4 16 3
32 4294967296 16
	1.000
	0.5
	0.296
	0.0000000056659665

Problem E
Endianness Transformation

(Time Limit: 1 second)

The byte order is a critical issue of data presentation, and there are big-endian and little-endian formats. The little-endian system stores the least-significant byte (LSB) at the space with smallest address while the most-significant byte (MSB) is saved at the space with largest address. The figure shown below is an example for little-endian and big-endian systems for the given data "01020304." Little-endian and big-endian systems output the results "04030201" and "01020304" respectively.



Input Format

The first line is an integer n indicating the number of test cases where $1 \leq n \leq 10$. Then, the user would input n strings (one string for one line) to specify the input data of test cases. Each test case has nine characters, and they are divided into two parts. The first part that is the first character is a binary number for the target system, where 1 represents big-endian and 0 for little-endian. The second part (from second character to ninth character) is the source data that consists of four pairs of hexadecimal numbers. For example, "00000000" is the minimum value of source data and "FFFFFFFF" is the maximum one.

Problem E

Output Format

Given a sequence of input with one-bit target system and eight-bit hexadecimal numbers, the program prints the hexadecimal numbers in little-endian or big-endian format. In the above example, the result should be “01020304” for big-endian format with input value “101020304”and “04030201” for little-endian format with input “001020304.”

Technical Specification

- Each test case consists of one binary number and eight hexadecimal numbers. Therefore, the first bit should be zero or one, and the numbers from second position to the last position should be within zero and F.
- The number of test cases n ≤ 10.

Example

Sample Input	Sample Output
5 001020304 101020304 000000000 1FFFFFFFF 010203040	04030201 01020304 00000000 FFFFFFFF 40302010

Problem F
A type of coffee

(Time Limit: 1 seconds)

An e-commerce platform was sponsored by some coffee manufacturers and it is going to launch a coffee giveaway event. If a user name, registered on the platform, is included the number of different letters which are used a multiple of 3, e.g. “Williams”, the prize will be a little pack of instant coffee. If the number of different letters of the user name is 1 less than a multiple of 3, e.g. “Tomorrow”, a little pack of hanging ear drip coffee. I the number of different letters of the user name is 2 less than a multiple of 3, e.g “Kissinger”, a cup of Americano coffee from the convenience store.

Input Format

The first line is an integer *n* indicating the number of user names. The next *n* lines contain *n* non-empty strings, that contain only English letters. The user names are case-insensitive - upper and lower letters are treated as equal.

Output Format

If the user ID gets the instant coffee, print "INSTANT COFFEE!" (without the quotes); if it gets the hanging ear drip coffee, print "HANGING EAR!" (without the quotes), otherwise, print "AMERICANO COFFEE!" (without the quotes).

Technical Specification

- The number of test cases $n \le 10$.
- Every string contains at most 100 letters.

Example

Sample Input	Sample Output
4 Spring Miriam Kissinger MoreAndMoreString	 INSTANT COFFEE! AMERICANO COFFEE! AMERICANO COFFEE! HANGING EAR!

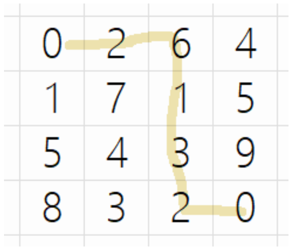
Problem G
Finding a minimum weighted path in a grid

(Time Limit: 1 second)

We have a matrix/grid/array with size of *k* by *k*. Each cell is populated by a nonnegative integer. The start cell and the end cell have values of zero. You start from the upper left corner and need to get to the bottom right. You can move only to adjacent cells either by sliding right, left, up or down (and not leaving the array). The goal is to find the path which makes the sum of values you've passed the lowest. An example of such 4x4 grid is shown in the figure below.

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

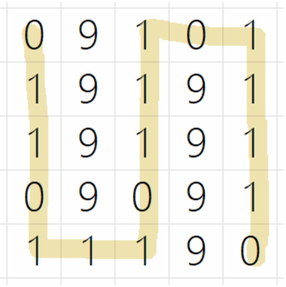
The solution for that specific grid would be **14** and the path is marked on the following image:



Another example of a 5x5 grid is shown in the figure below.

0	9	1	0	1
1	9	1	9	1
1	9	1	9	1
0	9	0	9	1
1	1	1	9	0

The solution for that specific grid would be **12** and the path is marked on the following image:



Problem G

Input Format

The first line is an integer n indicating the number of test cases. Each test case consists of a positive integer k; subsequently, it is followed by k rows of integers with each row consisting of k nonnegative integers. These k^2 nonnegative integers are the values of cells in the grid.

Output Format

The output for each test case prints out the nonnegative integer as the minimum total cost in the optimum (minimum weighted) path.

Technical Specification

- The number of test cases $n \leq 10$.
- The grid size $k \leq 20$.
- For each test case the value of each cell in the grid is a nonnegative integer v; $0 \leq v \leq 99$.

Example

Sample Input	Sample Output

3
2
0 7
3 0
4
0 2 6 4
1 7 1 5
5 4 3 9
8 3 2 0
5
0 9 1 0 1
1 9 1 9 1
1 9 1 9 1
0 9 0 9 1
1 1 1 9 0
	3
	14
	12

Problem H
Inverse of Rows and Columns

(Time Limit: 3 seconds)

For a given binary matrix X of size m×n. A binary matrix is a matrix where each element is either 0 or 1. You may perform some (possibly zero) operations with this matrix. During each operation you can change all values of the row (or a column), i.e. 0 to 1 or 1 to 0. For example, "inversing" a row is changing all values in this row to the opposite (0 to 1, 1 to 0). Inversing a column is changing all values in this column to the opposite.

Next, your task is to inverse the initial matrix to a matrix $\tilde{X}$ with size (m×n). The matrix is considered sorted if the element of this matrix $\tilde{X}$, i.e., $[\tilde{X}_{1,1}, \tilde{X}_{1,2}, \cdots, \tilde{X}_{1,n}, \tilde{X}_{2,1}, \tilde{X}_{2,2}, \cdots, \tilde{X}_{2,n}, \cdots, \tilde{X}_{m,n-1}, \tilde{X}_{m,n}]$ is sorted by the above inversing operators in non-increasing order.

Look at the following example of the inverse of rows and columns, a given binary matrix $X_{2x2}=\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}$, this matrix is increasing order. However, it can use the above inversing operation to get the sorted matrix $\tilde{X}=[1 \quad 0 \quad 0 \quad 0]$ by inversing the 1st and 2nd columns. So, it can be obtained a non-increasing order matrix from the inverse of rows and columns, and the result is YES.

Input Format

The first line is an integer indicating the number (≤ 10) of cases. For each case, the input contains two integers m and n (1≤ m, n ≤200) — the number of rows m and the number of columns n in the matrix.

The next *n* lines contain *m* integers each. The j-th element in the i-th line is $X_{i,j}$ ($X_{i,j}$ = 0 or 1).

Output Format

If it is possible to obtain a sorted matrix with non-increasing order, print "YES". Otherwise, print "NO".

Technical Specification

- m and n are both integers, 1≤ m, n ≤200.

Problem H

Example

Sample Input	Sample Output
10	YES
1 3	NO
0 1 1	YES
3 3	NO
0 0 0	YES
1 0 1	NO
1 1 0	YES
2 2	NO
0 1	YES
1 1	NO
4 4	
1 0 1 0	
0 1 0 1	
1 0 1 1	
0 1 0 0	
2 4	
1 0 1 0	
0 1 1 0	
3 3	
0 0 1	
1 0 1	
1 0 0	
3 4	
0 0 0 1	
0 0 0 0	
1 1 1 1	
4 5	
0 0 0 0 0	
0 1 0 1 0	
1 0 0 1 1	
0 1 0 0 1	
4 5	
0 1 1 1 1	

Problem H

1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
6 6
1 1 1 1 1 1
1 0 1 1 0 1
0 1 0 0 1 0
1 0 0 1 0 1
0 1 1 0 1 1
1 0 0 0 0 0	

Problem I
Longest Alternating Subsequence

(Time Limit: 1 second)

The longest alternating subsequence is a problem of finding a subsequence of a given sequence in which the elements are in alternating order and in which the sequence is as long as possible.
A **subsequence** of a given sequence is a sequence that can be derived from the given sequence by deleting some or no elements without changing the order of the remaining elements. For example, the sequence <A, B, D> is a subsequence of <A, B, C, D, E, F> obtained after removal of elements C, E and F. The relation of one sequence being the subsequence of another is a preorder. In order words, we need to find the length of the longest subsequence with alternate low and high elements.
Here, alternating sequence means a sequence $a_1, a_2, a_3, \dots a_n$ is called alternating if it follows any one of the conditions given below.

$a_1 < a_2 > a_3 < \dots a_n$

-OR-

$a_1 > a_2 < a_3 > \dots a_n$

For example, consider array A[] ={ ‘C’, ‘o’, ‘m’, ‘p’, ‘u’, ‘t’, ‘e’, ‘r’}. The longest alternating subsequence length is 6, and the subsequence is [‘C’, ‘o’, ‘m’, ‘p’, ‘e’, ‘r’] as (‘C’ < ‘o’ > ‘m’ < ‘p’ > ‘e’ < ‘r’).

Input Format

The input data must be a text with no white space.

Output Format

The length of the longest subsequence with alternate low and high elements.

Technical Specification

- The input text can only contain alphabet and number (maximum input length 64).
- All of elements in input sequence are comparing by ASCII code.

Problem I

Example

Sample Input	Sample Output
Computer	6
123456789	2
13572468	4
23323456AsIa	6